SECURE HARDWARE RESOURCE MONITORING, USAGE OPTIMIZATION AND AFFIRMATION FOR DATABASE OPERATIONS IN VIRTUALIZED CLOUD ENVIRONMENT

TAN CHEE HENG

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENT
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2014

UNIVERSITY MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **TAN CHEE HENG** **(**I.C/Passport No: **751022-08-5517)**

Registration/Matric No: **WHA060001**

Name of Degree: **DOCTOR OF PHILOSOPHY**

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"): **SECURE HARDWARE RESOURCE MONITORING, USAGE OPTIMIZATION AND AFFIRMATION FOR DATABASE OPERATIONS IN VIRTUALIZED CLOUD ENVIRONMENT**

Field of Study: **DATA MANAGEMENT**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name:
Designation:

ABSTRACT

Hardware resource management is an important topic in Information Technology (IT) industry. This is due to the increasing demand of computing power by ever-evolving applications, especially those which are Service Level Agreement (SLA)-bound. Undeniably, hardware cost has reduced significantly in recent time. However this does not translate into saving in capital and operational costs of businesses as the computing resource requirement from new applications overwhelms the reduction in hardware cost. Hence, cloud computing paradigm evolved from conventional grid and utility computing, to provide for the aggressive computational demands. To better serve the hosting in cloud environments, particularly in industries where data sensitivity and privacy is of major concern, better mechanisms are needed in the resource management arena. The proposed mechanisms in this research avoided access to real data in the database, to meet the objectives of effective hardware resource administration.

Here, the hardware resource management in virtualized cloud environment is scrutinized. The topics of interest are in the area of resource utilization *monitoring*, *optimization* and *affirmation*. The proposed mechanisms provide alternatives to conventional methods which are commonly adopted by the wide IT industry today. The target is to provide more simplified approaches to these conventional tools, with faster and more accurate attributes in sight.

In resource utilization *monitoring* area, metadata of the actual data is characterized to yield an understanding of the workload in the database, which then contributes to the decision in planning for hardware provisioning and de-provisioning activities, as well as resource scaling arrangement.

Consequently, a mechanism is proposed to serve the resource utilization *optimization* objective. In this research area, hardware fault and failure analysis are investigated, in

order to provide an optimal operating environment to database transactions. The analysis on the hardware fault and failure symptoms is performed against the output obtained from the iterative execution of Transaction Processing Performance Council (TPC)-H queries. Baseline is established and parameters' values obtained from subsequent testing on the same set of queries are compared to baseline's values to obtain insightful information on the hardware state.

Next, the resource utilization *affirmation* theme deals with the proposition to establish stress-testing scenario in the Virtual Machine (VM). The work here strives to construct an environment in the VM whereby validation on transactions' response time can be performed at the hypothetical resource constraining point in the VM. It serves this validation purpose in 2 situations: when the VM undergoes hardware change, or during normal operations. Verification is performed by stressing the VM to the resource constraining point using the proposed method; subsequently SLA-bound transactions are sent to the database and their respective response time is examined and compared to expected response time. The proposed mechanism also incorporates technique to determine the resource threshold from database transactions perspective.

The resource utilization *monitoring* utilizes metadata from representative workload, whereas the resource utilization *optimization* and *affirmation* mechanisms utilize the hypothetical data and queries from TPC-H benchmark, hence achieving the objective of eluding access to real data. These deliveries focus on the consistency, stability and accuracy attributes.

ABSTRAK

Pengurusan sumber perkakasan komputer merupakan satu topik penting dalam industri Teknologi Maklumat. Ini disebabkan oleh permintaan kuasa pemprosesan maklumat yang semakin meningkat, terutamanya daripada aplikasi yang mengalami evolusi yang pesat and terikat kepada Perjanjian Tahap Perkhidmatan yang ketat. Tidak dapat dinafikan, bahawa kos perkakasan komputer telah dikurangkan secara ketara dalam masa kebelakangan ini. Walaupun sedemikian, ini tidak dapat diterjemahkan kepada penjimatan dalam modal dan kos operasi perniagaan, kerana keperluan sumber pengkomputeran daripada aplikasi baru mengatasi pengurangan kos perkakasan komputer. Oleh itu, paradigma pengkomputeran awan berkembang daripada grid konvensional dan pengkomputeran utiliti, untuk mewujudkan peruntukan kuasa pengkomputeran demi memenuhi permintaan yang agresif ini. Dalam persekitaran awan, terutamanya dalam industri di mana sensitiviti dan privasi data merupakan faktor penting, mekanisme yang lebih baik diperlukan dalam arena pengurusan sumber pegkomputeran. Mekanisme yang dicadangkan dalam kajian ini mengelakkan akses kepada data sebenar dalam pangkalan data, sementara memenuhi objektif pentadbiran sumber perkakasan komputer yang berkesan.

Kajian ini meneliti tentang pengurusan sumber perkakasan komputer dalam persekitaran pengkomputeran awan. Topik-topik yang dikaji adalah dalam bidang pemantauan, pengoptimuman dan pengesahan penggunaan sumber pemprosesan komputer. Mekanisme yang dicadangkan menyediakan alternatif kepada kaedah konvensional yang luas dipraktikkan oleh industri IT kini. Sasaran objektif adalah untuk memberi pendekatan yang lebih mudah berbanding dengan kaedah-kaedahkonvensional; sementara sifat-sifat seperti lebih cepat dan lebih tepat juga merupakan sasaran kajian tersebut.

Dalam subjek pengurusan sumber pemprosesan komputer, topik pemantauan penggunaan meneliti metadata data sebenar untuk menghasilkan pemahaman tentang beban kerja dalam pangkalan data. Pemahaman ini kemudiannya menyumbang kepada keputusan dalam perancangan penambahan atau pengurangan perkakasan pengkomputeran.

Selepas topik pemantauan dikaji, topik seterusnya adalah untuk menghasilkan satu mekanisme untuk memenuhi objektif pengoptimuman penggunaan sumber pemprosesan komputer. Dalam kawasan kajian ini, kegagalan perkakasan pengkomputeran dan analisis kemungkinan kegagalan disiasat. Activiti sedemikian amat penting supaya transaksi pangkalan data dapat beroperasi dalam persekitaran yang optimum. Thesis ini menjalankan ujikaji analisa ke atas kegagalan and kemungkinan kegagalan perkakasan pengkomputeran dengan menggunakan output yang diperolehi daripada Institusi TPC. Data yang digunakan adalah daripada standard TPC-H. Garis dasar diasaskan dan nilai-nilai parameter yang diperolehi daripada ujian berikutnya pada set data yang sama berbanding dengan nilai-nilai garis dasar, akan memberikan penjelasan mengenai keadaan operasi perkakasan pengkomputeran.

Kajian seterusnya berobjektif untuk mengesahkan keupayaan sumber pemprosesan komputer di persekitaran pengkomputeran awan. Eksperimen yang dilaksanakan bertujuan untuk mengasaskan persekitaran sistem operasi yang menggunakan sumber pemprosesan komputer secara agresif. Selepas keadaan sedemikian berlaku, transaksi penting diperkenalkan kepada pengkalan data, supaya tempoh pemprosesan dan tindak balas bagi transaksi berkenaan dapat dibandingkan dengan jangkaan. Keperluan pengesahkan sedemikian boleh dilakukan dalam 2 keadaan: Apabila sistem operasi mengalami situasi perubahan dalam kuantiti and kualiti perkakasan, atau semasa keadaan pengoperasian biasa di mana tiada perubahan langsung dalam perkakasan pengkomputeran. Pengesahan dilakukan dengan menekan sistem operasi ke tahap yang

mengekang sumber pemprosesan komputer. Mekanisme yang dicadangkan juga menggabungkan teknik untuk menentukan had kuasa perkakasan pengkomputeran, daripada persepsi transaksi pangkalan data.

Pemantauan penggunaan sumber pemprosesan komputer menggunakan metadata dari beban kerja yang berupaya mewakili keadaan sebenar, manakala mekanisme pengoptimuman dan pengesahan penggunaan sumber pemprosesan komputer menggunakan data hipotesis dan data daripada standard TPC-H. Sumber input sedemikian dapat mencapai objektif untuk mengelakkan akses kepada data sebenar. Tumpuan ujikaji tersebut memberi penekanan terhadap ketekalan, kestabilan dan ketepatan sementara mencapai objektif yang dekehendaki.

ACKNOWLEDGEMENT

First and foremost I would like to express my special appreciation to my supervisor, Associate Professor Dr. Teh Ying Wah, you have been a tremendous mentor to me. I would like to thank you for your patient in this lengthy journey of my pursuit, and for improving and refining my views in many topics. Your perspectives and advices will become my valuable assets in my life journey.

A special thanks to my family. Words cannot express how grateful I am to my wife, Lee Choo, for her faith in me, and her incredible patience with me. Words cannot describe my love to my daughter, Ke Ying, my boys, Yong Hung, Yong Keat and Yong Jen. You have made my everyday life immensely joyous and fullfilling. Lastly, I would like to express my gratitude to my parents for their relentless support. Your kindness and confidence are critical elements for the completion of this thesis.

TABLE OF CONTENTS

ABSTRACT ..... **iii**

ABSTRAK ..... **v**

ACKNOWLEDGEMENT ..... **viii**

TABLE OF CONTENTS ..... **ix**

LIST OF FIGURES ..... **xv**

LIST OF TABLES ..... **xxi**

LIST OF ABBREVIATIONS AND ACRONYMS ..... **xxiii**

1. INTRODUCTION **1**

1.1 Background: Secure Resource Management 1

1.2 Overview of cloud computing 4

1.3 Motivation 12

1.4 Problem statements 16

1.4.1 SLA compliance and monitoring systems 16

1.4.2 Dynamic scalability issue for *Parallel Database* 17

1.4.3 Continuous fault analysis 17

1.4.4 Shortcoming of benchmarks 18

1.4.5 Data security issue 18

1.4.6 Insufficient measurement methods 19

1.5 Current practices 20

1.6 Research questions 22

1.7 Research objectives 24

1.8 Scope of research 25

1.9 Chapter organization 26

2. LITERATURE REVIEW **28**

2.1 Introduction 28

2.1.1 Resource utilization *monitoring* 30
2.1.2 Resource utilization *optimization* 31
2.1.3 Resource utilization *affirmation* 33
2.2 Virtualized cloud infrastructure 33
2.3 Data security 44
2.4 Resource utilization *monitoring* 57
2.4.1 Monitoring models and on-demand resource scaling 57
2.4.2 Resource scalability in *Parallel Database* architecture 69
2.4.3 Statistical modeling and benchmarking 74
2.4.3.1 Proof of concept – the linear correlation 74
2.4.3.2 Mathematical models 76
2.4.3.3 Linear regression 77
2.4.3.4 Machine learning 79
2.4.3.5 Fuzzy computing 80
2.4.3.6 Linear Programming and Simplex Method 82
2.4.3.7 TPC benchmark 86
2.4.4 Measurement methods 87
2.4.4.1 Hierarchical clustering 90
2.4.4.2 K-mean Clustering 91
2.4.4.3 Maximum Likelihood Estimation 92
2.4.4.4 Goodness of Fit 93
2.4.5 Workload characterization 96
2.5 Resource utilization *optimization* 103
2.5.1 Fault analysis and failure prediction 106
2.5.2 Resource utilization *optimization* models 114
2.5.2.1 Task scheduling 114

2.5.2.2 Auction-based resource scheduling 123
2.5.2.3 Resource brokering – the essence of *cloud bursting* 129
2.6 Resource utilization *affirmation* – stress testing 140
2.6.1 Conventional stress testing 140
2.6.2 I/O parameter 146
2.7 Summary and discussion 151
3. RESEARCH METHODOLOGY 154
3.1 Introduction 154
3.2 Approaches to research 156
3.2.1 Definition of research objectives 157
3.2.2 Proposed models 158
3.2.2.1 *Monitoring* model 158
3.2.2.2 *Optimization* model 159
3.2.2.3 *Affirmation* model 160
3.2.3 System design 162
3.2.3.1 The *monitoring* scheme 162
3.2.3.2 The *optimization* scheme 163
3.2.3.3 The *affirmation* scheme 163
3.2.4 Analysis of methods 165
3.2.4.1 *Monitoring* model 165
3.2.4.2 *Optimization* model 167
3.2.4.3 *Affirmation* model 168
3.3 Summary and discussion 169
4. SYSTEM DESIGN 170
4.1 Introduction 170
4.2 Overview of proposed models 170

4.3 Theme 1: *monitoring* scheme 173
4.3.1 Workload traces 173
4.3.2 Graphical representation – linear regression 176
4.3.3 Outliers effect 180
4.4 Theme 2: *optimization* scheme 182
4.4.1 Setup of environment 182
4.4.2 Linear regression and machine learning 186
4.4.3 Applicability to the production phase 188
4.5 Theme 3: *affirmation* scheme 190
4.5.1 Setup of stress-testing scenario 191
4.5.2 Application to production phase 196
4.5.2.1 Creation of stress testing scenario 197
4.5.2.2 Threshold verification 198
4.6 Potential improvement 200
4.7 Summary and discussion 206
5. EXPERIMENTAL RESULTS AND ANALYSIS 209
5.1 Introduction 209
5.2 Data sets 210
5.2.1 Metadata from real workload 210
5.2.2 Synthetic workload – TPC-H benchmark 212
5.3 Resource utilization *monitoring* 216
5.3.1 Environment for the experiments 216
5.3.2 Derivative parameters from the experiments 218
5.3.3 Experimental data 221
5.3.4 *Monitoring* model's accuracy and system overhead 224
5.3.5 Workload characteristic 225

5.3.5.1 SQL tracking 225
5.3.5.2 SQL optimization 226
5.4 Resource utilization *optimization* 228
5.4.1 Environment for the experiments 228
5.4.2 Experimental results 229
5.4.3 Potential deviation from the testing data sets 230
5.5 Resource utilization *affirmation* 233
5.5.1 Future work 236
5.6 Summary and discussion 237
6. CONCLUSIONS 239
6.1 Introduction 239
6.2 Summary of solutions to the objectives' questions 239
6.3 Limitations of current study 246
6.3.1 The *monitoring* scheme 246
6.3.2 The *optimization* scheme 247
6.3.3 The *affirmation* scheme 247
6.4 Recommendations and future directions 248
References 251
Appendix A 263
Appendix B 267
Appendix C 275
Appendix D 285
Appendix E 287
Appendix F 289
Appendix G 299
Appendix H 310

Appendix I 313

LIST OF FIGURES

Figure 1.1: Cloud computing models. 7

Figure 1.2: The service model in cloud computing.. 8

Figure 1.3: VMWare Virtualized Infrastructure.. 9

Figure 2.1: A summary of the research. 28

Figure 2.2: Virtualization Infrastructure diagram. 36

Figure 2.3: OpenNebula virtualization management software. 38

Figure 2.4: Eucalyptus platform. 39

Figure 2.5: Data hosting architecture proposed for hospital systems. 45

Figure 2.6: Top threats relevance. 48

Figure 2.7: Secure third-party publication. 51

Figure 2.8: Secure cryptoprocessor. 53

Figure 2.9: Patient medical record categories. 54

Figure 2.10: 'Patient-centric' cloud model. 55

Figure 2.11: Proposal of cloud hosting for medical data. 57

Figure 2.12: Throughput of the system with vertical scaling of resources. 60

Figure 2.13: CPU utilization in both the web and application VM during the test of vertical resource scaling. 60

Figure 2.14: Throughput keeps increasing with the addition of VM. 61

Figure 2.15: Dynamic scaling of web-based applications. 62

Figure 2.16: Markov Chain model.. 64

Figure 2.17: The utilization threshold versus SLA violation for particular workload. 67

Figure 2.18: Resource allocation to an application for an organization in cloud. 68

Figure 2.19: Relational Cloud architecture. 70

Figure 2.20: *Live Database Migration*.....72

Figure 2.21: Linear correlation between throughput and concurrent processing.....74

Figure 2.22: Linear correlation between SQL Processing Time, *S* and Server Load, *C*. 76

Figure 2.23: The application of methematical models into real world systems. 77

Figure 2.24: Linear Processing element. 77

Figure 2.25: A Fuzzy logic control system for resource utilization monitoring. 81

Figure 2.26: 3 membership functions in the resource utilization scale. 82

Figure 2.26: Image illustration of the 2 distance measurement method. 89

Figure 2.27: Hierarchical clustering. 91

Figure 2.28: K-Means Clustering. 91

Figure 2.29: Maximum Likelihood Estimate. 93

Figure 2.30: The chi-squared test of Goodness of Fit for linear regression. 94

Figure 2.31: Query plan and performance projections as the result of KCCA computation on the ‘distance metrics’ of *N* training queries. 95

Figure 2.32: Data Diffusion model. 99

Figure 2.33: Throughput vs # of nodes in a cluster. 100

Figure 2.34: Throughput vs file I/O size. 100

Figure 2.35: The graph representation of partitioning activity. 102

Figure 2.36: Time relations in *online failure prediction.* 108

Figure 2.37: Online failure prediction method based on classification of system variable observations. 110

Figure 2.38: Online failure prediction method based on pattern recognition. 110

Figure 2.39: Online failure prediction method based on *system models*..... 111

Figure 2.40: Fault detection strategy that filters out the normal signals, and leaves the outliers for analysis. 112

Figure 2.41: Task scheduling problem, with standby VM to service the 'abstract services'..... 117

Figure 2.42: Task scheduling system.. 118

Figure 2.43: ESCR task allocation algorithm. 118

Figure 2.44: Proposed resource optimization by introducing more efficient task scheduling in a particular VM. 122

Figure 2.45: Server busyness using server load limit as gauge..... 122

Figure 2.46: Optimization for supply and demand Auction-based resource allocation.. 124

Figure 2.47: First step in the VAM optimization. 124

Figure 2.48: The supply of resources from provider 2 has been allocated in full to consumer 2.. 125

Figure 2.49: The green colored cells indicate the resource allocation by the providers to each consumer. 126

Figure 2.50: Negotiation of services and pricing between the consumer and cloud provider. 127

Figure 2.51: The negotiation process for finite state of buyer's request and provider's resources. 128

Figure 2.52: Resource brokering model envisaged by Javadi et al.. 132

Figure 2.53: Resource brokering model envisaged by Javadi et al., with *cloud bursting* mechanism incorporated.. 134

Figure 2.54: *Cloud Hosting Provider* architecture envisaged by Fito et al. 138

Figure 2.55: Revenue benefit in deploying Hybrid Cloud computing compared to static servers. 138

Figure 2.56: *HP Loadrunner* Components. 141

Figure 2.57: Capacity calculator. 142

Figure 2.58: CPU utilization on the web and database servers from sampling of load testing carried out on a hypothetical online store. 144

Figure 2.59: CPU utilization on the web and database servers from sampling of load testing carried out on a hypothetical online store. 146

Figure 2.60: Comparison of I/O performance between direct and buffered I/O in virtualized clusters in Private Cloud. 148

Figure 2.61: Comparison of I/O performance between direct and buffered I/O in virtualized clusters in Private Cloud. 149

Figure 2.62: Throughput as a function of time. 150

Figure 3.1: Resource utilization *monitoring* model. 159

Figure 3.2: Resource utilization *optimization* model. 160

Figure 3.3: Resource utilization *affirmation* model.. 161

Figure 3.4: The creation flow in the resource utilization *monitoring.*. 162

Figure 3.5: The creation flow in the resource utilization *optimization.* 163

Figure 3.6: The creation flow in the resource utilization *affirmation.* 164

Figure 3.7: The second creation flow in the resource utilization *affirmation.* 165

Figure 4.1: Layered depiction of VMWare Virtualized Infrastructure for database hosting. 172

Figure 4.2: The relationship between DCT, SET and CPU run queue. 177

Figure 4.3: Response time fluctuation of a transaction in the VM. 179

Figure 4.4: Steps towards achieving the proposed monitoring scheme. 180

Figure 4.5: Steps to create the baselines and their applicability in production environment for *optimization* model. 184

Figure 4.6: Algorithm to load up the VM. 185

Figure 4.7: Metadata filtering to ensure stabilized condition in the VM before reliable data is collected. 186

Figure 4.8: The expected output for the *optimization* scheme. 187

Figure 4.9: The flow of the setup and application steps of the resource utilization *affirmation* model. 193

Figure 4.10: Array that stores the benchmark data for *affirmation* model. 194

Figure 4.11: Algorithm to produce the benchmark data. 195

Figure 4.12: Graphical result from the benchmarking experiment.. 195

Figure 4.13: Algorithm to produce the benchmark data using combination of TPC-H queries.. 196

Figure 4.14: Steps to discover resource threshold in the *affirmation* model. 200

Figure 5.1: High level depiction of the Oracle *Workload Repository* engine. 211

Figure 5.2: TPC-H data model. Adapted from (Kocakahin, 2010).....213

Figure 5.3: Experimental results that show relationship between $\tilde{S}_i$(SET), $\tilde{S}'_i$ (DCT) *and* C_i (CPU run queue size).....218

Figure 5.4: Membership Function for ΔS, $A(u)$.....222

Figure: 5.5. Membership Function for C, $B(v)$.....223

Figure. 5.6: Membership Function for ρ, $C(w)$. 223

Figure 5.7: Runtime Variation of Particular SQL in 1 Week. 227

Figure 5.8: *Parallel Database* hosting using *VMWare* Cloud Virtualization Infrastructure......228

Figure 5.9: The expected output from the *optimization* model.....229

Figure 5.10: Potential change in linear correlation between *S* and *C*. 230

Figure 5.11: Potential change in linear correlation between *S* and *C*. 232

Figure 5.12: Erratic behavior of hardware performance during backup process.....232

Figure 5.13: Testing result of the *affirmation* model.....233

Figure 5.14: Testing result with the combination of TPC-H queries.....235

LIST OF TABLES

Table 2.1: Cloud vs. Grid computing.. 41

Table 2.2: Summary of studied researches with critical comment on sub-themes 'monitoring models and resource scaling'. 73

Table 2.3: Tableau depicts the Simplex algorithm..... 85

Table 2.4: Summary of studied researches with critical comment on sub-themes 'statistical modeling and workload characterization'. 102

Table 2.5: Summary of studied researches with critical comment on sub-theme 'fault analysis and failure prediction'. 113

Table 2.6: Summary of studied researches with critical comment on sub-theme 'task scheduling'. 119

Table 2.7: Summary of studied researches with critical comment on sub-theme 'Auction-based resource scheduling'. 128

Table 2.8: Summary of studied researches with critical comment on sub-theme 'resource brokering – the essence of cloud bursting'..... 139

Table 2.9: Summary of studied researches with critical comment on sub-theme 'conventional stress-testing and I/O parameter'. 150

Table 4.1: Corresponding response time, memory and disk reads of TPC-H queries.. 184

Table 4.2: Attributes of the queries potentially involve in the construction of stress-testing scenario. 202

Table 4.3: Tabular representation of the dual objective function and constraints. 204

Table 4.4: The red colored cell depicts the pivot. 205

Table 4.5: The reduction of other cells' values to 0, in the pivot column. 205

Table 4.6: The continuous steps to produce the final resolution by the *simplex method.* 206

LIST OF ABBREVIATIONS AND ACRONYMS

ADDM	Automatic Database Diagnostic Monitor
AFR	Annual failure rate
AI	Artificial Intelligence
API	Application Programming Interface
ASM	Automatic Storage Management
AWRT	Average Weighted Response Time
AWS	Amazon Web Services
AWS	Amazon Web Services
CDA	Clinical Document Architecture
CHP	Cloud Hosting Provider
CRM	Customer Relationship Management
CSA	Cloud Security Alliance
DbaaS	Database-as-a-Service
DCT	DB CPU Time
DDoS	Distributed Denial of Service
DICOM	Digital Imaging and Communications in Medicine
EBS	Elastic Block Storage
EMR	Electronic Medical Records
ERP	Enterprise Resource Planning
FIPS	Federal Information Processing Standard
GFS	Google File System
GOF	Goodness of fit
GUI	Graphical User Interfaces
HA	High Availability
HER	Electronic Health Record

HIPPA	Health Insurance Portability and Accountability Act
HIT	HealthCare Information Technology
HPC	High Performance Computing
HRMS	Human Resource Management System
I/O	Input and Output
IaaS	Infrastructure as a Service
IGG	InterGrid gateway
IPS	Intrusion Prevention System
IT	Information Technology
ITAR	US International Traffic in Arms Regulations
KCCA	Kernel Canonical Correlation Analysis
MDP	Markovian Decision Processes
MLE	Maximum Likelihood Estimation
MNM	Meta-Negotiation Middleware
MOVR	Monitoring and Optimizing Virtual Resources
MSE	mean square error
OLTP	online transaction processing
OS	Operating System
OTF	Overload Time Fraction
PaaS	Platform as a Service
PDF	probability distribution function
PHI	Protected Health Information
PHR	Personal Health Record
PII	Personal Identity Information
QoS	Quality of Service
RAC	Real Application Cluster

RDBMS	Relational Database Management System
SaaS	Software as a Service
SCP	Secure cryptoprocessor
SET	SQL Elapsed Time
SLA	Service Level Agreement
SOA	Service Oriented Architecture
TCO	Total Cost of Ownership
TPC	Transaction Processing Performance Council
VAM	Vogel’s Approximation Method
VIE	Virtual Infrastructure engine
VM	Virtual Machine
VMM	Virtual Machine Manager
WAN	Wide Area Network
WHP	Web Hosting Provider
WSM	Web Service Monitoring

1. INTRODUCTION

Businesses of today depend on Information Technology to gain the competitive edge. Effective use of the technology is the main drive towards achieving growth and profitability. To stay ahead of competitions, businesses need to roll out new and innovative services faster, build more satisfying relationship with the customers, reduce capital and operational expenses, and make more efficient use of human resources in the companies. To promote the transformation to new technical methodologies, the security aspect must be contemplated in parallel with these innovative goals.

1.1 Background: Secure Resource Management

From the infrastructure perspective, effective and intelligent computing resource management is one way to promote such initiatives. Effective resource management has a profound impact on the feasibility of providing software services. The cost-effectiveness of the total application service offering is the primary barometric indicator to the continuation and evolution of the software. In this case, computing resource procurement and maintenance are major components in affecting the capital and operational expenditure.

Computing resources, or often termed system resources are components available in a computer system. They can be segregated into physical and virtual units. Processors, memory modules and disk drives are categorized as physical components. The memory swap area, filesystems and network connections are in the virtual category. They are interoperable to produce the desired computing outcome. Their availability is often limited and relatively costly for most organizations, hence accurate and careful deployment and usage are essential.

Definition 1.1: Resource Management. Resource management can generally be grouped into 3 phases. The first concerns with gaining visibility on the computing

resources in the system in order to discover abnormalities either in the end users' transactions or hardware state. The second obtains the input from the outcome of the monitoring mechanism, and action on the anomalies. The third involves the verification process, which provides an assurance on the computing resource performance, in order to guarantee consistency and optimality.

With question of "Why Resource management is important", the potential benefits are outlined as follows:

1) Computing resources are limited entities in a computer system. They can be relatively costly for enterprises. Over-provisioning of resources in a system is not cost effective, and under-provisioning could be disastrous to the business in mission-critical situations.
2) The computing hardware cost is steadily reduced overtime, however many new developments on new software are offsetting this benefits to the businesses, as the demand for computing power is so voracious, that often the cost to provide application services is increasing instead of reducing in the total cost of ownership.
3) Computing resources are failure-prone entities. They must be supervised continuously to ensure SLA-bound transactions can complete within stipulated durations.
4) Adequately sized computing hardware has great effect on carbon footprint in our environment. One of the methods to promote green computing is to reduce the usage of electricity, and this can be achieved by reducing wastage in operating the hardware.

Definition 1.2: Secure Resource Management. From IT perspective, security enhancement can be achieved at various levels. For instance, the existence of firewall protects the backend system components from malicious attackers who perform the

intrusion from outside of the organizations. The strengthening of authentication algorithms safeguards the applications and databases from insiders' abuses. The intensification of security monitoring against the software components prevents the attacks from outsiders and insiders. Of all the technologies deployed to provide such protections, they are all striving to protect the underlying data from the wrongful recipients. Hence, the security aspect in this research deals with the prohibition of data access from unnecessary personnel. In conducting the administrative works to manage the resources, IT staffs often gain superuser access to the databases, hence the real users' data. This is an undesired situation as the data can be compromised easily, with or without the knowledge of the organizations. Hence, this research aims to produce alternative mechanisms for resource management, which are different as compared to conventional practice, by discounting the requirement to grant access to real data for IT personnel. The approaches in this research strive to provide an answer to the intricate question in IT management of "How to securely perform IT administration".

Definition 1.3: Secure Resource Management in Virtualized Cloud Environment. Conventional resource management systems often use a centralized system and scheduler to monitoring the whole landscape. With the advancement of hosting technology, cloud computing paradigm has become the main architectural focus in constructing the hosting environment. As the cloud is distributed in nature, the proposed resource management mechanisms in this research are loosely coupled with a centralized monitoring system, where they can either be deployed to be managed centrally or in distributed manner. The proposed mechanisms also take the scalability and elasticity, which are the primary attractions in cloud computing as predominant considerations. The virtualized cloud environment is the focus in this research, as it often deployed for *Parallel Database* operations. Such hosting virtual architecture abstracts the hardware resources from physical servers, to create virtual machines which

provide hosting platform to various IT solutions. The following section provides a high level view on cloud computing.

1.2 Overview of cloud computing

As cloud computing serves as the focal point where the resource management proposals are targeted on, it is described to an introductory degree in this section. From infrastructure perspective, Cloud computing offers very attractive solutions to reduce cost and simplify IT management activities. Voted as Top 11 technologies of the decade by *IEEE Spectrum* (Upson, 2011), it offers new level of IT capability, through scalable, flexible and reliable models. It enables the agility required to accelerate the time to market of new products and services while reducing the cost to design, build, deploy and support these products and services, and is considered as generally best practice for Enterprise Architecture (Glas & Andres, 2011). Cloud computing has revolutionized the IT industry and is probably the most important paradigm ever modeled.

There are various perceptions in defining Cloud Computing. Zhang et al.(S. Zhang, Zhang, Chen, & Huo, 2010) defined Cloud Computing as an evolution of grid computing, and it comprises of thin clients, Grid Computing and Utility Computing. Buyya et al. (Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009) differentiated between Cloud Computing and Grid Computing at the *virtualization* level, where Cloud is defined as next-generation data centers with nodes “virtualized” through hypervisor technologies, dynamically "provisioned" on demand as a personalized resource collection. The virtualization provides the ease and flexible capability on resource allocation, which is the key motivation for this research. Foster et al. (Foster, Zhao, Raicu, & Lu, 2008) compared Cloud and Grid in length; and from dynamic resource provisioning perspective, Cloud is deemed more flexible than Grid, as Cloud is leveraging virtualization technologies more extensively. Zhang et al. (Q. Zhang, Cheng,

& Boutaba, 2010) defined Cloud Computing in a more end-user-friendly way, by quoting: *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction*. It is this ease of effort in application hosting that makes Cloud a popular and fascinating choice. The financial service firm Merrill Lynch estimated in 2008 that in the next five years, the annual global market for Cloud Computing would surge to $95 billion. In a May 2008 report, Merrill Lynch estimated that 12% of the worldwide software market would go to the Cloud in that period (King, 2008). Public Cloud vendors are building extremely large-scale, commodity-computer Data Centers in low cost locations, and they uncovered factors of 5 to 7 decrease in cost of electricity, network bandwidth, operations, software, and hardware available at these very large economies of scale (Armbrust et al., 2009).

As cloud promotes pay-per-use model, businesses can deploy critical applications without hampered by budgetary constraint in procurement of computing hardware and complex configuration of IT infrastructure. Furthermore, commodity servers are often utilized in cloud environment; hence reduce the hardware cost even further. This distributed computing platform is able to function with either homogeneous or heterogeneous hosts, in other words the servers that comprise the computing clusters do not necessary need to be identical. Hence, cloud computing not only enables much reduced total cost of ownership (TCO) during the construction phase, but it is also economical to be managed during on-going steady state phase. Another advantage of the pay-per-use model is that energy conservation is achieved, as electricity wastage can be reduced to minimum by this hosting model. In conventional standalone server hosting model, businesses generally end up using only somewhere between 8 and 20 percent of the servers' capacity that they have purchased (Gmach et al., 2008). Energy

consumption in global data centers in year 2010 accounted for between 1.1% and 1.5% of total electricity use, and 2% of global carbon footprint in year 2007. However as a prominent user of computing power in the world, Google's data centers which mainly serve Big Data operations via cloud paradigm only utilized less than 1% of total electricity consumed by all the data centers (Koomey, 2011; Pettey, 2007).

From maintenance perspective, standardized hardware and software are used in cloud environments, hence patching and other operational maintenance tasks can be performed in uniform and organized manner with less variety and complexity of IT components. The ease of maintenance results in significant reduction in operational overhead particularly in large organizations. This allows IT to transform its main focus from maintaining infrastructure services to building innovative services that connect to business goals and drive revenue. This is a revolutionary paradigm that enables IT to participate directly in business innovation, hence measurably fuel the business growth. The ease of scalability in the hosting resources allows companies to react faster to changing business needs. The cloud systems can scale-up and scale-out easily. Coupled with its flexible characteristic, the systems can be scaled-down when resources are no longer needed to service particular applications. Particularly in Public Cloud domain, as storage hardware is abundant, data can be partitioned and replicated to multiple storage locations to ensure high availability of services. Amid slower response time when the data is hosted in separate location compared to the application tier, this feature is beneficial for applications that have high availability requirement that overwhelms the fractional differences in response time experienced by the end users.

The 3 most popular deployment types for cloud computing are Public, Private and Hybrid Cloud. By the names, their architectural differences can be easily differentiated. Figure 1.1 depicts these models.

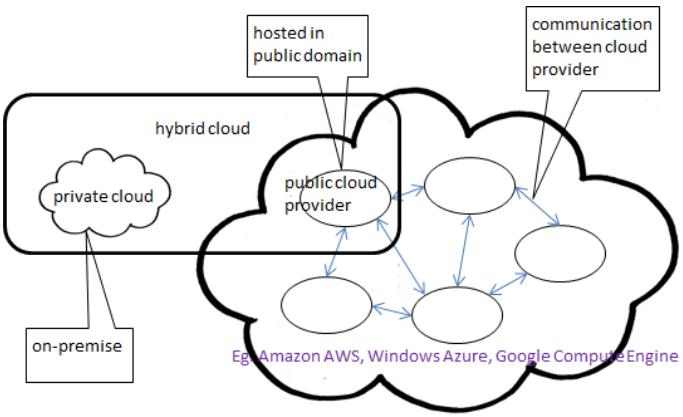


Figure 1.1: Cloud computing models. These are the 3 most popular deployment models in the industry.

There are 3 different layers of cloud services popularly adopted by the industry. They are:

- Infrastructure as a Service (IaaS): In this model, the cloud providers offer the clients with a set of virtual hardware environment. The providers own and maintain the underlying storage, servers, network components and other types of computing hardware. The operating system is managed by the clients (subscribers). In other words, the computing resources are provided as a service to the end users. The subscribers can either pay the services via *pay-per-use* model or by having a set of resources provisioned in a cluster of VM, which is easily scalable.
- Platform as a Service (PaaS): Cloud providers establish the hardware, virtual machines, and setup the operating system. Furthermore, depending on the requirement, the providers also deliver databases, web servers and other native utilities so that developers can use these facilities directly to run their developed software.
- Software as a Service (SaaS): In this standard, apart from installing the hardware and operating system, cloud providers also setup and maintain the application software for the clients. CRM vendors typically use this model to provide

services to their clients, as many small organizations cannot afford the high price-tag of these ERP applications, but unavoidably need to utilize the products in their businesses.

Figure 1.2 shows the position of each layer. IaaS is the basic and lowest layer, and the higher layer abstracts the details from the lower layer. In Public Cloud domain, the segregation of each layer in term of service offering is more obvious; nevertheless in private or on-premise cloud, all 3 layers are assimilated to serve the end products' need.

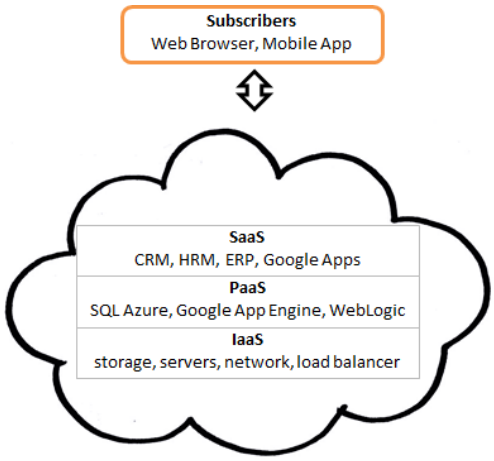


Figure 1.2: The service model in cloud computing. Each layer is preferred by different consumers, depending on the intensity of desired abstraction.

The focus in this thesis is at this juncture, where usage *optimization on hardware resource performance* in the cloud virtualized environments, which requires close examinations on fault analysis that subsequently provides indication to hardware failure prediction and performance degradation are studied. Together with this *resource optimization*, methods to achieve more effective resource utilization *monitoring* and *affirmation* are scrutinized. The increase in complexity and dynamics of these systems renders any current heuristic and rule-based resource management approaches insufficient. Hence the significance of the researches in this thesis is to provide new insights on resource management from non-conventional perspectives, as well as

complementing existing tools and utilities which are commercialized to achieve the same objectives.

One of the most popular key enablers to Cloud computing is virtualization. This type of cloud environment comprises of multiple physical machines interconnecting together to form VM. At high level, the virtualization layer is positioned between the physical hardware and the operating system, and it provides management services for the popular scalability and flexibility attributes in Cloud, via a software utility called *hypervisor*. Figure 1.3 depicts at high level an implementation of Private Cloud virtualization, powered by the *VMWare* Cloud Virtualization Infrastructure (VMWare, 2006). IBM SmartCloud Application Services at the PaaS level (IBM, 2011) is another example that enables on-premise Cloud hosting. The database populated with TPC-H data as displayed in Figure 1.3 is not a typical implementation of virtual environment; rather it is a proposal from this thesis, which serves as the core element in enabling the proposed algorithms and mechanisms in subsequent sections.

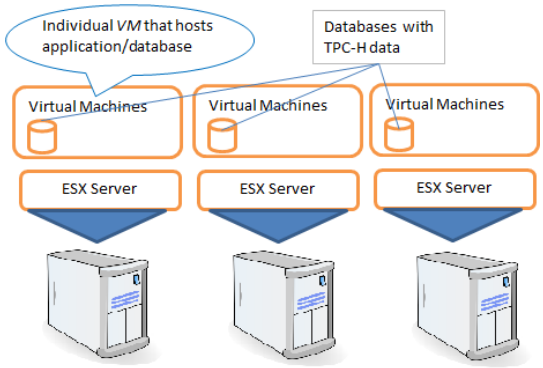


Figure 1.3: VMWare Virtualized Infrastructure. The virtual machines can scale to hundreds or thousands.

The scalability is achieved by provisioning hardware resources to the individual VM whenever needed. The ESX server enables flexible hardware provisioning and de-provisioning of resources for particular VM. Each VM is independent, able to host different desired operating systems, for a great variety of functions. For instance, a

virtualized environment by 1 ESX server can host database operations, HR applications, and all kinds of other front and back office applications. This diagram is illustrated as it characterizes a common on-Premise Private Cloud configuration. This architecture is typical for *Parallel Database* hosting; and the VM utilized for experiments in this thesis is constituted of these components.

This research focuses on on-premise solution, due to data integrity and security reasons. It is not the problem with the technology itself, as human always found ways to address shortcomings or challenges technically. However the current perception and skepticism of Enterprises on security and reliability will delay the Public Cloud adaptation. In this case, Private Cloud is the easier solution. As described by Harms and Yamartino (Harms & Yamartino, 2010), the *Horseless Carriage Syndrome* when automobiles were introduced in early 20th century perhaps will slow down the embracement of SQL Azure (Microsoft, 2011); however the economics of the Cloud could overwhelm the constraining factors in time to come. Amazon is taking a step forward by introducing AWS GovCloud (US), which is hosted in Amazon Web Services (Amazon, 2012). Its compliance with US International Traffic in Arms Regulations (ITAR) and Federal Information Processing Standard (FIPS) Publication 140-2 are hoped to prove to the world its robustness in data hosting by Public Cloud. Google claims its strength in data security via ten components of Google's multi-layered security strategies incorporated in Google Apps (Google, 2010). Oracle through its Exalogic Elastic Cloud product provides similar offering for Public and Private Cloud, plus Hybrid Cloud that is capable of Cloud bursting (Oracle, 2011). Even though Public Cloud computing has been widely accepted and deployed for web-based application, its use for mission critical database operations is still at early stage of adoption. While it is anticipated that Public Cloud will mature and flourish, this thesis is written to detail on the resource

administration mainly targets the databases hosted in on-premise Private Cloud as it is expected that database hosting on Private Cloud is going to thrive for quite a while.

Nevertheless, the proposed mechanisms are applicable for resource planning in public or Hybrid Cloud from resource management perspective. The research is in alignment with perception from Harms and Yamartino (Harms & Yamartino, 2010) that the full advantage of Cloud Computing can only be properly unlocked through proper intelligent resource management. Due to problem with current RDBMS licensing model and the unpredictable nature of SQL queries, over-dynamic resource allocation paradigm will take a while before it is widely adopted for database hosting in Cloud. In the studies for this research, it is discovered that for most enterprises, unless the allocated resource is provisioned specifically for a short timeframe of surged transactions, the actual resource requirement has the tendency to accumulate and stay in the VM, via static on-demand request. Furthermore, at this moment, it is not easy to map QoS requirements to low-level resource states such as CPU and memory requirements (Q. Zhang et al., 2010). This is especially true in database hosting that has many variables in its operations that does not adapt well with dynamic resource provisioning model at this time. The motivation of the resource management proposals is developed with such scenarios in perspective.

It is noteworthy that in Cloud computing, the popularly deployed database technology is segregated into 2 segments. The first type is the recently popularized paradigm: the *Hadoop MapReduce* framework for 'Big Data', made popular by Google in 2003. Initially the company kept the proprietary right to this technology, but later the Apache Software Foundation via its global community of contributors built the Apache Hadoop framework together with its open-source ecosystem, based on published Google's MapReduce and Google File System (GFS) papers. The second type is the *Parallel Database*, a technology widely deployed to various businesses serving numerous

industrial functions. This technology has stood the test of time and many today's mission critical operations depend on it for survival. The focus in this thesis is on improving this *Parallel Database* hosting on the virtualized cloud architecture.

1.3 Motivation

This research focuses on database operations. This is the layer that is usually the most resource intensive of all the layers in the application service offering, hence is chosen for more effective depiction of the resource management proposals. As the target of the researches is scrutinizing on the database layer where data integrity, security and privacy criterions are sometimes of utmost concern, the proposals strive to refrain from actual data access to preserve these goals. The mechanisms manipulate metadata of real workloads, and utilize synthetic queries to synthesize artificial workloads to provide data input to the proposed mechanisms.

The resource utilization *monitoring* on the hardware adequacy focuses on meeting the computing resource requirements for the database transactions. The output from the proposed monitoring method is subsequently channeled for *resource planning* and *scaling* purposes. The resource utilization *monitoring* couples the parameters obtained from operating system and database, and combines both perspectives to produce the dashboarding output for *resource planning* and *scaling* decisions. The result is deemed more insightful as end users' experience from the database transactions is matched to the aspect of hardware resource consumption.

Resource utilization *monitoring* in above cannot function as an isolated widget. It needs to go along with performance optimization and fault analysis in the VM so that resource or job scheduling efforts are not overshadowed by lackadaisical hardware. Hence the resource utilization *optimization* topic is studied subsequently after *resource monitoring* topic is examined. It is observed that workload delineation is an NP-hard problem.

Hence in order to arrive at the required precision, often time heuristic approach is the best method to be employed. With this, the initial set of data for baseline is obtained, which characterizes the behavior of transaction processing in the database. Subsequently the output from the same test configuration is compared to the baseline to discover if there is any change in the VM's computing behavior. In this context, this research leverages the TPC-H benchmark data and queries, and proposes the optimization mechanism that is taking the hardware resources as variable instead of rigid constant. The proposed model can also examine software aging and provide indication if rejuvenation is required. This work is significant because in virtualized environments, VM that comprises of a hundred machines is not uncommon. The increase in the number of nodes, coupled with the fact that most systems are made up of inexpensive commodity computers, the chance for hardware failure is high. For instant, in the paper that described the architecture of Google File System (GFS), it was mentioned that GPS is designed in the cloud with anticipation that failure will happen regularly (Ghemawat, Gobioff, & Leung, 2003).

Apart from the significant reduction in operating cost for enterprises by having optimal resource utilization *optimization* mechanisms, the living environment also benefits from the cutback in electricity demand, hence reduces the carbon footprint. Green computing is the paradigm that is gaining significance, primarily due to the ever-increasing business computing requirements, the acceleration of energy cost and growing awareness of global warming issues.

In typical traditional production environments, many of the servers are running below optimal capacity in terms of memory, CPU and disk space resources. Virtualization is able to pool the need of resources from these servers together, serving them as a unit that lessens the complexity of IT management, for instance operating process

standardization and patching administration. Hence combined virtualization and resource utilization *optimization* provide for a better sustainable environment.

With the resource utilization *monitoring* and *optimization* mechanisms put in place, the hypothetical affirmation on resource adequacy and hardware health is established. However, this solves only half of the puzzle. At this juncture, the effectiveness of the proposed mechanisms needs to be practically proven. Furthermore, critical SLA-bound transactions must be regularly verified to ensure that acceptable response time is always preserved in the allocated set of hardware. The most accurate method to measure the anticipated performance is to attest the VM capability with real end users' experience. Hence, a method is envisaged to synthesize high load conditions in the VM, and subsequently allows transactions to be executed in such stressed scenarios. In such cases, if these transactions can deliver the desired response time in the stressed condition, the VM can be convincingly released for production use. Nevertheless, in order to stress the VM appropriately, the synthetic workload employed to stress the system will need to mimic the actual desired workload, so that comparative system state can be obtained for the test. The challenge here is then to discover the similarity between the synthetic and real workloads, so that real transactions can be stress-tested in the VM that matches the real stressed situation. Of course in this sense, the best method is to conduct load testing using convention load testing software, for example *HP LoadRunner*(HP, 2007) software that iteratively executes the real workload to load the system to the desired threshold for testing on real transactions. However, this method commands a lot of coordination efforts. The works here involve the building of test cases, allocation of technical personnel to standby for the lengthy test time. In view of these, the proposed methodology in this thesis provides an alternative to reduce the time and effort needed to create the stress-testing environment, which delivers the same objective of transaction verification at particular host conditions. These are detailed in

the resource utilization *affirmation* section. In the case where hardware change occurs in the VM, this proposed resource utilization *affirmation* model is able to determine the new constraining threshold point in the VM, for *resource planning* purpose.

Scholars have defined *resource management* slightly differently, based on the different approaches taken. C. J. Huang et al. (2013) employed the *support vector regressions* (SVRs) method to predict the adequacy of the computing resources by baselining the SLA response times, in order to maintain the desired performance in cloud environments. Younge, Laszewski, Wang, Lopez, and Carithers (2010) scrutinized on power-aware scheduling techniques, variable resource management, live migration, and a minimal virtual machine design to produce a resource management framework which attempt to increase the efficiency of cloud deployment. Yuan and Liu (2011) proposed a strategy to pre-reserve the resources in anticipation of near future computing requirements. A. Beloglazov and Buyya (2010) proposed live migration strategy to continuously move and consolidate the VMs, based on the CPU requirement, which takes reference from the required QoS. Such resource management strategy was conducted in virtualized cloud data center, where a substantial amount of hardware resources are available. Roy and Mukherjee (2011) introduced resource brokering to manage the resource requirement of tasks sent to the grid computing environment. Such agent-based resource management style is potentially applicable in cloud computing environment, where it could possibly applicable for *VM placement* strategy. These researches have 1 common objective: To increase the efficiency of the computing resource utilization. This research is taking the motivation from the exact same objective in producing the resource management mechanisms.

1.4 Problem statements

1.4.1 SLA compliance and monitoring systems

In considering the 'optimized' mechanisms, SLAs for the applications running the workloads are taken as the bottom line to ensure full compliance with business objectives (Blagodurov et al., 2013; Garg, Gopalaiyengar, & Buyya, 2011; Sakr & Liu, 2012). Ideally these workloads benefit from abundantly provisioned hardware resources in the VM. Nevertheless, businesses cannot allow for such configuration, as capital expenditure on computing hardware significantly affects profitability. To operate the database in oversized hardware to cushion the fear of impact from lack of required processing power will not be cost effective as it incurs unnecessary wastage, and to go down too little will be too less for the needed transactions. During the onset capacity planning stage, the actual requirement of computational power and storage, whether it's designed during the startup or meant for subsequent growth of the database operations, will be accurate only for the known initial application processing requirements. Hence a solution needs to arrive to provide an accurate insight on the hardware resource requirement during the production cycle of the application service offering. This stage is also known as the runtime phase where the transactions' characteristic will evolve from the initial expectation during the application construction stage. Hence subsequent scalability and performance must continue to meet the demand of the businesses. In this context, the monitoring utilities must be up to the task to ensure clear visibility on the resource usage. Conventionally, many monitoring utilities (Nimbus, 2013; OEM, 2013; Sitescope, 2013) focus on silo monitoring and scanning of the database and operating system parameters. Such tools are deemed inadequate in the sense that they cannot provide a holistic view of the combined database and resource state in the VM. The *monitoring* scheme in this research strives to produce a much clearer illustration on the

resource state in the VM, so that the resource scaling point can be determined more precisely.

1.4.2 Dynamic scalability issue for *Parallel Database*

It is to note that current aggressive provisioning and de-provisioning of hardware in the cloud is not suitable for the VM that is deployed for database hosting. This is due to the problem with the majority of current RDBMS licensing model and the unpredictable nature of SQL queries, together with the architecture of *Parallel Database* which captures the allocated hardware parameters during its initialization phase. In order for such databases to capture the new hardware configuration, a restart of the instance is needed (Ward, 2011), which is not practical for most applications that do not have the luxury of frequent database bounce. Hence the over-dynamic resource allocation paradigm will take a while before it is widely adopted for database hosting in virtualized Cloud. In this case, resource management mechanisms which allow a semi-dynamic approach to resource allocation are warranted. There are exceptions to such scenario, for example Microsoft SQL Azure is capable of dynamic scaling (Azure, 2010). However as this RDBMS is running in Public Cloud, there is another issue with skepticism on data security.

1.4.3 Continuous fault analysis

To ensure efficient planning of resources, the underlying hardware needs to have strict adherence to consistent high performance criteria, with no degradation in performance over time. To fulfill this requirement, hardware vendors can perform server health checks which typically involve lengthy downtime, and these can only be carried out during selective prolonged maintenance window. During normal operations, potential performance degradation due to partial hardware failure (Salfner, Lenk, & Malek, 2010) is usually gone unnoticed, and these hidden indicators might further evolve into total

failure. Apart from faulty hardware, suboptimal software performance also contributes to wastage in resource utilization. Progressive degradation (Hanmer, 2010) of the running software; or software aging can eventually result in hung up or crashes, and along the way deteriorate to a level that breaches the SLA. Hence there must be a proactive mechanism to detect these symptoms during normal operations, in other words fault analysis is required. Such fault analysis should be easy to be conducted periodically. Such ease of maintenance is intended in the proposed mechanism for the resource utilization *optimization* theme. Leaving the VM to operate in degraded condition is dangerous as the SLA could be breached at any time as the host or the database can go down abruptly without warning. Moreover, promised transactions' response time cannot be guaranteed as the host does not behave as expected.

1.4.4 Shortcoming of benchmarks

Industrial players and scholars have proposed many benchmarks to represent different workload types (Cole et al., 2011; Pavlo, Curino, & Zdonik, 2012; Pavlo et al., 2009), to cater for various metrics involved in the systems. However these benchmarked data are rigid in the sense that most of them cannot flexibly adapt to change in application and hardware technology. Because of this reason, a lot of these benchmarks cannot produce the same results even in proportionate response time when they are rerun in different hardware platform. Moreover some of the benchmarks are not generic for all platforms. Hence, the load testing instrument is still required in reality. However, the conventional load testing utilities are relatively time consuming and cumbersome, hence a light-weighted version of the load testing could potentially alleviates these concerns.

1.4.5 Data security issue

From the security perspective, by conventional definition, database operations from this aspect is generally scrutinizing on layered authentication, masking of data by

segregating end users' roles and functionalities, for instance by implementing Oracle Fine Grained Access Control (FGAC) (FGAC, 2003) to control visibility to data, enhancing network transport layer by incorporating encryption algorithms on the network packets, and strengthening of data security at the storage layer. In this research, the unbreakable database architecture is visualized, particularly to support industries that have stringent requirement in protecting against leakages of sensitive data. the masking of data from IT administrators can already be accomplished by industrial tools, such as Oracle Vault (Tbeileh, 2009). As the real users' data is hidden in such case, IT personnel need alternative input of data in order to perform administrative works particularly on performance tuning related matters. The proposals in this research are deriving the input for the resource management algorithms from metadata and synthetic data, however with no compromise in the delivery of the expected administrative tasks.

1.4.6 Insufficient measurement methods

In today's IT infrastructure, hardware resource *monitoring* is often conducted discretely. Even in the frequently deployed database monitoring system such as Oracle Enterprise Manager (OEM) (Huber, 2013), the operating system parameters are loosely coupled with the database variables. Another inadequate monitoring condition is that the alerts on CPU, memory, I/O and network utilization thresholds are notified individually instead of collective aggregation and analysis on these parameters. This, to an extend will hold the promise of service offering if many of the variables involved in supporting the applications are static in nature. However, such monitoring mechanism is not suitable in cloud environment where constant changing in the hardware configuration is common. For example, if CPU threshold is determined at CPU run queue of 4 in an initial set of hardware configuration, this value does not hold true if the number of virtual processors is increased, or the CPU power is enhanced. In these cases, a reevaluation on the new threshold value, by taking into account other hardware

parameters is required to continue serving the established transactions' response time. The same is true even if there is no change in hardware configuration, where the same run queue threshold value does not hold true after the application has operated in the VM for some time. This condition is contributed by potential partial hardware failure. Even if the run queue value in this case signifies correct CPU resource threshold, it still cannot guarantees that the transactions will deliver the promised response time, due to the change in execution path of the SQL involved. This scenario is primarily caused by the increase in data volume in the tables queried or modified by the SQL. In other words, the I/O reads and writes are different with the evolvement of the data. Even monitoring on database logs is not capable of discovering such degradation in SQL performance.

Oftentimes the logs from operating system do not clearly signify hardware performance degradation, until the break point which will then incur unwanted outage that results in the breach of SLA. This is especially undesired in VM that serve mission critical applications. To secure the host from such disaster, enterprises often spends large amount of money in procuring expensive hardware and software to enhance the high availability (HA) feature of the services. This directly erodes the profitability of the businesses. The more cost-effective and efficient method is to have a more proactive monitoring system in place. The proposal in the resource utilization *monitoring* theme strives to provide such mechanism where the aggregation of database and OS parameters is put into perspective.

1.5 Current practices

In the topic of resource utilization *monitoring*, enterprises usually provision a larger than needed hardware configuration in the VM to offset the risk of running into resource constraint in servicing the needed transactions. This results in significant wastage in

capital expenditure which considerably erodes the profit margin of the companies. For example, the list price for Oracle Exadata Database Machine Quarter Rack is USD 330K (Oracle, 2012). To upgrade from Quarter Rack to Half Rack will required the same amount of investment. Hence exaggerated hardware planning is detrimental to the bottom line of the businesses. To ensure wastage is kept to minimum, resource utilization *monitoring* is put in place. From the infrastructure perspective, current practices in the wide industry generally rely on monitoring and analysis at operating system level, where dedicated monitoring on CPU, memory and I/O utilization is common. For example the NIMBUS monitoring system(Nimbus, 2013) is often used by enterprises to monitor the host performance.

Server and database health monitoring are normally conducted by different entities in the IT department, where aggregation of inputs on the monitoring parameters from operating system and database is rare. From application view point, the monitoring often is focused on transactions' response time. One of the popular tool for this monitoring purpose is *HP SiteScope* software(HP, 2012). At each instance when the transactions' response time breaches the SLA threshold, they are flagged for all caretakers to investigate the problem. However it cannot specifically point out the root cause of the slowness, hence delays resolution to the problem. In the proposed model for resource utilization *monitoring* in this thesis, the variables involved in monitoring and analysis on workload processing are reduced comparatively to the conventional tools. The novelty of the proposed monitoring method is in the aggregation of the parameters from the operating system and database, to provide a faster and clearer visibility to the resource state in the VM, which then hasten the problem resolution.

To ensure resource utilization *optimization*, hardware vendors periodically perform health-check on the underlying hardware of the VM. Stress-tests are conducted and hardware failure symptoms are identified from the generated logs. These activities often

require long outages. Moreover, the health of the host is perceived only from the operating system perspective. On the other hand, to ensure the health-checked VM is capable of servicing required application transactions, application load testing using conventional load testing software is conducted. With these comprehensive tests, the VM's ability to support the SLA-bound transactions is guaranteed. However the main concern for these 2 methods is the lengthy outage window.

From resource utilization *affirmation* perspective, the conventional application load testing is capable of achieving the objective of providing the stressed host environment with the verification on critical transactions, practically. Nevertheless, these activities induce the same problem of long outage window, together with large amount of effort in coding test cases and coordination. Hence these activities are not sufficient and suitable to accommodate the scenario of elastic resource allocation in database operations, in virtualized cloud environment.

1.6 Research questions

In formulizing the research objectives, the questions to be answered are established as follows. The heads-up regarding the relevant research is provided.

5) What are the appropriate methods to provide barometric indicators to determine the host performance.

    - This question is answered in every researched theme, where the *monitoring*, *optimization* and *affirmation* mechanisms are utilizing the identified indicators for input.

6) How users' experience can be matched to these indicators discovered in (1).

    - The answer to this is illustrated by the *optimization* and *affirmation* themes.

7) What are the significant and appropriate parameters to be used to measure host performance.

    - These parameters are adequeatly revealed particularly in the *optimization* theme.

8) How these parameters interact with each other, in order to provide a more solidified output to measure the host and database performance.

    - The interaction of the parameters is established in the experiments conducted in the *optimization* theme.

9) How the proposed mechanisms can deliver the intended objects, in term of accuracy and consistency.

    - The accuracy and consistency criterions are achieved by combining parameters from the OS and database. The mechanisms to prove are shown in the experiments conducted for the *monitoring* and *optimization* schemes.

10) How the hardware in the VM performs before and after resource constraining threshold.

    - Such condition is explained in section 2.4.3.1, based on works from previous researchers.

11) Many of the RDBMS products in the industry have not developed the capability to be dynamically scaled, and the migration from 1 RDBMS platform to another is quite unlikely in commercial arena, what type of resource management mechanisms are appropriate for such semi-dynamic scalability requirement by these database systems.

    - The utilization of historical metadata in the methods proposed for the monitoring and optimization schemes appropriately address such scenario,

where the mechanisms have the time and space to revisit the previous data for analysis.

12) As fault analysis is a continuous effort, how capable is the proposed mechanisms in accomplishing this goal.

    - The mechanism proposed in the *optimization* scheme is meant to be continuously and periodically executed during steady-state condition, to detect any anomaly.

13) How to address the shortcoming of the current available benchmark, where one-size-fit-all scenario is almost non-existent.

    - In the proposal for the *affirmation* scheme, the creation of benchmark is proposed, which is tailored for particular environments.

14) How security aspect is addressed in details, by the proposed mechanisms.

    - All the themes are not utilizing real data in the proposals, hence avoiding access to real data. As such, the security of the data is ensured.

1.7 Research objectives

1) Resource utilization *monitoring*: To produce a mechanism to monitor the resource consumption in real workload processing, via the depiction from the associated metadata.

2) Resource utilization *optimization*: To compute an algorithm that is capable of extracting the underlying hardware performance characteristic.

3) Resource utilization *affirmation*: To develop a method that defies conventional load testing mechanism, that can be utilized to load-test desired SLA-bound transactions which need to adhere to strict response time requirement.

4) To evaluate the effectiveness and shortcomings of the above proposed mechanisms.

1.8 Scope of research

The scope of the research focuses on hardware resource management, with focus on resource utilization *monitoring, optimization* and *affirmation* areas, for application transactions carried out in *Parallel Database* architecture hosted in virtualized cloud environment, in a secure manner from the data security perspective. On-premise Private Cloud architecture is envisaged during analysis, formulation and creation of the proposed algorithms. Nevertheless, the planning and analysis of these algorithms are delineated such that they are extensible to Public Cloud environment if desired.

The research in the *monitoring* segment is targeted at discovering the behavior of the operating system and database parameters when they are aggregated. This is often neglected in normal practice of today's IT management. The aim is to accurately characterize the hardware, together with associated software components, for the purpose of achieving the most optimal condition for application service offering.

Statistical modeling method on TPC-H data and queries is employed to determine the resource consistency and performance in the VM, together with *machine learning* and *linear regression* analysis as the foundation to the resource utilization *optimization* research. The output is subsequently compared with the baseline data using the same data sets to compute potential failure symptom in the systems.

In the resource utilization *affirmation* study, queries in the TPC-H benchmark are chosen and subsequently aggregated to synthesize a workload scenario that stresses the VM, so that resource threshold together with transactions' response time can be verified.

The introduced mechanisms in this thesis are generic for all hardware platforms, in the sense that TPC-H benchmark can be adapted by various RDBMS serving multiple types

of operating platforms. Most importantly, as real data is not engaged throughout the study, secure resource management goal is achieved.

1.9 Chapter organization

This thesis consists of 6 chapters. Chapter 1 contains introduction to the research topics. It provides information on problem statements and motivations of the research. The high level explanation of the research topics: resource utilization *monitoring, optimization* and *affirmation* is provided. Subsequently the current practices are examined. Consequently the research questions and objectives are reiterated and summarized. The boundary of the researches is then presented in 'Scope of Research' section.

Chapter 2 details the literature reviews on related works on cloud hosting development particularly in resource management and optimization. Cloud security is also examined, as this is important for database operations hosted in cloud environments. Subsequently the feasibility of the resource management subjects scrutinized in the studies are analyzed, by matching these to related researches.

Chapter 3 describes the research methodologies in details. In this case, all the 3 schemes: resource *monitoring*, *optimization* and *affirmation* are elaborated, with focus primarily on input coming from real workload's metadata and synthetic workload characterized by TPC-H benchmark data and queries.

Chapter 4 details out the *design* of all the 3 themes. The construction details of all the 3 models are elaborated in length, with explanation on the logic and feasibility of the choices in the design.

Chapter 5 features the analysis and evaluation on the conducted experiments. The discussions touch base on the currently available tools and utilities, and compare them to the proposed mechanisms.

Chapter 6 concludes the research and discusses on how the research objectives are fulfilled. The contribution of the research outcome is then discussed, followed by suggestions on potential future works.

2. LITERATURE REVIEW

2.1 Introduction

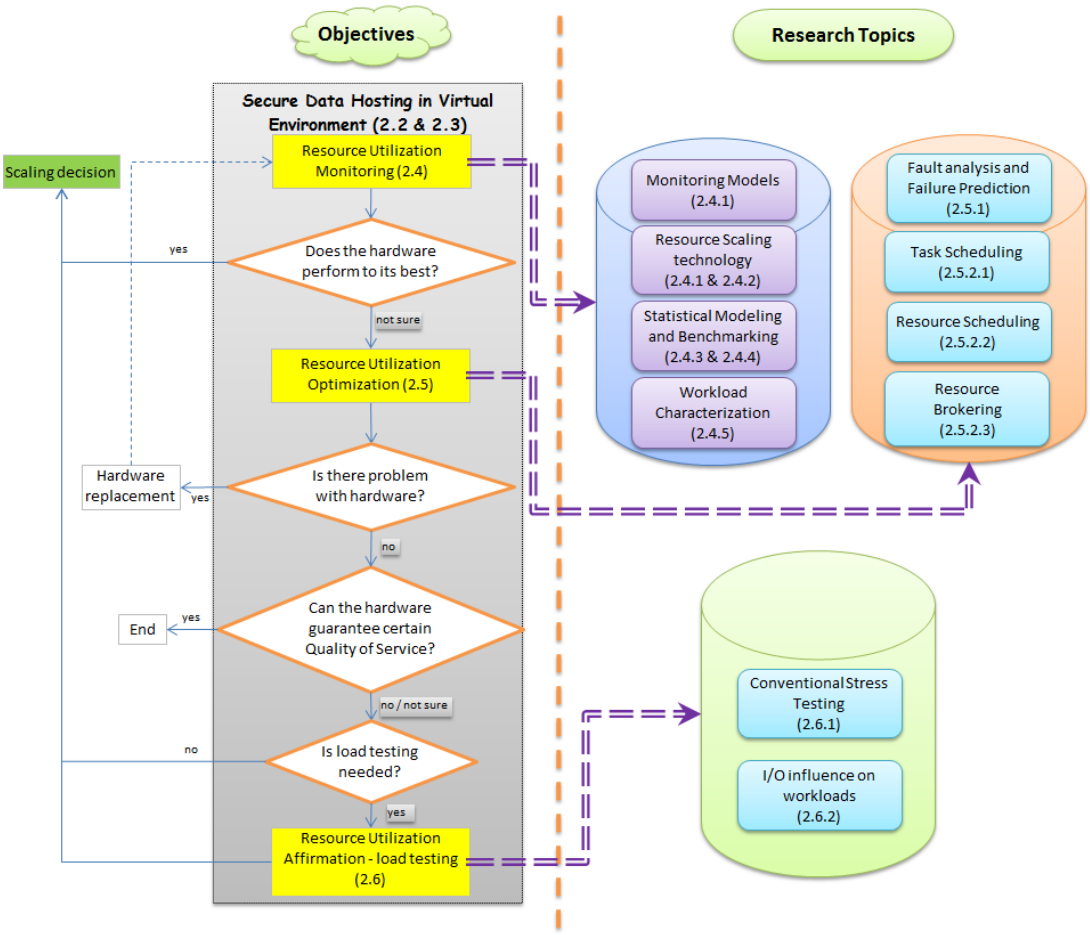


Figure 2.1: A summary of the research. The flow chart at the left side illustrates the contributions to resource management for cloud environment. The yellow boxes depict the main topics centered in this thesis. The right half details into the sub-topics which constitute the essences for the buildups to the proposals in the thesis. Each researched component is labeled with related chapters and sub-chapters in parentheses.

The core of the research is dealing with resource management in virtualized cloud environments, serving *Parallel Database* architecture. In such system, the database operations are executed in parallel, to take advantage of the multi-processors framework. The majority of RDBMS running today are classified in this category. This technology has matured since the past 2 decades, and it is powering many of today's mission critical applications. The wide deployment of *Parallel Database* has generated vast interests in the research and development area, and created enormous employment opportunity. In this cloud computing era, the adoption pace of this database technology

is relatively slow as compared to web based applications. Nevertheless, there are huge opportunities to be harnessed by *Parallel Database* technology, in view of the many benefits offered by cloud paradigm, particularly in the area that takes advantage of the elasticity and scalability features of the computing resources. Hence, effective resource management will be the dominant component in the evolution of this technology. This thesis strives to improve in this area, and takes the database hosting architecture to a greater height.

In proposing the resource management mechanisms, the security of the data is accounted for. The proposed algorithms intend to provide as much protection to the underlying data as possible, at the same time increase the efficiency of resource management in the virtual hosts. As depicted in figure 2.1, the 3 main themes of this research are the resource utilization *monitoring*, *optimization* and *affirmation*. The relationship between these 3 themes is illustrated in the flow diagram in the figure. It starts with resource utilization *monitoring*, where the state of resource usage is monitored and analyzed via new proposed non-conventional mechanism. With these data obtained from the monitoring instrument, further question is asked, if the data fittingly represents the expected resource performance in the system. This is because if there is hardware issue in the infrastructure, the resulted resource state or threshold cannot appropriately depict the condition in the VM. Hence the next research theme focuses on resource utilization *optimization*. The scope here is to discover abnormality in the hardware, so that the computing capability is maximized and optimized. A holistic view on the hardware state can be provided by the proposed mechanism. At this point, the resource usage situation and hardware condition are understood. With the information, there is still no guarantee that the agreed upon QoS can be delivered by the hosting platform. For instance, if a transaction is required to complete within 2 seconds, the business will need reassurance that the computing resources can constantly produce

such response time. Hence, the next theme is regarding the resource utilization *affirmation*, where load testing is often deployed to verify the workload response time and attest the capability of the VM. The proposal here strives to shorten and simplified the load testing mechanism. The eventual objective is to arrive at a juncture, where resource scaling decision can be made accurately and convincingly.

2.1.1 Resource utilization *monitoring*

To arrive at these resource management themes outlined above, a wide range of literatures have been reviewed. In the *monitoring* segment, the topics surveyed are:

1) Monitoring models. The monitoring mechanisms deployed commercially, as well as envisaged models from scholars are studied. The employed methods to produce such models are also scrutinized. Only models that are aspired by the direction of technical advancement in cloud computing are examined.

2) Statistical modeling and benchmarking. The underlying technology in producing the monitoring models as in point #1 is examined here. These techniques strive to serve the purpose of providing greater visibility on the resource utilization condition in the VM. Even though the research focus here is on *Parallel Database*, the models deployed in *MapReduce* framework are investigated too, as they are more aggressively studied recently, and have great potential in incorporating into this research area.

3) On-demand resource scaling technology. Another main contribution of scrutinizing resource state in VM is to ensure the resource scaling algorithms are accurate in matching the computing needs in the VM. Hence a lot of literatures have studied the monitoring mechanisms prior to their subsequent proposals on the on-demand scaling designs. Almost all the surveyed on-demand resource scaling methods are applicable for the application layer, particularly for web based applications. Such

aggressive resource allocation and de-allocation techniques are seldom scrutinized in database domain. Nevertheless, this topic is included in this survey, as it has great potential to be applied for the database layer. The recent launch of Oracle 12c (Avril & Hardie, 2013) has provided such insight of the future direction of the *Parallel Database* technology, where the elasticity feature is enhanced in this new product, in both the licensing and technology segments.

4) Workload characterization. Another purpose of resource monitoring is to ensure the invested hardware is optimally utilized, in the sense that there will be minimal burstiness in the real workloads. Burstiness denotes occasional spikes in resource usage due to sudden surge in the load demand, or unoptimized scheduling of jobs. One way to address such concern is to partition the workload into manageable distributions. In order to do this, the workload characterization effort is essential. To arrive at this understanding of the workload, the monitoring on its resource utilization is the primary input to be considered. Hence resource monitoring is also studied in a lot of literatures that cover the characterization of workloads.

2.1.2 Resource utilization *optimization*

The scope for the next theme, which is on resource utilization *optimization*, involves following subjects:

1) Fault analysis and Failure Prediction. To guarantee that the provisioned resources are utilized fully, the primary target is to ensure that the backend hardware performs to its full capability, in consistent manner. To safeguard this interest, the hardware needs to be free from partial failure or potential complete breakdown in future. This topic is not widely studied by scholars particularly who involve in cloud environmental studies, as it is deemed an old topic in the computing world, and it is perceived that the hardware can be replaced quickly in the cloud. Hence the impact

of failed hardware in the cloud architecture is not as high. Nevertheless, when it is coupled with the interest in resource utilization *optimization* topic, it becomes the primary concern before any other software optimization mechanisms.

2) Task Scheduling. Many scholars studied this as a way to distribute the tasks in the workload to different VM or different cloud providers, in order to take advantage of resources available elsewhere. By studying their proposals, this thesis turns some of the suggestions into workload partitioning effort that segregates the workloads into chunks of smaller tasks. These tasks are scheduled to different time blocks available in the VM, instead of to different hosts. With this, the resource utilization efficiency in particular VM can be elevated.

3) Resource scheduling. This type of scheduling deals with resource addition or subtraction from particular VM. It is relevant to this *optimization* theme, as over-aggressive scheduling algorithm will cause significant overhead in the system, whereas algorithms that do not respond fast enough to the scheduling requirement will potentially breach the SLA. Vertical and horizontal resource scaling modes are studied in this area.

4) Resource brokering. This topic is related to this *optimization* initiative. The studied brokering mechanism is the Auction-based type. The providers and consumers both achieve the objective of selling and buying at the price agreed upon by both parties. The meeting price point from both sides is the target of the brokering mechanism. By arriving at the prices that satisfy both parties, more optimized resource provisioning and de-provisioning conditions are met.

The scope of the research is on database operations hosted on virtualized cloud domain. As the applications are running at the foreground of this database layer, the term ‘applications’ is used interchangeably to also depict the database operations, unless specified otherwise.

2.1.3 Resource utilization *affirmation*

The third theme deals with load testing in the VM. This resource utilization *affirmation* is not solely applicable for cloud environment, but also in other platforms. In this section, the conventional load testing methods are scrutinized, and the essence of them is extracted to construct shorten and simplified version of the load testing mechanism. In this sense, the I/O charateristic is examined, and its influence in characterizing the desired workload is applied to build the light-weighted load testing mechanism. The proposed construction of the shorten version of load testing mechanism is not to replace the conventional load testing, rather it provides complement to the conventional tools to alleviate the time constraint factor.

2.2 Virtualized cloud infrastructure

In this section, the scholarly studies on cloud infrastructure, as well as commercially deployed systems are examined. It is to note that these virtualized environments are not solely designed for *Parallel Database* architecture, but they are also applicable to host *MapReduce Framework* as well as the front application layer. A comparison between current cloud computing and older hosting technology is also carried out. In general, cloud infrastructure can be categorized into 2 realms. Firstly, the cloud computing can be configured by creating OS image directly on the hardware. Such method is termed as *bare metal provisioning*. Such method is commonly deployed for the *MapReduce Framework.* The second type involves virtualization on the underlying hardware, where OS images are created on the *virtual machines.* Such practice is more widely deployed for web and application server hosting, as well as database operations running on *Parallel Database* architecture. The thesis bases its proposed resource management approaches on this latter cloud infrastructure.

Zhang et al. (2010) defined Cloud Computing as: *"a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction"*. It is this ease of effort in application hosting that makes Cloud a popular and fascinating choice of hosting. Cloud Computing enables the agility required to accelerate the time-to-market of new products and services while reducing the cost to design, build, deploy and support these products and services (Glas & Andres, 2011). Before advancing to other topics, it is noteworthy to mention the 2 database models currently serving the vast industrial community. In this cloud computing world, the database hosting architecture can be segregated into 2 main categories: The first one is the *MapReduce framework*, which is popularly powered by *Apache Hadoop* solution. The second category is dealing with *Parallel Database* hosting. The *Apache Hadoop* framework predominantly targets the *Big Data* which currently is accommodating unstructured data from the web-based applications. Due to the computing nature of the Big Data processing and analysis, it has adopted cloud computing much earlier than *Parallel Database*. Google is the first to pioneer this, where it introduced the *MapReduce* algorithm to process and mine enormous data from the web, using nodes that scale to thousands (Dean & Ghemawat, January 2008). The *Parallel Database* essentially provides for the relational database model, where it has developed and evolved over the past 3 decades (Codd, 1970). Relatively, *Parallel Database* architecture can be termed conventional in comparison to the *Hadoop MapReduce* Architecture by scholars and the industrialists. Both technologies are different from each other, from the perspective of data storage and processing algorithms. Nevertheless, many scholars and industrial players are exploring the opportunity to aggregate and incorporate the advantages from one domain to each other. From the hosting aspect, both can be adequately accommodated by cloud virtualization, which

provides a wide range of benefits in terms of scalability, elasticity, flexibility, economics of scale, reduction in capital expenditure and more efficient deployment of workforce. Nonetheless, it will be a while before these exciting researches of technology integration make their way to the industry. Hence this research scrutinizes the possibilities of improving the current matured technologies, focusing on virtualized *Parallel Database* architecture hosting.

The predecessor to the virtualized database hosting is the standalone server architecture in the client-server model. This hosting model has stood the test of time for the past 2 decades before the advent of cloud computing paradigm. Many mission-critical applications were and some are still being deployed in such standalone server platform. However, primarily due to the rapid progress in software development, it is inevitable that the computing resources required by the database transactions need to be available instantly and affordably to ensure viability in application service offerings. In order to accomplish this requirement, hardware virtualization makes its way to the forefront of the hosting technology. In 2011, Gardner predicted that virtualization will be the first in the top 10 list of most significant future IT technology-related trends (Cooney, 2011), and apparently this still holds true today.

In the cloud computing arena for database hosting, Private Cloud is often deployed in today's enterprises. There are also database deployment in the Public Cloud, for example those that engage Microsoft SQL Azure (Microsoft, 2011) product. Oracle through its Exalogic Elastic Cloud product provides similar offering for Public and Private Cloud, plus Hybrid Cloud that is capable of Cloud bursting (Oracle, 2011). The enigma in the security and privacy aspects, particularly in the data storage and transportation matters is slowing down the embracement of public hosting of database operations. As described by Harms and Yamartino (2010), the *Horseless Carriage Syndrome* when automobiles were introduced in early 20th century resembles the

current perception in the embracement of SQL Azure. However the economics of the Public Cloud might overwhelm the constraining factors in time to come. Public Cloud vendors are building extremely large-scale, commodity-computer Data Centers in low cost locations, and they uncovered factors of 5 to 7 decrease in cost of electricity, network bandwidth, operations, software, and hardware available at these very large economies of scale (Armbrust et al., 2009). While the anticipation is that Public Cloud will mature and flourish eventually, the resource management topics in this thesis focus mainly on the immediate needs of databases hosted in on-premise Private Cloud, as the database hosting on Private Cloud is going to thrive for quite a while.

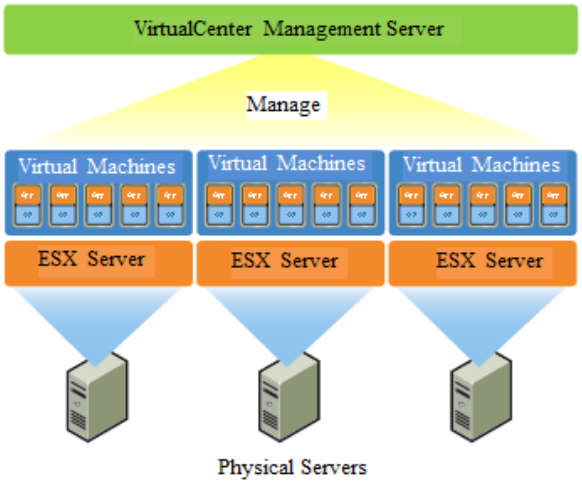


Figure 2.2: Virtualization Infrastructure diagram. Such architecture is typical and popular for *Parallel Database* hosting. Adapted from (VMWare, 2006)

Figure 2.2 denotes a typical implementation of virtualized environment for on-premise Private Cloud. The diagram depicts a VMware Infrastructure (VMWare, 2006) which is commonly deployed for *Parallel Database* architecture. The scalability is achieved by provisioning of resources from the underlying hardware, made possible using the *hypervisor* component. In VMware infrastructure, such component is called VMware vSphere (VMWare, 2013b). It is denoted by the ESX Server layer in figure 2.2, where it is also named VMware vSphere ESXi. It is to note that the hardware is mainly comprised of x86 commodity servers, which significantly reduces the capital expenditure in IT infrastructure spending. Apart from the relatively cheap hardware, the

clustering of these commodity servers which aggregated and interconnected by same network and storage subsystems managed by 1 ESX server does not require identical configuration in each of the servers, hence greatly reduces the cost and complication in procuring additional servers in future. The experiments conducted in this thesis are carried out using this hosting platform, where the research outcomes are produced from this hosting technology. The task scheduling, resource scheduling, cloud bursting, security and availability control discussed subsequently in this chapter are managed by the virtualization program at the front end. VMWare product in this layer is named VMware vCenter Server (VMWare, 2013a). The terminology employed to classify virtualization products in the industry often denotes the toolkits in this layer. In the same category, OpenNebula (OpenNebula, 2013) which is an open-source project claims to provide wider range of virtualization support, as illustrated in figure 2.3. From its documentation, it is said to be able to accommodate different type of hypervisors, for instance Xen (Xen, 2013), KVM (KVM, 2013) and VMWare vSphere. It is also to note that the vendors to these hypervisors also have their own management software; hence there are a wide range of choices for the industry to choose from. The other frequently deployed industrial on-premise cloud virtualized infrastructure solutions are Oracle VM3 which originates from Xen enabled virtualization (M. Kumar, Roberts, & Kawalek, 2011), IBM SmartCloud (S. Williams, 2011) and Microsoft Hyper-V (Microsoft, 2007).

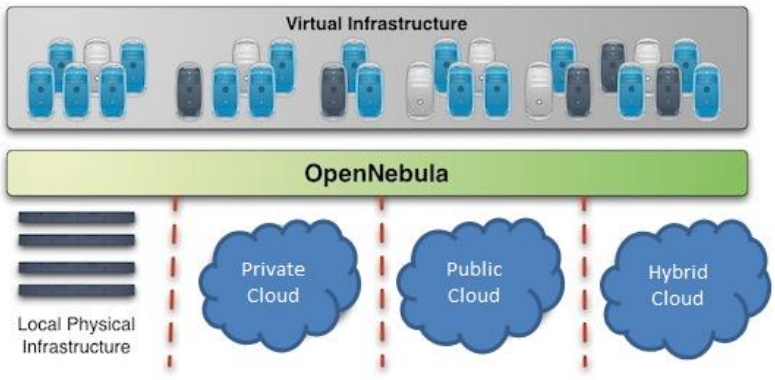


Figure 2.3: OpenNebula virtualization management software. The advantage of this software is that it is developed by open-source community, hence it has the potential to flourish in the same way that Linux did. Adapted from (C12G, 2010)

Another frequently deployed open source virtualization management toolkit is Eucalyptus (Eucalyptus, 2013c). The private virtualized platform managed by this product is often engaged by scholars as test beds to carry out experiments on cloud related researches. From the industry perspective, it is ideal for test and QA environments. As the development of this toolkit focuses on compliance with API utilized in Amazon Web Services (AWS), it is almost inevitable that the eventual production environment will be hosted by AWS. Figure 2.4 illustrates the position of this toolkit in the virtualized Private Cloud environment. Note that its architecture is not much different from figure 2.3, with the exception that it is only applicable for Private Cloud deployment only. Eucalyptus is an excellent platform for researches in view of its open source nature, feature-rich and freely downloadable.

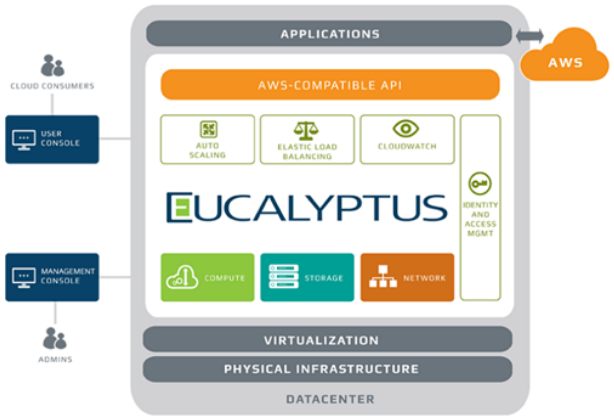


Figure 2.4: Eucalyptus platform. Note the tight association with AWS which makes deployment of applications in Public Cloud easier after the development phase. Adapted from (Eucalyptus, 2013b)

Early in this chapter, it is mentioned that the predecessor to the virtualized database hosting architecture is the standalone server model. A more granular view is also studied from another angle in this subject of cloud computing evolution. Zhang et al. (2010) defines Cloud Computing as an evolution of grid computing, as it comprises of thin clients, Grid Computing and Utility Computing. Buyya et al. (2009) differentiated between Cloud Computing and Grid Computing at the *virtualization* level, where Cloud is defined as next-generation data centers with nodes "virtualized" through hypervisor technologies, dynamically "provisioned" on demand as a personalized resource collection. The virtualization in cloud computing provides the ease and flexible capability on resource allocation. The authors also define grid computing as '*a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed `autonomous' resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements'*.

Foster et al. (2008) compared Cloud and Grid in length; and from dynamic resource provisioning perspective, Cloud is deemed more flexible than Grid, as Cloud is leveraging virtualization technologies more extensively. Hashemi et al. (2012) described how cloud computing is more superior as compared to grid computing, by

taking insights from various aspects. Table 2.1 provides a summary on these comparisons.

Table 2.1: Cloud vs. Grid computing. Generally cloud is perceived as more advantageous than grid in serving wide range of applications.

	Comply with criterions		**Comments**			
Criterions	**Cloud Computing**	**Grid Computing**	**(Foster et al., 2008)**	**(Hashemi & Bardsiri, 2012)**	**(S. Zhang et al., 2010)**	**(Buyya et al., 2009)**
Reduce the cost of computing	Yes	Yes	In Grid, Commodity clusters are expensive to operate in comparing to low-cost virtualization offered by Cloud	Grid: Promoting sharing of resource to other location	Cloud provides cheaper alternative as development can be done in public (more economical) or Private Cloud	Grid: Reduction in cost depends on VO management.
Massively and easily scalable	Yes	Partial	Grid: Resource span across multiple virtual organizations (VO), harder to control different groups. Cloud: Resource is managed by single vendor, easy to be provisioned	Grid is less scalable than cloud	Grid: Potential of over-provisioning of resources	Cloud: easier to scale using *Virtual Machine*
Deliver on demand resource	Yes	Partial	Grid: Need to wait for arrival of resource from VO. Cloud: Can provision resource within few seconds to few minutes.	Grid: resource not provided real-time, need to wait	Grid: Tasks need to wait if requested resource not available at particular point in time. Cloud: Guarantee resource for transaction processing	Cloud: authors propose a resource brokerage system to deliver resources to needed transactions
computing resource are packaged as metered services	Yes	Yes	Grid: Resource is provisioned from VO cloud: resource is provided by single vendor	Grid: resource might be limited, depending on the Grid participants Public Cloud: unlimited	Grid: Might need to wait for available resource from VO Cloud: instantly available	Grid: pioneer in 'computer utilities' concept, much like electical utility
Easy to control computing standard	Yes	Partial	Grid: as VO is control by multiple organizations, harder to implement uniform standard of computing environment. Cloud: 1 organization controls the cloud data center, easy to standardize.	Grid: More standards possible from different VO	-	Cloud: Cloud providers still working to standardize computing standard across different providers

Clear visibility of data locality	Partial	Yes	Public Cloud: Data are scattered in various locations Grid: locality of data easy to track Private Cloud: Data is located in own premise	Grid: suitable for data intensive operations Cloud: not suitable if for operations that need a lot of IO	-	Cloud: Some cloud providers, eg. Akamai (Su, Choffnes, Kuzmanovic, & Bustamante, 2006) and Mirror Image (Mirror-Image, 2013) provide locality service for consumers to host global application
Virtualization	Yes	Partial	Grid: Control of resources by individual VO discourages full virtualization	Grid: Not necessarily needed Cloud: vital element in the architecture	-	Cloud: Stronger support of virtualization
Audit trail on transactions	Partial	Yes	Grid: Mature workflow tracking on transactions Public Cloud: audit tracking is an open problem	-	-	Cloud: Authors propose MetaCDN that is capable of providing logging audit tracking
Data security and privacy			Grid: take advantage from many years of evolutions in security related matters Cloud: Relatively simpler	Grid: lower security model than cloud due to different VO management	-	Cloud provides flexibility in allowing consumers to alter their security requirements
Generality of usage	Yes	Partial	HPC applications run better in grid compared to cloud due to shorter interconnect in network and processors.	For computing intensive operations, that does not adhere to tight SLA response time	Grid: Provide for specific domains, eg. biology grid, geography grid Cloud: Provide for more generic application use	Grid: For collaborative scientific and high throughput computing applications Cloud: able to support generic applications
Robustness						Cloud is perceived as more robust

In the following section, the challenges of migrating from either standalone server or other platforms are briefly discussed. It is not to be elaborated in length as it is not the core topic of this thesis, but as it is related to the choice of hosting platform chosen for the experiments in the thesis, it is worth the mentioning here. Babar et al. (2011) studied the migration of a customized system from standalone server to cloud. The application and database coexist in the same server before the migration. From this paper, 3 critical points are observed. Firstly, the system to be migrated should be able to take advantage of the scalability feature of cloud computing. Secondly, the system should be able to work in both Private and Public Cloud. Thirdly, if possible, the migration should be transparent to the end users, particularly if the system supports vast community where alteration on the login interfaces is tedious. For the first point, in order for the system to be scaled easily, the application and database layers should be separated. This is because the dynamic scalability works well for the application components, however the database operations do not react well to the change of hardware resources, particularly on the number of processors. RDBMS which is a type of *Parallel Database* widely used today, hardcodes the startup parameters from the operating systems, particularly the number of processors. In order to scale up or down the number of processors in the host, the database services often need to be restarted. Hence, different virtual hosts are needed for each application and database layer. The second point with regards to the portability between Private and Public Cloud is not as rigid from hosting perpective, however the security and budgetary concerns play a major role in deciding such hosting decision. Thirdly, for ease of routing end users' requests, proxy server routing can be configured if necessary. The authors also mentioned about the disadvantage of lack of control to the application codes if the codes are further developed using the service API provided by the *PaaS*, as the portability of the application will be restricted to only the standard provided by the cloud provider. Another disadvantage is the hardware

technology, whereby enhancements and changes to the hardware components are controlled by the provider. The paper also mentioned about load testing, whereby application performance in virtual environment cannot be guaranteed when the resources are scaled up and down. The authors proposed frequent testing on the application performance, which is a challenge to mission-critical transactions as there is not much downtime allowed on the application. From this perspective, the objectives of this thesis strive to shorten and simplify the load testing mechanism, together with a proposed control system to gauge the virtual resource performance. In this way, frequent testing can be carried out without incurring much downtime to the applications and databases.

2.3 Data security

In general, security concerns in cloud computing are considered an enigma, rather than a publicly acknowledged problem. From the literature reviewed in this topic, there is no real technical issue discovered that should hamper the embracement of cloud hosting. There are solutions to all the technical challenges. For instance, Amazon is taking a step forward by introducing AWS GovCloud (US), which is hosted in Amazon Web Services (Amazon, 2012). Its compliance with US International Traffic in Arms Regulations (ITAR) and Federal Information Processing Standard (FIPS) Publication 140-2 is hoped to prove to the world its robustness of data hosting in Public Cloud. Google claims its strength in data security via ten components of Google's multi-layered security strategy incorporated in Google Apps(Google, 2010). Oracle through its Exalogic Elastic Cloud product provides similar offering for Public and Private Cloud, plus Hybrid Cloud that is capable of Cloud bursting (Oracle, 2011). Rather, it is a perception that placing data in cloud, especially the Public Cloud, will create the issue of security and privacy breach. Such phenomenon can be equated to the *Horseless Carriage Syndrome* as observed by Harms and Yamartino (2010).

The concern with regards to security and privacy issues for Healthcare industry is studied in this thesis, as this industry generally has more stringent demands in safeguarding the data as compared to other industries. Lupse et al.(2012) proposed Private Cloud environment to host centralized patient data, where the data is disseminated to various location via HL7 Clinical Document Architecture (CDA) messages, via the Service Oriented Architecture (SOA). Remote access to these Protected Health Information (PHI) and Personal Identity Information (PII) information via this architecture is critical, in order to ensure patient data is available when patients are transferred, or in situation when the medical personnel in charge is on vacation. The reason provided for the preference in the choice of Private Cloud is that the medical data can only be accessed by medical personnel in such platform. In view of the Health Insurance Portability and Accountability Act (HIPPA) requirements, safeguarding confidentiality and integrity of the patient data is of utmost important, hence the sharing of Electronic Medical Records (EMR) cannot be compromised under any circumstances. Thus, Private Cloud is proposed to avoid the perceived security risk in Public Cloud. The ideal data dissemination model which utilizes Private Cloud envisaged by the authors is illustrated in figure 2.5.

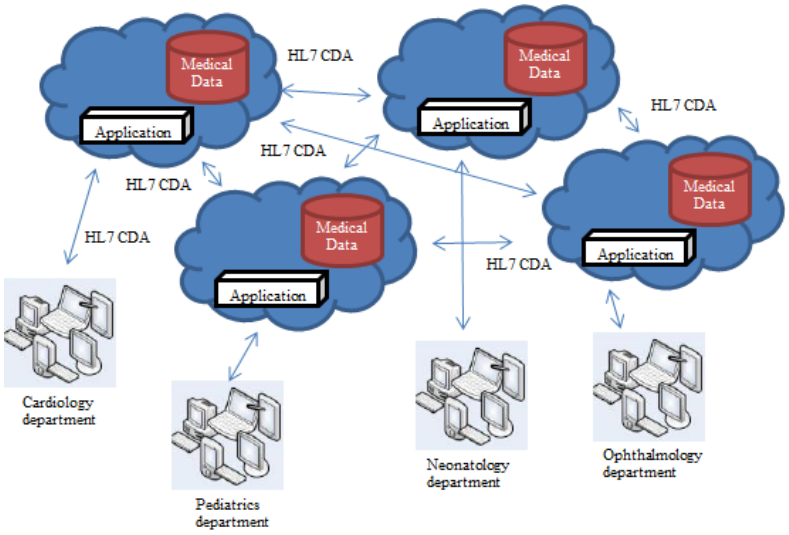


Figure 2.5: Data hosting architecture proposed for hospital systems. Adapted from (Lupse et al., 2012)

Ahuja et al. (2012) quoted a list of security measures from Cloud Security Alliance (Cloud-Security-Alliance, 2010) to enhance the relevance of Public and Private Cloud to support healthcare organizations. From the observation of current trend, Private Cloud would most likely be implemented first. Apart from setting up their own on-premise Private Cloud, enterprises can engage commercial vendors to provision Private Cloud instances for them. IBM provides such service in their product called IBM SmartCloud Application Services (Saugatuck, 2011). Subsequently when the security perception on Public Cloud has improved, these healthcare applications can make their way to public infrastructure easily with the standard API from these established providers (Wan, Greenway, Harris, & Alter, 2010). The suggested implementations for healthcare industry are multi-layer of login authentication, robust administrative capabilities to assign appropriate privileges to users and groups, strong password creation and encryption, encrypted data exchange and federated authentication. Asides from these security measures, the authors also suggested robust backup and disaster recovery policies in order to comply with the strict availability requirement of medical data. In this case, cloud providers are regarded to be more equipped as compared to local data centers managed by individual organizations. It is also suggested that a specialized cloud to be created solely for Healthcare organizations with tight security architecture. Apart from the security and robustness concerns, the interoperability among some decade-old applications with cloud standards is of concern. The applications in the healthcare industry mostly do not evolve as rapid as common applications deployed in other industries. Hence these applications are older and different in term of database design, operating systems, programming languages, platforms and data formats (Myers, 2012). The cloud providers who are able to adapt to these standards, or provide a smooth migration strategy to these applications, couple

with fulfillment of the requirements outlined by HIPPA in the *IaaS*, *PaaS* and *SaaS* models will command the market share in healthcare application hosting.

Kumar et al. (2012) indicated that the major element that delays the cloud adoption in healthcare, is the issue of *trust*. Despite the fact that cloud providers have sophisticated methods and utilities to maintain high level of data security, the consumers in the Healthcare industry still stays skeptical. The authors also touched base on the surveys carried out recently by some prominent cloud providers. In particular, *IBM's Institute for Business Value 2010 Global IT Risks Study* (Ban, Cocchiara, Lovejoy, Telford, & Ernest, 2010) revealed that 77% of surveyed participants believed that cloud hosting would jeopardize data security. About 50% were having more negative perception, where they thought data loss would occur if data is hosted in cloud.

Both papers published by Kumar et al. (2012) and Ahuja et al. (2012) made reference to Cloud Security Alliance. In the latest survey results (Gray, Los, Shackleford, & Sullivan, 2012), 50% of the respondents were gathered from United State of America, 8.6% from India, 5.5% from UK, 4.1 from Canada and 31.8% from the rest of the world. The new top threats ranking is much different as compared to the survey result carried out in year 2010 (Cloud-Security-Alliance, 2010), despite the threats relevance still stays almost the same. The new ranking in the survey is categorized as follows:

1) Data Loss/Leakage
2) Insecure API
3) Malicious Insiders
4) Account/Service and Traffic Hijacking
5) Abuse of cloud computing
6) Unknown Risk Profile

7) Shared Technology Vulnerabilities

8) Distributed Denial of Service (DDoS)

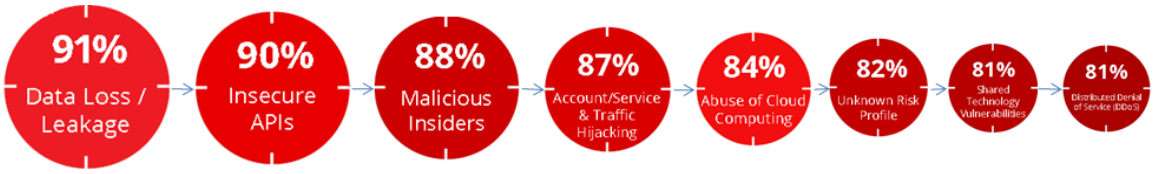


Figure 2.6: Top threats relevance. The security relevance is surveyed for cloud computing. Adapted from (Gray et al., 2012)

Figure 2.6 demonstrates the relevance and significance in percentage of the threats based on the feedback from the vast community of cloud consumers. Nevertheless, the ranking does not rigidly represent all the industries, and the role of the ranking is served only for general reference. For example, for applications that do not use the service API in the *PaaS* model, there is no threat of the *insecure API*. For mission-critical services, the *DDoS* could be ranked higher as connections from the end users to the system need to adhere to strict High Availability requirement.

Data Loss/Leakage is the most significant threat in cloud hosting. The severity level from its damage can be very high, particularly in mission-critical applications. Especially in the case where the cloud provider does not implement robust backup and disaster recovery standard, compromising of data can result in loss of revenue, and even jail times in extreme cases of negligence. This threat can be originated from accidental deletion of data, failure in the virtual hosts that support the databases or malicious activities from unauthorized personnel. The preventive measurement to this cause is to implement access control to the API, enforce encryption on data in storage as well as in transit, control access to the database whereby only relevant personnel is allowed to gain entry to the real data, undertake legally binding agreement between the consumers and cloud providers to ensure vigorous backup and recovery strategies are in place.

Insecure API can happen in cloud service offering, in the case when the management and monitoring in this layer is not proper. The APIs are provided by cloud providers to enable the creation of application in the cloud instance. For *IaaS,* Infrastructure API is serving the function of virtual resource provisioning in the VM. In *PaaS,* service API is responsible in providing the capability to launch the generic database, storage, Exchange and web portals components. Whereas application API offers the interface to software related API, such as CRM, ERP, trading web services etc. In case if these API are compromised by unauthorized personnel, the API can be manipulated to cause undesired damage to the cloud consumers or public. The proposed remediation includes stricter access control to these APIs, re-engineered the security of the APIs and clearer understanding in the relationship between the interconnecting APIs.

Malicious Insiders threat is difficult to be solved, especially if consumers have no visibility to the procedures and processes implemented by the cloud providers. These individuals are hired to work for the cloud providers, and their roles sometimes allow access into the customers' data and transactions. The remediation to this threat is to enforce strict hiring process, establish legally binding employment agreement with the employees, reveal operational processes and procedures to the consumers and generate and disseminate audit trails on security breach.

Account/Service and Traffic Hijacking is the popular intrusion method used by hackers to gain access to the application or database. Such threat exists since the beginning of the computing era. Via the phishing, social engineering and exploitation of application or database vulnerabilities methods, data is manipulated, and falsified information is returned to the clients or redirecting the clients to illegal web sites. The remediation solutions to this mischievous act are to introduce 2-factor authentication, proactive monitoring of abnormal users' activities, establish auditing trails on user's transactions

and to understand better the services offered by the cloud providers, by scrutinizing on their security policies and procedures.

Unknown Risk Profile refers to the ambiguity of cloud providers in providing full exposure of their service offering. Often time in order to secure their business interests, cloud providers are reluctant to reveal the total architecture landscape to the consumers or outsiders. Without the full knowledge of the hosting platform, the customers are left with unknown risk profile, where they do not know the exact level of exposure to danger regarding their data. In this case, before engaging the service from the cloud providers, the customers should be equipped with knowledge on how their data and related audit logs are stored and who have access to this information, how much details will the providers provide in case of security breach, the level of hardening and patching on the underlying hosting hardware and software and the frequency of auditing and logging in the cloud instance. If these factors are overlooked, the consumers will be left with potential disaster which they do not anticipate.

The *abuse of cloud computing* mainly targets the *PaaS* model, where fraudulent credit card is used to purchase the cloud instance from the providers, and the temporary platform is used to conduct malicious and even criminal activities. Apart from *PaaS, IaaS* model is also targeted for similar intention. The proposed remediation to this vulnerability is to enhance the registration process of cloud instance provisioning, more stringent checking on fraudulent credit cards, auditing on consumers' traffic and close checking on blacklisted individuals.

Shared Technology Vulnerabilities threat refers to issue in multi-tenant architecture. Often in the VM, CPU resource in particular is not isolated between different clients hosted by the same application or database. Hence there are chances that some clients might overuse this resource and create resource constraining situation in the VM. As

such, strong resource allocation and isolation mechanisms are needed, together with effective monitoring of the resource consumption in the VM. The CSA community also proposes regular patching and vulnerability scanning in the VM to fix and discover abnormalities before the tenants' transactions are impacted.

Distributed Denial of Service (DDoS) is usually the result of a *Trojan horse* attack. The web sites affected will have their services interrupted and their clients will not be able to gain legitimate access to them. This often happens in high-profile web pages, for example the online banking, credit card payment gateways, government portals, news portals etc. The motive to these attacks is to gain competitive advantages, either in business, politic or social domains. It can also happen to exhibit protest on certain disgruntle issues by certain groups of people. Remediation to this treat involves a wide range of computing variables. Cloud providers can strengthen the rules in their firewalls, switches and routers to prevent unauthorized access and activities, installing robust Intrusion Prevention System (IPS) (Cisco, 2013; Fortinet, 2013; HP, 2013b) at the network layer in their hosting landscape. This includes activities called "blackholing" where traffic is routed to a 'black hole' when attacks are detected, "sinkholing" where traffic is channeled to another IP address when abnormaly is detected and 'cleansing' where the good and bad packets are monitored and filtered.

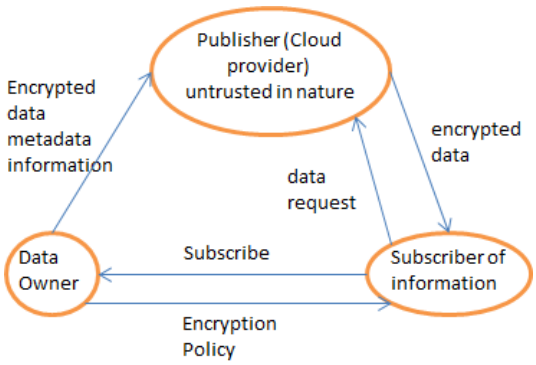


Figure 2.7: Secure third-party publication. The subscriber requests and receives data from the publisher, and verifies the authenticity of the data using the encryption key from the data owner. Adapted from (Hamlen, Kantarcioglu, Khan, & Thuraisingham, 2010)

Hamlen et al. (2010) scrutinized security in cloud environment, particularly at the storage layer. The authors proposed security mechanisms to protect the data in storage as well as in transit. For data in transit, the data is requested from the publisher, which in this case is the data storage component in the cloud. Since the data is hosted in the cloud, the machine that stores the sensitive data is assumed insecure. When the information is requested from this cloud storage, the subscriber sends a message to the data owner to obtain the encryption policy regarding the requested data. As soon as the data arrives, the subscriber compares the encryption policy sent by the data owner to the encryption key information tagged together with the delivered data. When a match is found, the authenticity of the data can be confirmed. Figure 2.7 illustrates this sequence. This way, the owner, or the patient in this case can control the amount of data exposed to the subscriber. For the data in storage, the authors proposed to utilize the Secure cryptoprocessor (SCP) (IBM, 2013a) as part of the cloud infrastructure to enhance the robustness of the stored data. The purpose of SCP is to protect the data stored in the publisher, by eliminating other protection requirements at other physical components in the storage machine. The SCP does not output the decrypted output to the system bus, hence even the authorized personnel cannot tamper into the underlying data. As the storage of the encryption and decryption information is solely stored and confined to the SCP, the data owner can be rest assured that the encryption key that is sent to the publisher is correctly utilized to decrypt only the necessary data. Any attempt from hackers to steal these encryption and decryption keys in the SCP will need the physical possession of the device, as well as skills and tools which are beyond the technical knowledge of most hackers. Figure 2.8 illustrates the placement of SCP in a storage machine.

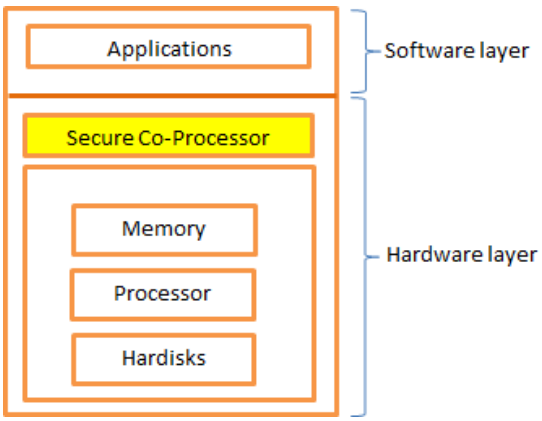


Figure 2.8: Secure cryptoprocessor. The SCP is a temper-resistant hardware that eliminates the need to harden other components in the hardware layer. Adapted from (Hamlen et al., 2010)

As in figure 2.7, the data owner is required to send the encryption key to the cloud provider in order to authenticate the legitimacy of data request. The encryption key widely deployed nowadays is in the form of a smartcard. However, there are some drawbacks in utilizing smartcards for authentication purpose. Karthikeyan et al. (2012) pointed out that smartcard can be stolen or replicated easily, hence is deemed insecure. They proposed an authentication method using palm vein pattern recognition. The reason to use the palm for pattern recognization is that the human palm has complicated vascular pattern, thus is able to hold many differenting characteristics for personal identification. As the pattern of the blood vein lies under the skin, this pattern discovery method is deemed more secure compared to thumb print recognition.

Personal Health Record (PHR) is a fairly new term used in medical record exchange domain. It contains the personal health and medical history of individual, which can be shared to interested parties that possess the necessary credential. PHR is initiated and maintained by individuals. It collects its records from Electronic Medical Record (EMR) and Electronic Health Record (EHR). EMR is the complete medical record stored when an individual engaged inpatient or outpatient treatment in a hospital. EHR is a subset of EMR, owned by the patient but is maintained by each hospital. The clear difference between EMR and EHR is that EMR is not modifiable by individuals, but

EHR can be appended by individuals. Figure 2.9 illustrates the overlapped relationship of these 3 types of medical records (R. Zhang & Liu, 2010).

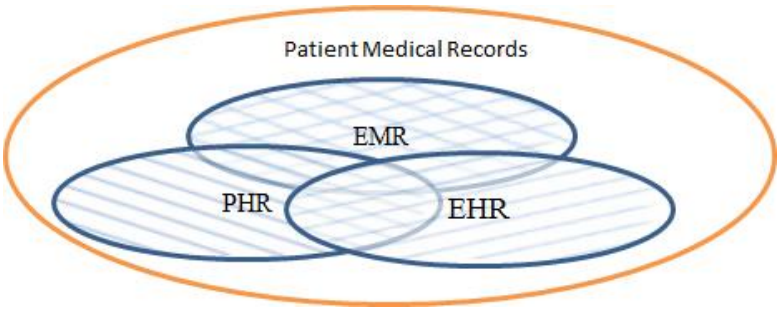


Figure 2.9: Patient medical record categories. Overlapped in relationship of PHR, EMR and EHR is illustrated. Adapted from (R. Zhang & Liu, 2010)

The signficance of presenting this information here, is that the current trend in cloud hosting in Healthcare industry is focusing on the 'patient-centric' model. Figure 2.10 exhibits such model. The PHR is collected by individuals from EHR data in cloud, and subsequently this information is filtered and disseminated to authorized personnel. In the PHR data dissemination domain, scholars (T. S. Chen et al., 2012; M. Li, Yu, Zheng, Ren, & Lou, 2012) employed similar model as in figure 2.7 to encrypt and decrypt exchanged data between the individuals and the subscribers. The encryption and decryption algorithms are not elaborated in length here as the purpose of quoting these researches is to outline the general current trend and progression of the cloud migration strategies that happen in the industry. Microsoft provides such PHR services in Microsoft health Vault (Microsoft, 2013b). However it is to note that such services in the cloud are not totally adhered to HIPPA rules, hence this is yet another foreseeable potential development in the HIPPA regulations to encourage the healthcare players to move to cloud.

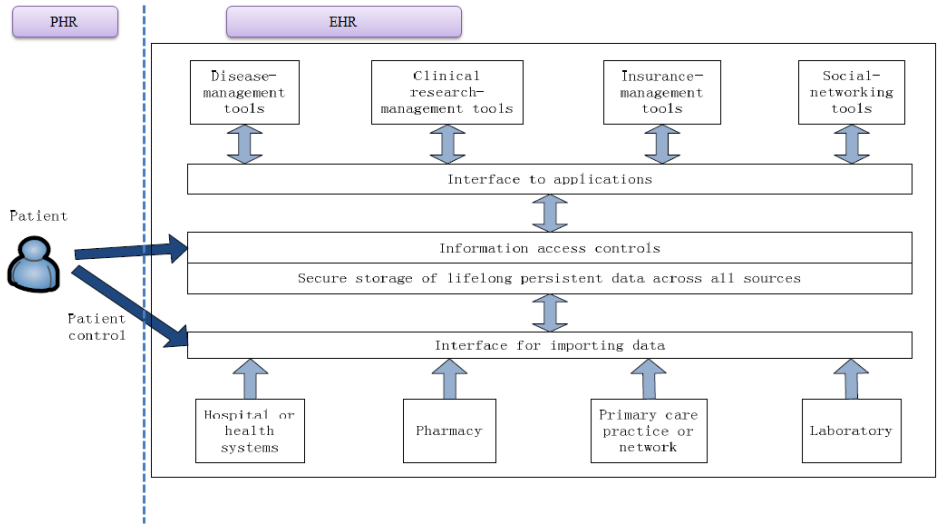


Figure 2.10: 'Patient-centric' cloud model. Individuals retrieve their personal health and medical data from EHR stored in cloud, and then categorize and disseminate the information to related parties. Adapted from (R. Zhang & Liu, 2010)

Donahue (2010) revealed some exciting progress in Healthcare industry, made possible by hosting in cloud computing environment. The authors revealed that the healthcare information technology (HIT) is 10 years behind the development in other industries. The reasons cited are the lack of knowledge in the healthcare industry players to fully understand and embrace the IT technologies, together with the skepticism on security and privacy issues on medical data. The priority at the moment is to digitize the medical records on paper format into EMR. The author outlined few conditions for the successful implementation of this initiative, with absolute precaution being considered for sensitive patient data:

1) Capital expenditure spending needs to stay as efficient as possible.

2) The new cloud architecture must permit sharing of data among different remote entities without combersome protocols.

3) Scalability criterion must be in place for future expansion.

4) High Availability feature must be incorporated into the new architecture.

As the perception that enterprise computing environments are not suitable for Healthcare industry, a separate community cloud called *hcloud* is proposed, that solely cater for the players in the Healthcare sector to achieve the above 4 goals.

Cardeñosa et al. (2012) also stated that the biggest challenge to migrate legacy and in-house healthcare application to cloud is the *trust* issue among the industry players and cloud providers, in hosting the medical data. The authors conducted some feasibility analysis on the hosting of EHR in the Public Cloud using Amazon Web Services (Amazon, 2012), and realized that another challenge with this implementation is the bandwidth issue between the medical personnel in the hospital and the Public Cloud. They discovered that another unsuitability reason to host data in the Public Cloud is that most medical personnel, whether they are situated remotely or in the hospital, the WAN connection to the Public Cloud instances is poor. As the medical data involves a lot of Digital Imaging and Communications in Medicine (DICOM) images transfer, the network pipeline will need major increment, which involves huge investment. Such scenario will discourage many migration efforts in the industry. In order to remediate this shortcoming, the authors proposed a Hybrid Cloud architecture, where text data is stored in the Public Cloud, whereas Private Cloud is established to host the DICOM images. Figure 2.11 exhibits their proposal. With such model, the applications will need to be recoded to connect to proper interfaces which also command a substantial amount of investment.

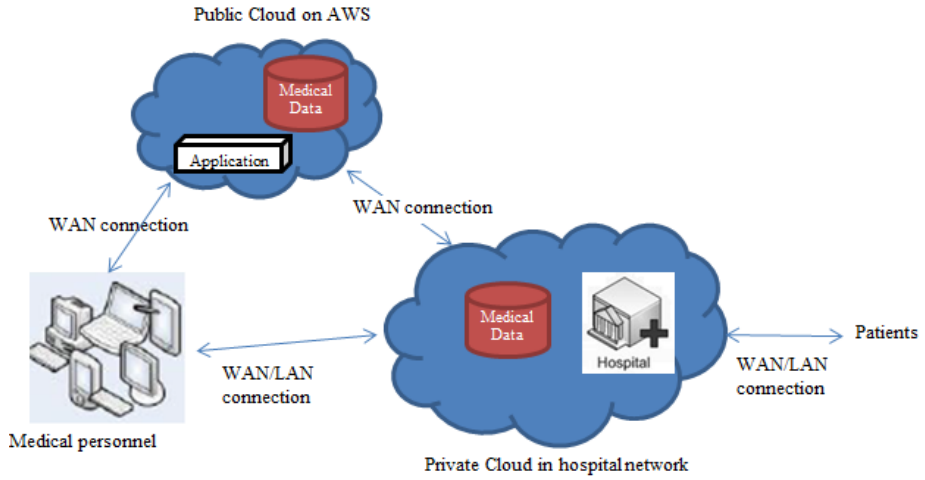


Figure 2.11: Proposal of cloud hosting for medical data. Due to large DICOM image transfer requirement, Hybrid Cloud model is deemed more suitable. Adapted from (Cardeñosa et al., 2012)

From the perspective of the research in this thesis, the proposed resource management mechanisms contribute to tighter access control in real data. It achieves one of the major initiatives in protecting and controlling access to sensitive data either in storage or in transit. By prohibiting such access to the IT personnel, the proposed mechanisms enable IT services to continue serving their functions, at the same time improve the *trust* factor among the cloud consumers. From the commercial perspective, there are products that allow for the masking of customers' data, so that IT functions can be performed without jeopardizing the privacy and security of these real data. Oracle Database Vault (Tbeileh, 2009) is such product that promotes the compliance with security regulatory requirement and reduces the insider threat.

2.4 Resource utilization *monitoring*

2.4.1 Monitoring models and on-demand resource scaling

One of the very attractive features of cloud computing is the ease of scalability in resource provisioning and de-provisioning in the VM. Resources in cloud can be scaled up or down easily, primarily due to the virtualization concept. Depending on the types of software deployed in the VM, the resource allocation mechanisms can generally be categorized into 2 classes: first, it is the dynamic on-demand resource allocation in the

VM, where needed resources are allocated rapidly when the resource threshold is detected. Such method allows the VM to have its resource state altered as frequent as needed. The second type is not as dynamic, where the additional resource is staged, and a restart on the software is needed to take advantage of the additional resources. The formal type is currently suitable for web applications, where dynamic scaling of resources can be detected by the application instantly to take advantage of the additional computing power. The latter is applicable mainly for RDBMS software and certain CRM applications, where the software needs to recognize the additional provisioned hardware by rebooting. Main reason to this scenario is that these software are licensed based on the amount of provisioned hardware, and these vendors do not allow their products to be used indefinitely without the customers paying for a premium when the new slice of additional resources is run against their products. As this research deals with database operations in the VM, the focus is on this latter resource allocation mechanism. Nevertheless the proposed mechanisms are applicable for on-demand scaling type, amid some modifications. It is also to note that the newly introduced Oracle 12c (J. Williams, 2013) has the feature to allow their customers to enjoy the pay-per-use licensing model. However at the time of this thesis writing, this product has just been released and the cost feasibility of its implementation and its acceptance level by consumers in cloud environment are still at evaluation stage.

In both the resource allocation categories, the resource allocation methods can be performed by 3 techniques. The first technique is the horizontal scaling of the application tier by mean of load-balancing. Web-based application can normally benefit from this scaling method. The second approach to increase the resource allocation in the VM is by mean of vertical scaling, where resources are added to the particular VM. The third method is called VM placement, which is also considered in the category of vertical scaling. However instead of adding resource into the VM, the VM is migrated

to another physical machine which has a larger or smaller hardware configuration (Iqbal, Dailey, & Carrera, 2010). In order to achieve efficiency in determining the trigger point when resources are to be allocated or de-allocated, monitoring of resource consumption is of utmost important. A profound monitoring mechanism will need to be able to process the collected data from the past, present and future resource utilization. In this section, the studies examine the monitoring agents deployed in both the web applications and databases, as the hardware monitoring parameters are similar for both worlds.

Iqbal et al. (2010) experimented and tried to proof the advantages of horizontal scaling in comparing to vertical scaling. The hosting platform of the experiments is on Eucalyptus Cloud. As explained early in this chapter, Eucalyptus(Eucalyptus, 2013a) is an open source software that provides dynamic scalability feature in building the Private and Hybrid Cloud using Amazon Web Services API. To simulate the workloads, Httperf utility is employed. Httperf (httperf, 2013) is a tool usually used by industrial players to create artificial workloads in their web servers to investigate and evaluate resource capability, as well as generating benchmark for future reference during production operations. The authors segregated the web and database tiers into 2 VM initially. The initial objective is to detect the bottleneck in the VM, so that the response time SLA can be adhered to by adding resources to the VM for further transaction processing. When the response time threshold is detected at the front-end, the suggested mechanism first determines if the bottleneck is originated from the web tier. If it is, the resources in this tier are scaled. If there is no constraint detected in this tier, the database tier is scaled instead. The threshold is breached when 95 percentile of average response time of dynamic and static content requests is above the stipulated value, based on moving average calculation. The experiments reveal some interesting facts. For vertical scaling, where resources are added to the VM when threshold is detected, it is observed that the

throughput saturated at certain CPU run queue level, as illustrated in figure 2.12. The throughput never increases even the CPU utilization never gets saturated in both web and database VM, as in figure 2.13.

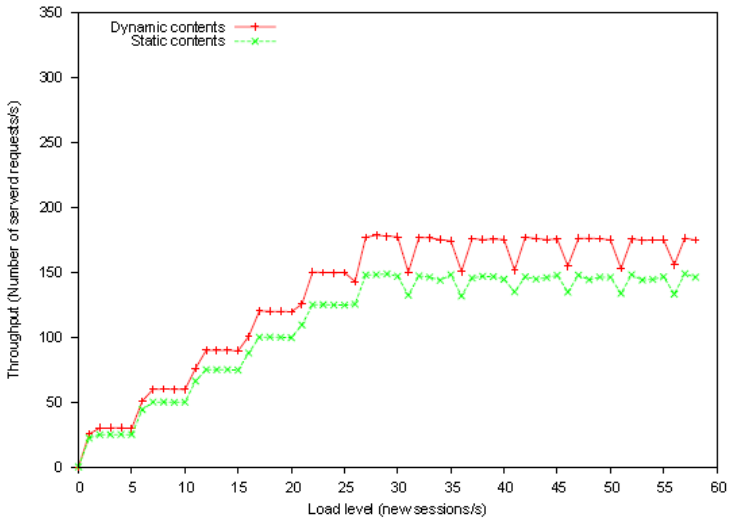


Figure 2.12: Throughput of the system with vertical scaling of resources. It is observed that the throughput is saturated at certain CPU run queue level even the overall CPU utilization is not constrained. Adapted from (Iqbal et al., 2010)

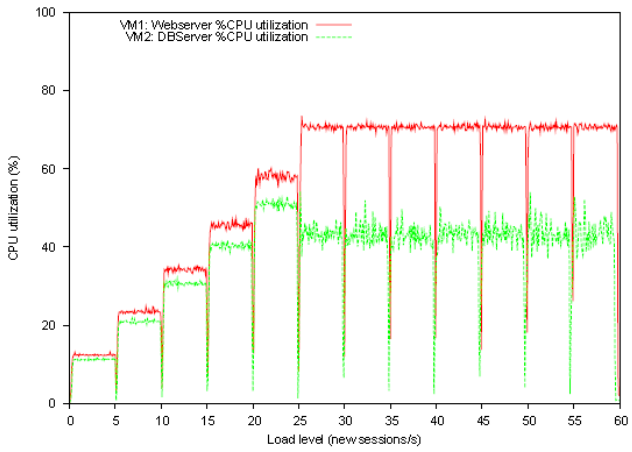


Figure 2.13: CPU utilization in both the web and application VM during the test of vertical resource scaling. Adapted from (Iqbal et al., 2010)

However, in the second experiment when the horizontal scaling of resources is conducted, the throughput increases in tandem with the additional of extra VM into each tier, as exhibited in figure 2.14.

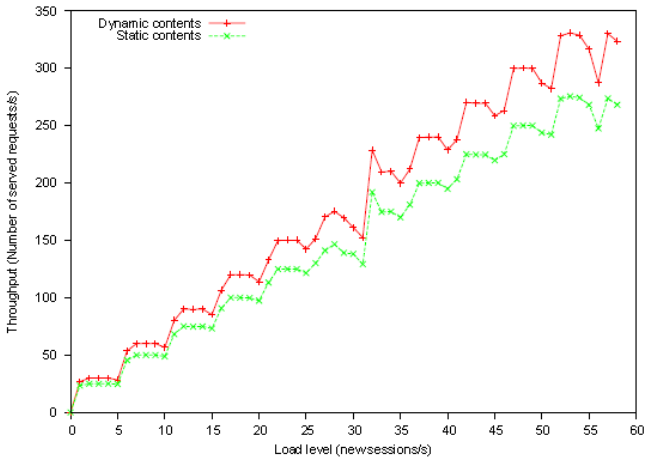


Figure 2.14: Throughput keeps increasing with the addition of VM. Adapted from (Iqbal et al., 2010)

Hence, apart from proposing a simple, yet perceived effective way to guarantee response time stipulated in SLA, the authors also pointed out that certain cloud platforms will post bottleneck in the hosting backbone, where usual resource parameter addition in CPU, memory, storage and network cannot elevate the performance of the VM. To alleviate this constraint, horizontal resource scaling is inevitable. Another solution is to engage VM placement mechanism, where the VM is migrated to more powerful cloud platforms.

Chieu et al. (2009) presented a typical illustration of dynamic resource scaling mechanism for web-based applications. The architecture for the test bed in their experiments is similar to the one proposed by Iqbal et al. (2010). Figure 2.15 depicts the components in the architecture. The web servers are scaled horizontally by adding VM, based on the resource needs. The main reason for such architecture to work for web applications very much depends on the capability of the load-balancer. Theoretically, the Apache HTTP Load-Balancer can be re-configured on-the-fly when there is change in the underlying web application configuration. Thus, when the number of VM running the web services is altered, the load-balancer can react automatically to route the traffic to the new virtual web machine. Radware (Bercovici, 2010) is one such agent available

commercially to orchestrate this elastic horizontal scaling and load balancing, and it is integrated with the cloud provider management system. It is to note that the database tier does not have such capability due to the design of most RDBMS; hence dynamic on-demand scaling is generally not working in this tier.

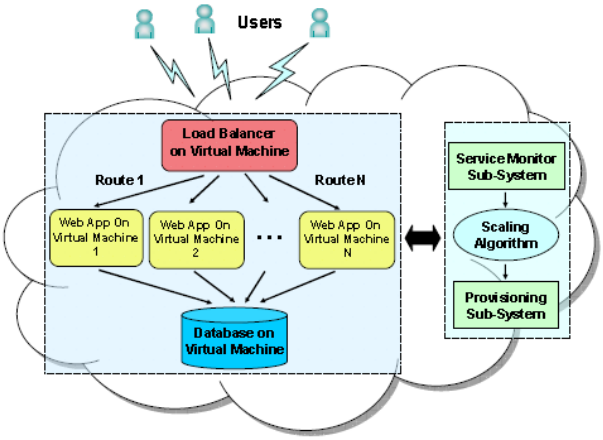


Figure 2.15: Dynamic scaling of web-based applications. Such dynamic on-demand scaling is suitable for the application tier. Adapted from (Chieu et al., 2009)

The horizontal scaling of the virtual web machine is accomplished via "Image-based provisioning". This capability is not a new technology in the cloud; rather it has been available in the industry for a while. Many operating system vendors have this feature in their products, for example Red Hat Enterprise Linux (IBM, 2013b) and Microsoft Windows Server (Microsoft, 2013a). This ability to have the additional VM created automatically and instantly is made possible via the cloning of the new machine from a 'golden' virtual image. The authors mentioned that there are 4 criterions to determine the threshold points of the virtual web machines. They are:

1) The number of concurrent users
2) The number of active connections
3) The number of processed requests/s
4) Average response time/request

However, criterions 1-3 might not be suitable to be employed in cloud environment. The reason is that in cloud computing, commodity hardware is utilized, in contrast to conventional hosting where web servers are normally having homogeneous hardware configuration. In cloud, the collection of hardware that comprises a resource pool can be heterogeneous; hence the additional provisioned virtual machine can have difference in computing capability. Due to this reason, the scaling algorithms should not rigidly lock in certain values of concurrent users, active connections and 'total running processes/s' as the threshold to gauge the VM performance. The 'average response time/request' parameter should be the more accurate barometric indicator in this case. It is also important to note that in virtualized environment, the CPU and memory utilization parameters are sometimes misleading; hence silo monitoring solely from operating system perspective will not guarantee accurate result (V. Kumar & Garg, 2012). The combination of monitoring techniques, by taking the response time parameter in association with the operating system parameters is more appropriate.

The method of rigidly locking in the threshold values as input for scaling decision in a VM is also disputed by Dutreilh et al. (2010). This threshold-based scaling policy is deemed inaccurate, as it promotes wastage of resources, due to the mismatch between the control system and control parameters utilized to arrive at the scaling decision. There are 3 sources cited for this misalignment:

1) Latency in attaining the stable performance condition after the scaling output is performed, thus incurring an unstable duration for the subsequent scaling algorithm.
2) Oscillation and instability in the input control parameters, which reduce the accuracy of the scaling decision.
3) Too aggressive scaling algorithm, where the magnitude of resource scaling does not represent the actual transactions' requirement.

To remediate the above, the authors proposed Markovian decision processes (MDP) model to manage the resource scaling decision. This model relies on reinforced learning in order to produce a collection of states and actions in the VM. With this model, the resource scaling decision inclines more towards the condition in the VM, instead of relying solely on the fixed threshold values which commonly decided upon the state of the VM during initial application deployment process. MDP is an extension of Markov Chain (Bolch, Greiner, Meer, & Trivedi, 2006). The high level illustration of Markov Chain model is as exhibited in figure 2.16.

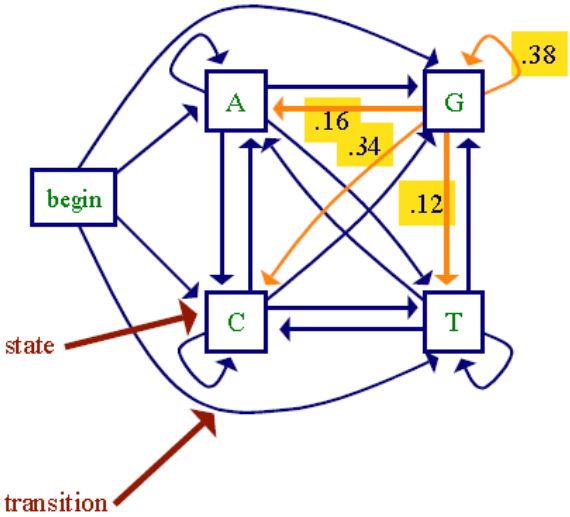


Figure 2.16: Markov Chain model. The states (A, G, C & T) and the transition probabilities provide input for the next course of action.

For illustration purpose, by equating figure 2.16 to the real scenarios in the VM, there are 4 potential states in the VM. The transition probability from 1 state to another is depicted as below:

$$P(x_i = a \mid x_{i-1} = g) = 0.16,$$

$$P(x_i = c \mid x_{i-1} = g) = 0.34,$$

$$P(x_i = g \mid x_{i-1} = g) = 0.38,$$

$$P(x_i = t \mid x_{i-1} = g) = 0.12.$$

By relying on the probabilities of each transition, the cloud providers or the consumers can determine the necessary actions, whether to proceed with scaling, stay with the status quo, or delay scaling decision for *t* amount of time. However, there is a caution to be noted here, that Markov Chain model is suitable in cases where the 'states' are not dependent on each other. In situations where independence of states cannot be assumed, this model should be avoided.

As mentioned above, the over-aggressive scaling algorithm is not desired, as it induces wastage in resources due to the reason that the state in the VM is not properly understood before the scaling is performed. The same sentiment is echoed by Belaglazov et al. (2012). The authors even cited that quality of VM migration and consolidation is inversely proportional to the number of nodes in the cluster. In other words, it is best to not encourage the scaling algorithm to reduce overhead in VM operations. With a predefined set of Quality of Service (QoS), the algorithms proposed by the authors strive to achieve the maximum mean time between VM migrations for this purpose. To achieve this, they introduced a parameter called Overload Time Fraction (OTF), which denotes the maximum timeframe where the host is allowed to stay in overload condition. To calculate the OTF, the authors too employed the Markov Chain Model. The states in this case correspond with the CPU utilization in the VM, and the associated transition probabilities are the chances of the potential migration.

In many of the researches, the CPU utilization threshold is considered as the most important parameter to be measured to determine the oversubscription of resources in the VM. In this thesis, the CPU run queue size too is employed to serve the barometric measurement on the resource states in the tested VM.

Khatua et al. (2010) introduced event-based scaling algorithms in their proposed Monitoring and Optimizing Virtual Resources (MOVR) architecture. The standout

feature in MOVR is the ability to map each event to an action, called ‘workflow’ in the paper. These workflows determine the appropriate actions to be carried out in the VM; for example to scale up or down, horizontal or vertical scaling of the computing resources. There are 4 events that guide the scaling decision. First, it is the threshold-based event occurrence, which is similar to the resource threshold trigger as discussed above. The second event inclines towards prediction-based, where the system will base on the historical data, to forecast the resource scaling timeframe and duration. For instance, the performance evaluation applications are active particularly at the end of each financial quarter, or the payroll processing is active for few days before payment to the employees. The scaling mechanism will base on the historical data to decide for additional resource allocation to the VM. The third event is request-based, and the scaling decision is made based on the length of request queue in the applications. If the queue is hitting a certain threshold, the scaling of resource will be triggered. The fourth event is simpler, where it is based on the scheduling decision of the administrators, regardless of wherther or not there is resource constraint in the VM. Such event and workflow bases can be expanded to included more criterions, which can increase the efficiency of the on-demand scaling model.

The determination of the appropriate threshold value before the VM is deemed fully loaded very much depends on the QoS required by the particular applications. This is because when threshold-based system is being put in place, only the mean value of the threshold can be utilized as the real resource usage pattern in the VM. The OS and database parameters oscillate quite substantially during the monitoring process as indicated by Dutreilh et al. (2010). Hence, even though the resource utilization is < 100%, there are still chances for some transactions to spike the resource usage to > 100% which cause violations in the application SLA. Beloglazov et al. (2012) and Buyya et al. (2010) realized such condition in the host, and they conducted experiments

to show such behavior. The experimented results can subsequently be employed by their recommended task or resource scheduling mechanisms to determine the best scheduling algorithms to achieve the objective of optimizing the resource usage, at the same time comply with the SLA requirements. Their experimental results are exhibited in figure 2.17. It is expected, and confirmed from this figure that the higher the mean value of the utilization threshold, the higher chances for SLA violation, due to the fact that some transactions in the workload spike the resource usage until the constraining level. The data from this diagram is not representative for all workloads and VM; however such experiment should be carried out in all environments to determine the most accurate resource threshold value that maximizes the resource utilization and at the same time confines the SLA violation to acceptable level.

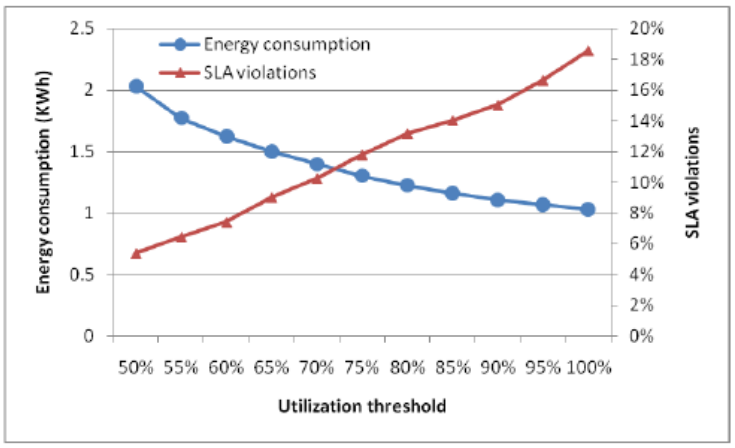


Figure 2.17: The utilization threshold versus SLA violation for particular workload. The authors were trying to map the energy consumption to these 2 parameters. Their objective here is to produce a green cloud computing architecture by discovering the equilibrium between the 3 parameters. Adapted from (Buyya et al., 2010).

From commercial perspective, these resource monitoring and scaling tasks are performed by the cloud virtualization toolkits, which are explained in section 2.2. In enhancing such enterprise level products, Gulati et al (2011) studied and extended VMware DRS solution for resource management purpose. They examined particularly the *reservation, limit* and *share* attributes offered by DRS utility. In this case, *reservation* denotes the pool of resources offered to an organization in the cloud, for

scaling purpose. *Limit* sets the boundary of amount of resources permitted to be used by a VM. Whereas *share* signifies the relative importance of a unit of resources for a particular VM, so that more important transactions can access to resources in constraining situation. The explanation of these 3 attributes is depicted in figure 2.18. The significance of their study is in the improvement in the DRS load balancing, where the benefits of migration and the cost of performing the migration are calculated automatically, to trigger automatic VM placement.

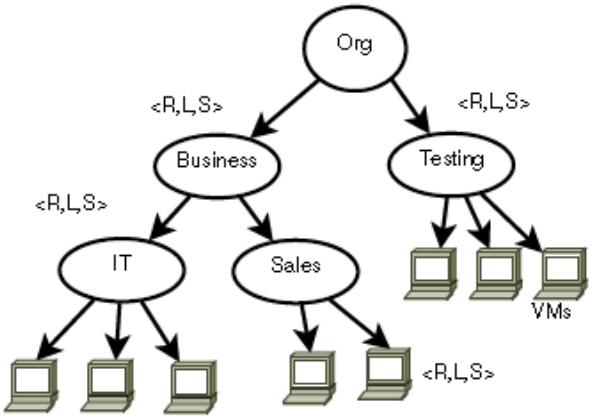


Figure 2.18: Resource allocation to an application for an organization in cloud. Each department is allocated a resource pool, a resource usage limit and relative importance in the transactions carried out. Adapted from (Gulati et al., 2011)

Microsoft releases similar product as VMWare DRS in System Center Virtual Machine Manager (VMM) (Microsoft, 2013d) specifically on Windows platform. Oracle provided an integrated enterprise cloud IT management tools in their new release of OEM 12c (Oracle, 2013). ConVirt Enterprise Cloud (Convirture, 2013) is another popular product in this category. Nevertheless, there is not a way to determine the *potential* breach of resource utilization threshold in the VM itself based on real transactions in the individual *VM*, rather a macro view is perceived by these utilities, based on historical or current workload inputs. From this perspective, the resource usage *monitoring* and *planning* scheme in this thesis fill in this gap, to provide an insight and predict the resource consumption from real transactions in the individual VM.

2.4.2 Resource scalability in *Parallel Database* architecture

At this time of the writing, dynamic on-demand resource allocation is only suitable for web-based applications, as explained above. The conventional RDBMS design does not work well with this cloud model because since the early day of the relational database development, the vendors always envisage the licensing model based on the fundamental of available number of processors in the database server. As this matured technology has been in place for so long, couple with the wide adoption and strong reliance on them by the industry, many commercial RDBMS suppliers do not consider a change to this model to take advantage of the elasticity and scalability features of cloud as urgency. Hence, the database hosting in cloud is lagging behind in relative comparison with the cloud web hosting in term of harnessing the full cloud advantages.

In realizing this shortcoming in the database tier, a group of researchers from Massachusetts Institute of Technology (MIT) has started to develop a "database-as-a-service" (DBaaS) model named Relational Cloud (Curino et al., 2011). The main objective is to harness the full advantage of cloud computing for database operations. This model is still in development stage, with the eventual promise to achieve efficiency in multi-tenancy, elastic scalability and safeguard the data privacy. The data privacy factor is developed via a separate project called CryptDB (Raluca Ada Popa, Redfield, Zeldovich, & Balakrishnan, 2011). With these goals in mind, the model is hoped to be able to convert capital expenditure cost to operational cost, by converting the traditional licensing model from processor-based, to usage based. In this case the consumers will need to pay only for the database services that they use, instead of the costly processor licenses. Furthermore, the Relational Cloud's architecture strives to save in hardware and electricity, by consolidating multiple physical databases into a single physical database and multiple logical databases. Figure 2.19 illustrates a high-level depiction of the Relational Cloud architecture. The recent development in relation to resource usage

prediction in this model is published in (Mozafari, Curino, & Madden, 2013). As the architecture encourages multi-tenancy hosting, the challenges as indicated, are on predicting the resource usage pattern and transactions' performance from each tenant, and providing a level of adequate resource isolation so that hardware can be shared among the tenants, but at the same time restricting resource constraint from occurring due to overrun transactions triggered by particular tenants. It is interesting to note that these researchers are not reinventing a whole new type of RDBMS; instead their works extend from current RDBMS offering. For instance, the 'backend node' in figure 2.19 is actually hosting a complete non-modified MySQL database. It is the mechanism on how the transactions flow from the end-users to the backend database; couple with the workload placement in each of the backend database that makes this model impressive. As of today this DBaaS model has yet to make its way to the commercial arena. Nevertheless the presented concepts that promote cost effective usage of database services and reduction in licensing investment will become very appealing to many industrial consumers.

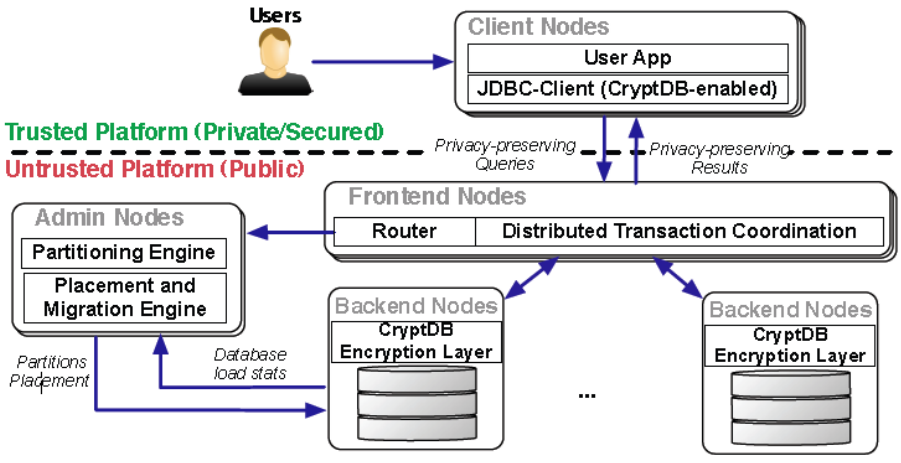


Figure 2.19: Relational Cloud architecture. The model strives to achieve efficiency in resource utilization and reduce the licensing cost. Adapted from (Curino et al., 2011)

As mentioned, the above Relational Cloud proposal strives to improve the usage of cloud for multi-tenancy, elastic scalability and privacy. The benefits of elastic

scalability have been explained. The importance of privacy is also elaborated in details in section 2.3. Many readers puzzle on the significance of the multi-tenancy feature, particularly for IT personnel who have been supporting conventional RDBMS system. In many *Parallel Database* models, 1 database is usually related to only 1 client. Even for shared applications hosted on such databases, where many clients are sharing the same functionalities, it is unlikely that more than 50 clients are sharing one database, due to the resource management and segregation issues by clients.

However in cloud computing, large web-based applications normally are characterized by having small data footprint couples with large number of tenants. As such, the need for better sharing and isolation of resource mechanisms is compelling. This gives rise to the desire to enable a mechanism for ease of moving consumers' transactions around within the available clusters for resource management purpose.

Das, Nishimura, Agrawal, and Abbadi (2010) proposed *live database migration* strategy in the virtual cloud infrastructure to address the requirements for this opportunity. In their test bed, the single virtual machine is hosting multiple databases, serving few thousands clients. The database service is scaled up by constant addition of databases into the VM, and the resource is scaled out by adding nodes into the hardware cluster. The scaling of the nodes is not solely attributed to horizontal addition of hardware to the VM. In this paper, the authors proposed a new approach to migrate the database out to another VM, similar to the VM placement concept that happens at the infrastructure layer. In order for this *live database migration* strategy to work in such multi-tenancy architecture, the clients' data is compartmented into *cell*, where each *cell* denotes a group of self-contained data and metadata for particular client. This atomicity feature of the cells enables the migration to happen. Nevertheless, the proposed migration technique does not move the actual data residing on the SAN storage; rather the authors suggested migrating only the data residing in the memory. In this sense it is similar to

the approach taken by VM migration where the in-memory data is transferred as part of the complete VM migration procedure (Q. Zhang et al., 2010). As such, a service interruption time of only 70ms is achieved. The migration strategy is illustrated in figure 2.20. Apart from this short interruption of service, the advantage of such database migration strategy, compared to conventional stop and start migration method, is that the memory residence data is not erased as the database is never rebooted. Hence the overhead of rebuilding the database cache as part of the stop-and-start migration technique is avoided. The authors conducted experiment on ElasTraS (Elastras, 2013), a cloud database system that supports multi-tenancy application. The concept in figure 2.20 is articulated in this system, as the database architecture allows for ease of read cache migration. It is to note that current development for such database migration method can work only in ElasTraS and it has yet to be commercialized. The current trend on RDBMS development by other prominent database vendors does not reveal plan on such enhancement in their products. Nevertheless this proposal provides a directive on how the constraints in existing *Parallel Database* architecture can be liberated for more efficient cloud hosting.

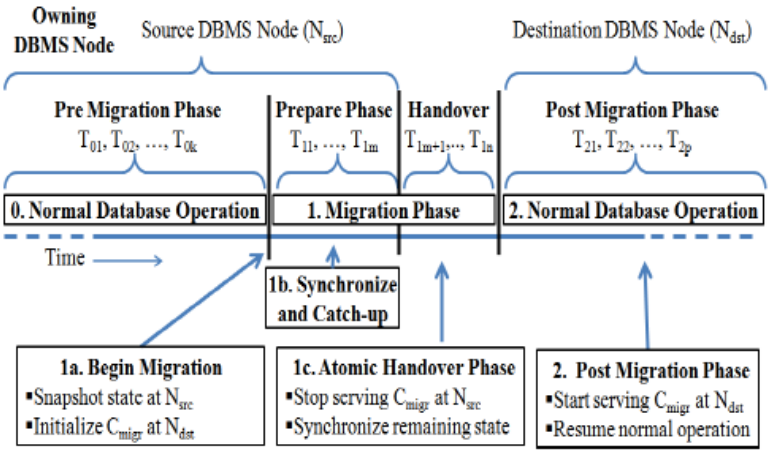


Figure 2.20: *Live Database Migration*. The migration and synchronization is accomplished via iterative copy and update on the database state between the source VM and destination VM. Adapted from (Das et al., 2010)

Table 2.2: Summary of studied researches with critical comment on sub-themes 'monitoring models and resource scaling'.

Scholars	contribution	comment
(Chieu et al., 2009)	Presented a typical illustration of dynamic resource scaling mechanism for web-based applications. Authors mentioned that dynamic scaling is taking the monitoring input from: 1) The number of concurrent users 2) The number of active connections 3) The number of processed requests/s 4) Average response time/request	Criterions 1-3 might not be suitable to be employed in cloud environment as underlying cloud hardware is composed of heterogeneous hardware.
(V. Kumar & Garg, 2012)	Studied on monitoring models. Found out that silo monitoring solely from operating system perspective will not guarantee accurate result.	Authors' discovery in tandem with the research finding in this thesis.
(Dutreilh et al., 2010)	Proposed MDP model to manage the resource scaling decision. The authors disputed rigidly locking in the threshold values as input for scaling decision.	The Markov Chain model is suitable in cases where the 'states' are not dependent on each other. In situations where independence of states cannot be assumed, this model should be avoided.
(A Beloglazov & Buyya, 2012)	Found out that quality of VM migration and consolidation is inversely proportional to the number of nodes in the cluster. The authors proposed algorithm to achieve the maximum mean time between VM migrations for this purpose.	Over-aggressive scaling algorithm is not desired. Proper monitoring models to ensure scaling point.
(Khatua et al., 2010)	Introduced event-based scaling algorithms. Each 'event' is mapped to an action, called 'workflow'.	Such event and workflow bases can be expanded to include more criterions, which can increase the efficiency of the on-demand scaling model.
(Iqbal et al., 2010)	Experimented and tried to proof the advantages of horizontal scaling in comparing to vertical scaling.	For vertical scaling, the throughput saturated at certain CPU run queue level. Horizontal scaling does not have such problem.
(A. Beloglazov et al., 2012) (Buyya et al., 2010)	Tried to map the energy consumption to utilization threshold versus SLA violation. Produced a chart to show the relationship.	Experiment result is convincing, however each workload will demonstrate different behavioral relationship in the chart.
(Curino et al., 2011) (Mozafari et al., 2013)	Developed Relational Cloud, to take advantage of dynamic scaling in cloud.	The presented concepts that promote cost effective usage of database services and reduction in licensing investment will become very appealing to many industrial consumers.

(Das et al., 2010)	Proposed *live database migration* strategy in the virtual cloud infrastructure to address the requirements for multi-tenancy.	Memory residence data is not erased as the database is never rebooted during the migration. Hence cache building phase is avoided.
(Gulati, Kumar, & Ahmad, 2009)	Studied and extended VMware DRS solution for resource management purpose.	The significance of the study is in the improvement in the DRS load balancing, where the benefits of migration and the cost of performing the migration are calculated automatically, to trigger automatic VM placement.

2.4.3 Statistical modeling and benchmarking

2.4.3.1 Proof of concept – the linear correlation

As mentioned, the strength in this research relies on the belief that the combination of parameters from the operating system and database provides an improved monitoring mechanism in arbitrating the resource usage condition in the VM. The 2 most studied parameters are the SQL processing time in the database and the CPU run queue size obtained from the operating system. To integrate these 2 parameters, a linear relationship between them is assumed. To prove this linear correlation between SQL processing time and CPU run queue, following studies are carried out. Such linear relationship forms the cornerstone of all the proposed algorithms in this thesis.

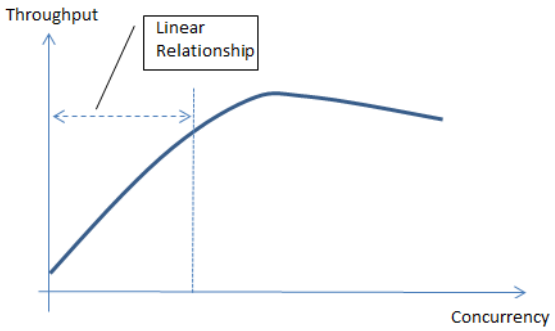


Figure 2.21: Linear correlation between throughput and concurrent processing. Beyond a threshold point, the linear relation is not conformed to. Adapted from (Banga & Druschel, 1997)

As experimented by Banga et al. (1997) and Mosberger et al. (1998), the correlation between throughput and concurrency of processes in a server is linear before a

breakpoint, as illustrated in Figure 2.21. Both papers were experimenting on web transaction processing. In all their experiments, synthetic loads are simulated in the web servers in order to gauge the host performance. This same methodology is widely employed by many researchers in resource management related experiments, including the ones proposed in this thesis. Mosberger et al. (1998) pointed out a very valuable point in their experiments. Conceptually, they assumed that the throughput measurement is as simple as taking the total number of triggered requests, and divided this number with the time it took to complete the test. However in their tests, they realized that the quality of the measurement oscillated quite substantially, and calibrations and adjustments are needed, to discover the stability point before accurate measured readings can be taken. The same observation is true for the experiments conducted in this thesis, where the time to stability in the VM's operating environment, and the quality of measurement duration are important variables. In these cases, heuristic effort is needed to determine the most appropriate values for both before readings can be harvested. Subsequently according to Little's Law of queuing theory (Allen, 1990; Little, 1961), a server's CPU mean queue length, Q is the product of its response time per visit, R and throughput, X, which is $Q = R \times X$. Utilizing these concepts, it is derived that ideally the same relationship will apply to SQL processing time, S and server load, C as depicted in figure 2.22. In the research, the interest is to ensure that the database transactions are processed within this linear correlation to ensure consistency in hardware performance. If this linear relationship is not conformed to, resource contention, hardware performance degradation, total or partial hardware failure, undesired OS processes might have occurred in the host.

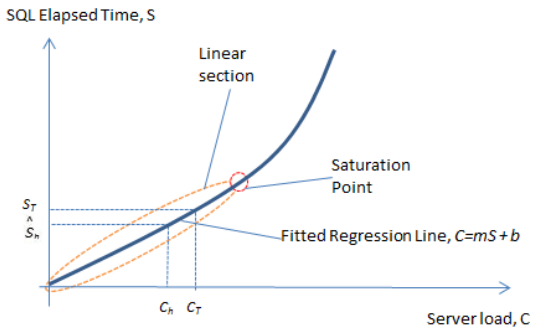


Figure 2.22: Linear correlation between SQL Processing Time, *S* and Server Load, *C*. The behavior of the plot beyond the 'saturation point' is not of interest in this research.

2.4.3.2 Mathematical models

In constructing the resource utilization mechanisms in this thesis, some mathematical models are employed. The fundamental of these models are outlined in this section. The idea of inferring the trend or characteristic from collected data can be properly and accurately represented by mathematical reasoning. The high level depiction of this envisaged methodology is illustrated in figure 2.23. Such method is widely deployed by scholars in many resource management problems. The data collection phase must adhere to following criteria in order to ensure accuracy of applicability:

1) The collected amount of data must be sufficient to describe the investigated real world systems.

2) The data must be relevant to the fundamental principle of the examined features.

3) Noises must be filtered so as not to affect the representability of the data to the studied problem.

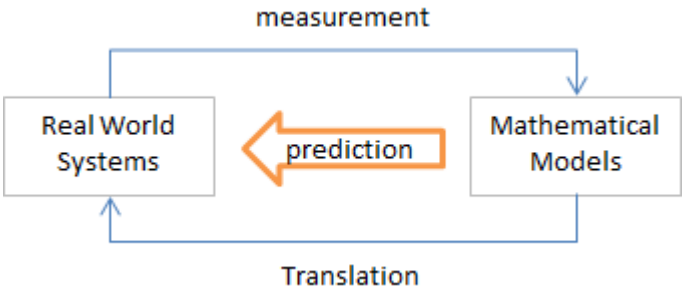


Figure 2.23: The application of methematical models into real world systems. As long as the real systems are measurable in some ways, the methematical models can be utilized to improve the performance of them.

2.4.3.3 Linear regression

The linear regression analysis (D. Kleinbaum, L. Kupper, K. Muller, & A. Nizam, 1998; Neter, Kutner, Nachtsheim, & Wasserman, 1996; Principe, Euliano, & Lefebvre, 2000) employed in this thesis's proposals adopts the following simple equation:

$$y = wx + b.$$

This equation can be converted to a diagrammatic representation, as shown in figure 2.24. This diagram is called the linear processing element. It is made up from 2 multipliers and 1 adder. The multiplier w scales the input, while b is the bias.

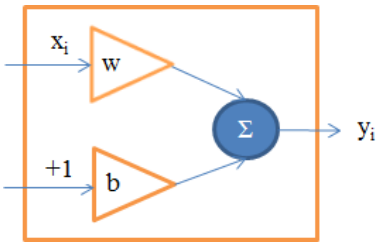


Figure 2.24: Linear Processing element. The interest by using this diagram is to solve the problem with linear relationship between the input x_i and y_i. Adapted from (Principe et al., 2000)

As mentioned in section 2.4.3.2, the collected data must be free from noises. To categorize the data as legitimate points or outliers, a variable called residual is defined, as $\varepsilon = y_i - \tilde{y}_i$.

To find which line is the most ideal line that best fit the collected set of data, a criterion is defined, called *mean square error (MSE), J*. In this case *J* is the squared average sum of all the residuals, as

$$J = \frac{1}{2N}\sum_{i=1}^{N} \varepsilon_i^2,$$

where *N* is the total number of data points.

It is clear that to find the best fitted regression line, the objective is to minimize the above equation. Gaussian models(Wiesel, Eldar, & Yeredor, 2008) derived that the best fitted regression line will have the values of the gradient, *w* and y-intercept, *b* calculated as follows:

$$w=\frac{N\sum_{i=1}^{N} x_i y_i-(\sum_{i=1}^{N} y_i)(\sum_{i=1}^{N} x_i)}{N\sum_{i=1}^{N} y_i^2-(\sum_{i=1}^{N} y_i)^2},$$

$$b=\frac{(\sum_{i=1}^{N} x_i)(\sum_{i=1}^{N} y_i^2)-(\sum_{i=1}^{N} y_i)(\sum_{i=1}^{N} y_i x_i)}{N\sum_{i=1}^{N} y_i^2-(\sum_{i=1}^{N} y_i)^2}.$$

The MSE is useful to determine the line that best fit the collected data. However it does not show how good the best fitted line in representing the set of data. To address this, a parameter called correlation coefficient, *r* is defined, where

$$r=\frac{\sum_{i=1}^{N} y_i x_i-\frac{1}{N}(\sum_{i=1}^{N} y_i)(\sum_{i=1}^{N} x_i)}{\sqrt{\left[\sum_{i=1}^{N} y_i^2-\frac{1}{N}(\sum_{i=1}^{N} y_i)^2\right]\left[\sum_{i=1}^{N} x_i^2-\frac{1}{N}(\sum_{i=1}^{N} x_i)^2\right]}}.$$

r has a value between -1 and 1. When *r*=1, all the data points are perfectly fitted on the best fitted regression line. In this case, when x_i increases, y_i too increases by the same magnitude. The same happens when *r*=-1, however when x_i increases by a magnitude, y_i decreases by the same magnitude. When *r*=0, x_i and y_i do not have relationship with each other.

2.4.3.4 Machine learning

Machine learning (Mohri, Rostamizadeh, & Talwalkar, 2012) is a branch of Artificial Intelligence (AI), that deals with model or prototype construction that has the interest to understand and learn from the collected data. In the resource *monitoring* and *optimization* proposals, the data is collected to understand the database and VM behavior. Subsequently with this information, the resource and system states are learned to serve 2 purposes. First, the learning mechanism translates the learning into action for resource scaling purpose. Second, the learned information is converted into input for hardware fault and failure analysis. There are a great variety of machine learning algorithms. However the particular algorithm of interest in this thesis is the regression analysis. Regression analysis is a statistical method to estimate the characteristic of the relationship among parameters. In this case, the concern is about the discovery of relation between the VM and the database transactions.

The proposal in the resource utilization *monitoring* area utilizes the semi-supervised machine learning technique. This method of machine learning consists of labeled and unlabeled data. In most literatures, the unlabeled data is deemed as easily to collect and inexpensive, whereas the labeled data is perceived as scarce and expensive in computational term. In the domain of semi-supervised learning algorithms, there are many choices of learning algorithms. The relevant learning technique in this thesis is the Self-Training algorithm. At high level, it consists of 4 steps (Zhu, 2007):

1) Train f from labeled data, $(\boldsymbol{X_i}, \boldsymbol{Y_i})$.
    - This is equivalent to conduct training on the TPC-H queries against the TPC-H data in this thesis.
2) Predict on $x \in \boldsymbol{X_u}$.

- This can be translated to discovering the values of *w* and *b*, from the linear regression model in section 2.4.3.3.

3) Add (*x*, *f(x)*) to labeled data.

   - This step can be deciphered as repeating the step #1 and #2 for different set of labeled data, to obtained different values of *w* and *b* for different set of labeled data. In this sense, the different set of labeled data can be different queries in TPC-H benchmark, for instance, 1 set of w & b values for query 8, then another set of result for query #21.

4) Repeat step 1 to 3 to achieve accuracy.

The obtained array of *w & b* values is stored as baselines. Subsequently the unlabeled data is trained, and the result is matched to this array. Depending on the degree of conformance between the unlabeled data and the baselines, the deviation is learned continuously to determine the VM resource adequacy and hardware state.

There is a shortcoming in self-training algorithm, where the initial mistake in training the labeled data can lead to subsequent inaccurate comparison between the unlabeled data and the baselines. It is also to note that machine learning is a branch of knowledge in AI that constantly evolves (Zhu, 2008). The studied knowledge in this discipline is expected to be extensible and flexibly applied in various applications.

2.4.3.5 Fuzzy computing

In the area of resource utilization management, it is inevitable in some cases to depict the condition in the VM using arbitrary language. In such cases, *fuzzy logic* (Alavala, 2008; Ganesh, 2008) has been employed to handle situation of partial truth, where the truth value can reside between the range of completely truth and completely false values. Professor Lotfi Zadeh proposed the linguistic fuzzy concept (Zadeh, 1996) in his

paper titled “Fuzzy Computing with Words”. Since then the *fuzzy computing* has been widely deployed into a wide range of industries. To illustrate this concept, figure 2.25 is illustrated which takes the topic in this thesis as model. The ideal resource condition is the state of resources in the VM where all SLA-bound transactions are adhering to the required QoS. Such condition is inputted to the scaling mechanism, as reference to determine the scaling need. The scaling algorithm is responsible to direct the scaling decision based on input from the resource condition, which in this case is referenced to the resource adequacy scale in figure 2.26.

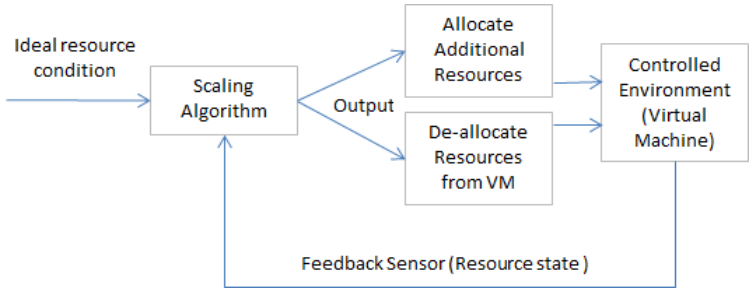


Figure 2.25: A Fuzzy logic control system for resource utilization monitoring. Adapted and modified from (Kaehler, 2005)

Resource utilization in a VM can be characterized as having subranges of a continuous parameters. Figure 2.26 depicts such condition. The blue, orange and red lines each depicts separate membership functions that represent the ranges of utilization intensity. In order to use this in the scaling algorithm in figure 2.25, each function will map the same utilization value to truth value of between 0 and 1. A combination of 3 values will control the decision to either scaling the resource up or down, or it remains unchanged. For instance, resource utilization condition in the VM is denoted by the straight line in figure 2.26. The red arrow in this figure shows a truth value of 0 for high utilization. The orange arrow is pointing to a value of ~0.2, which in fuzzy computing term, can be classified as slightly moderate utilization. Then the blue arrow denotes fairly low

utilization in the VM at value of ~0.8. The scaling algorithm makes decision based on these values in this fuzzy logic control system.

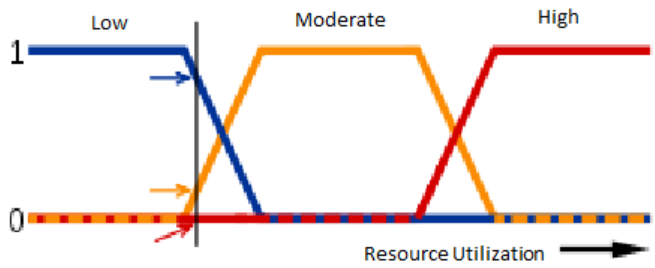


Figure 2.26: 3 membership functions in the resource utilization scale. These membership functions serve the purpose as input to the fuzzy logic control system.

2.4.3.6 Linear Programming and Simplex Method

Linear programming is a mathematical formulation that searches for the best outcome of a problem, given a mathematical model which has a linear relationship among the variables. To find the most optimized solution for this mathematical problem, it is subjected to linear equality as well as linear inequality constraining factors. In the simplest and most significant form without compromising its principle, the linear programming problem can be written as:

To optimize (maximize or minimize)

$$\sum_{i \in [n]} x_i(f_i),$$

Subject to

$$\sum_{i \in [n]} a_i(f_i) \leq b_k,$$
$$f_i \geq 0.$$

When translating the above to the database transactions, n is the variety of SQL in the workload. f_i denotes the frequency of each SQL, and x_i represents individual run time of each SQL. Value of a_i depicts the database parameter limits. These can be the memory reads parameter, where the total memory reads, b_k of the SQL cannot exceed certain threshold for the optimization problem to be solved. It can also include physical reads

parameter, if the SQLs are doing physical I/O intensive operations. Following illustrates the manual steps in utilizing *linear programming* technique.

For instance, there is a problem to maximize following equation to discover the ideal combination of TPH-C queries to synthesize a stress-testing scenario in the VM (C. H. Tan & Teh, 2013b):

$$Q = x_1 f_1 + x_2 f_2.$$

Subject to

$$0 \leq P_1 f_1 + P_2 f_2 \leq P,$$

$$0 \leq C_1 f_1 + C_2 f_2 \leq C,$$

$f_1 \geq 0$, $f_2 \geq 0$, because run frequency cannot be negative,

Where,

- P_i is the individual Physical Gets (PG) of SQL, S_i. PG is equivalent to physical reads in the database.
- P is the total *PG*, which matches the *PG* in the steady state during initial conventional load testing. This value is to serve as baseline.
- C_i is the individual Consistent Gets (CG) of SQL, S_i. CG is equivalent to memory reads in the database.
- C is the total *CG*, which matches the *CG* in the steady state during initial conventional load testing. This value is to serve as baseline.

The constraints need to be converted to *slack form*, in order to be solved. So,

$$P_1 f_1 + P_2 f_2 + s = P,$$

$$C_1 f_1 + C_2 f_2 + t = C,$$

Where, *s* and *t* are slack variables.

With the aforementioned, the problem can now be solved by Simplex method (Sinha, 2006). To illustrate this, the variables' are assumed to have following values:

$x_1 = 10s, x_2 = 15s, P_1 = 400, P_2 = 300, P=5000, C_1 = 400, C_2 = 600, C=12500$

In real practice, values of x_1, x_2, P_1, P_2, C_1, C_2 can be obtained easily by running the individual SQL in the database. *P* and *C* are the values that match the total *PG* and *CG* during initial load testing when the application went live, which corresponds to initial C_h. Now,

$Q - 10f_1 - 15f_2 = 0,$

$400\ f_1 + 300\ f_2 + s = 5000,$

$400\ f_1 + 600\ f_2 + t = 12500.$

These data are then put into tableau format, as in Table 2.3. The italic section shows the data processed by *Simplex* method. This method is a popular algorithm to solve problem for *linear programming*. The discovery of this mathematical technique is very significant, to the extent that it is ranked in one of the top 10 algorithms in 20th century (Dongarra & Sullivan, 2000).

Table 2.3: Tableau depicts the Simplex algorithm. The value in red font shows the pivot.

Q	f_1	f_2	*s*	*t*	*values*
1	-10	-15	0	0	0
0	400	**300**	1	0	5000
0	400	600	0	1	12500
1	*10*	*0*	*1/20*	*0*	*250*
0	*4/3*	*1*	*1/300*	*0*	*50/3*
0	*-400*	*0*	*-2*	*1*	*2500*

With this result, the *objective equation* becomes

$$Q + 10f_1 + S/20 = 250.$$

Hence the optimized solution is Q=250 as the rule requires the variables in the objective function to be 0. With this value, the optimized values for f_1 *and* f_2 are obtained. So now,

$$10f_1 + 15f_2 = 250.$$

If f_1=10, f_2 is then 10. The frequency ratio to run the combination of mixed workload S_1 and S_2 in the new hardware configuration is 1:1. With this ratio, the *VM* is loaded with the 2 SQL to reach the new induced C_h. When the *VM* is stabilized at this level, the SLA-bound transactions are executed for validation purpose to serve the objective of meeting the SLA requirement.

If iterative calculation is needed, the simplex method can be accomplished conveniently by the Matlab software (Hueeber, 2011). Such usage is common in cases where continuous optimization is needed, for instance in (Vandenberghe, Boyd, &

Nouralishahi, 2002), minimization of distance measurement is envisaged to partition the workload iteratively, and such software is handy in such situation.

2.4.3.7 TPC benchmark

The 2 most widely referred TPC benchmarks are TPC-C (TPC, 2013a) and TPC-H (TPC, 2012). TPC-C is an online transaction processing (OLTP) benchmark, whereas TPC-H benchmark is of decision-support type. Both benchmarks are meant to provide a foundation of measurement for hardware and software vendors to showcase their products' capability. In the research world, they are frequently utilized as standard queries and data to generate output that is applicable for analysis to produce new theorems and algorithms. TPC-C is most suitably used to measure the capability of hardware or software, where it measures the number of orders that can be processed in a minute. In computing term, the parameter is called tpm-C, and this parameter is commonly understood by the wide industry. The design of TPC-C is not meant for performance analysis as the triggered transactions are generally not consuming much system resources.

On the other hand, TPC-H is regularly utilized in stress tests. All the 22 queries in this benchmark are capable of stressing the hosts to their limits. Such ability is harnessed for the resource utilization *affirmation* experiments in this thesis, as this benchmark is employed to create stress-testing scenario in the VM. Furthermore, due to its design that allows for all transaction processing system, regardless of hardware type or operating systems, it has been deployed in the resource utilization *monitoring* and *optimization* segments in this thesis, as the proposed algorithms can be easily proliferated to other platforms.

This thesis intensively makes use of TPC-H queries and data in the experiments. For future research, it is interesting to exploit TPC-C transactions to represent some

functional values in the proposals. With this potential enrichment, more robust resource management algorithms can be constructed.

2.4.4 Measurement methods

In large part of the studies in this thesis, statistical models play a major contribution towards achieving the objectives of resource management, particularly in utilization *monitoring* and *planning* matters. With the increase in demand and complexity of the systems through virtualization, heterogeneous and distributed components become common elements that articulate the hosting architecture. With this development, many heuristic and rigid rule-based approaches towards system management for better performance and failure prediction are rendered ineffective. Because of the multiplicity of system configuration, benchmarks and standards definition for reference is at its nascent stage as sufficient maturity in the industry for this distributed computing technology has not been achieved. Chen et al. (2011) explained the importance in merging both statistical analysis and system design. The challenges are described as identifying the statistical techniques for specific systems, how to evaluate the statistical-driven optimizations and apprehend the statistical output for ease of human interpretation. The characterization of the workloads is critical for better understanding of how system configuration can be consummated and system performance can be improved. With the statistical characterization of the system and workload established, these models are evaluated to demonstrate their quality, as well as their significance in system performance. The evaluations are performed via hypothetical workloads and real workloads testing, with periodic re-training (Ganapathi et al., 2009) and testing, so that they can keep pace with the evolving systems and workloads. Another important element in evaluating either the workload of the system, is to enable an effective monitoring mechanism. As replaying the full workload in the system is not viable for mission-critical applications, a robust monitoring mechanism ensures that all aspects in

the critical workloads are controlled, with performance remediation or improvement set in perspective.

In managing the data system, particular in resource management topics, integration of effort from IT system designers and statisticians are useful. Traditionally, the IT architects are responsible to design appropriate system to serve different varieties of applications, while the statisticians are mainly focusing in effort to produce effective algorithms or models for the use of the wide industries. It will be more rewarding if the knowledge from both fields are assimilated, as explained by Chen et al. (2011). Such combination of knowledge is also demonstrated by the proposals in this thesis.

Designing statistical models that are representative for particular IT environments is very time-consuming and error-prone, as the characterization works are unprecedented. Hence the ideal scenarios for workload modeling appear when there are solid academic or industrial benchmarks for the particular workloads. However, at the point of this writing, these benchmarks are not widely available. Chen (May 2012) detailed into benchmarking effort currently available, and concluded that the time is yet to come for "the big data benchmark". While the industrial players and scholars are working on designing robust benchmarks for Big Data, at the moment, a more accurate way for performance evaluations is using realistic workloads as the input in *Hadoop Mapreduce* systems. This is illustrated by the subsequent research works by Chen et al. (March 2011). The authors analyzed and discovered that the current Big Data benchmarks do not provide a one-size-fit-all solution for *MapReduce* performance evaluation. The currently available benchmarks: Gridmix (Iosup et al., 2008), Hive Benchmark (Jia & Shao, July 2009), Pigmix (Apache, August 2011) and Hibench (S. Huang, Huang, Dai, Xie, & Huang, March 2010) are observed to fit certain type of workloads. However as *MapReduce* system is going through enormous challenges in increasing growth, diversity, computational volume and consolidation of the data and computing resource,

more dedicated benchmarks are required. Hence, the authors provide insights on how intricate workloads can be apprehended. The objective is to enable better cluster provisioning and management. The *MapReduce* framework is not the focus in this thesis; however the works conducted in this statistical modeling and benchmarking area are applicable to the *Parallel Database* arena. Some of the prominent statistical methods employed to characterize the workloads are k-means (Elkan, 2003), Hierarchical clustering (Manning, Raghavan, & Schütze, 2009), maximum likelihood estimation (MLE) (Myung, 2003) and Goodness of fit (GOF) (Narsky, 2003). To provide input to these clustering algorithms, there are 2 commonly deployed distance measurement techniques. There are the Euclidian distance (Weisstein, 2013) and Cosine distance (P. N. Tan, Steinbach, & Kumar, 2006). They are explained as follows:

For 2 real n-vectors, x = (x_1, x_2,..., x_n) and y = (y_1, y_2,..., y_n), if they are 2 points in Euclidean n-space, the distance between these 2 points is defined as

Euclidian distance, $d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$.

Cosine distance, $d(x,y) = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n}(x_i)^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i)^2}}$.

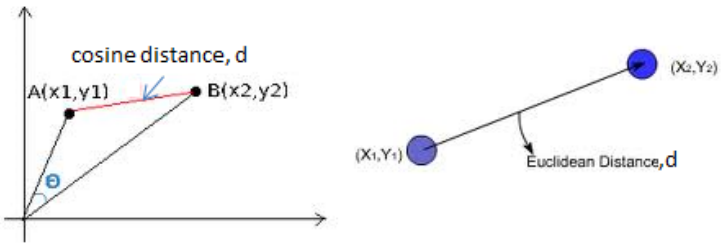


Figure 2.26: Image illustration of the 2 distance measurement method.

From workload perspective, assume the SQL processing time with logical reads as 2 parameters, they can be translated to vector form to take advantage of such measurement for clustering purpose. The various clustering methods are explained in following.

2.4.4.1 Hierarchical clustering

If given a set of *N* items to be clustered, with the distance between each item computed, the hierarchical clustering is accomplished via following 4 steps:

1) Group each item into its own cluster.
2) Find the closest or most similar pair of clusters, and group them into another cluster.
3) Compute the distance between these clusters. There are few ways to calculate the distance between the items. The commonly utilized methods are:
    - Complete linkage clustering, where the maximum distance between items in a cluster is considered.
    - Single-linkage clustering, where the minimum distance between items in a cluster is considered.
    - Average linkage clustering. The average distance between all items in a cluster is computed.
4) Repeat step 2 and 3, with the eventual aim of a single cluster.

Such algorithms may be useful when there are multiple sets of workloads, which need to be grouped in their similarities based on multiple criteria to deliver certain purposes, for example, a group of SQL can be grouped by their processing time, then subsequently by their logical reads, and then physical reads etc.

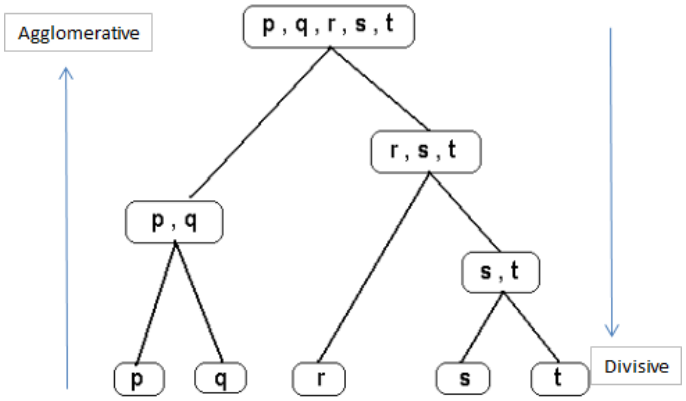


Figure 2.27: Hierarchical clustering. The distance between elements in a cluster determine the clustering result. The agglometative activity increases the distance between clusters.

2.4.4.2 K-mean Clustering

K-mean clustering aims to cluster a set of *N* items into *k* number of clusters, where each cluster contains the items with mean distance within a stipulated limit. This clustering method is considered as *NP-Hard*; hence heuristic algorithms are employed to compute these k-clusters. Mathematically, the goal is to minimize an objective function, *J* as

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^j - c_j \right\|^2,$$

where $\left\| x_i^j - c_j \right\|^2$ is a distance measure between an item, x_1 and the cluster's center position. The end result of the clustering algorithm is depicted in figure 2.28.

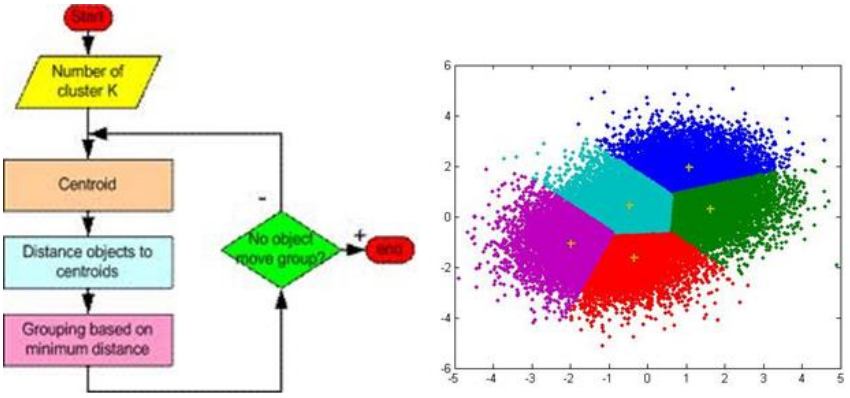


Figure 2.28: K-Means Clustering. The left workflow depicts the steps to arrive at the end clusters. The right exhibits the result from such clustering algorithm. Adapted from (Samplify, 2013)

In workload aspect, take SQL processing and memory reads as the parameters to be clustered, this K-Means Clustering method can be used to segregate between OLAP or OLTP transactions, to satisfy certain functions in workload characterization.

2.4.4.3 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) takes the result from a smaller set of data, to estimate the likelihood of the same occurrence in a larger set of data. Suppose there are *N* sets of measurements, (x_1, y_1), (x_2, y_2), …, (x_N, y_N), and the relationship between *x* and *y* is known, $y = q(x, \alpha, \beta, \ldots)$, where $\alpha, \beta, \ldots$ are parameters. If the *y* is related to *x* linearly, the equation can be written as $y = \beta x + \alpha$. By employing MLE, the maximum likelihood function can be obtained, by calculating α and β as:

$$\alpha = \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} y_i x_i \sum_{i=1}^{n} x_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2},$$

$$\beta = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}.$$

With these 2 values, for particular *x,* the most probable value of *y* is determined. Such method may be useful to predict the processing time of particular SQL in a workload. Such prediction can be useful for transaction clustering in particular groups for scheduling purpose.

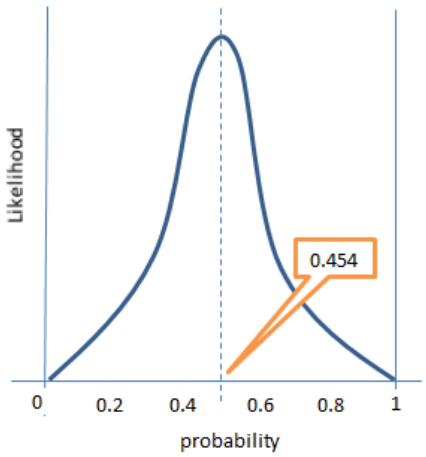


Figure 2.29: Maximum Likelihood Estimate. The diagram depicts a coin toss test that reveals the highest probability of 0.454 for the likelihood of heads, in 11 attempts, with 5 heads and 6 tails as the result of the test, using particular coin. This is a simple example where *n* number in occurance of heads can be calculated in *m* samples. In more complex cases, the MLE algorithm strives to achieve the same objective of meeting the highest probability for a particular parameter in a particular set of test.

2.4.4.4 Goodness of Fit

Another workload prediction technique is called Goodness of Fit. The Goodness of Fit (GoF) for a statistical model measures how well this model fits into a set of real observed data. The measurement output will show the discrepancy between the benchmarked model and the real data. The chi-squared test statistic can be calculated by

$$x^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i},$$

where,

O_i is the real observed values, and E_i denotes the expected value. For a linear regression model, if the value of x^2 is getting closer to 1, the real set of observed values are deemed more fitting to the hypothetical or expected model. This is displayed in 2.30.

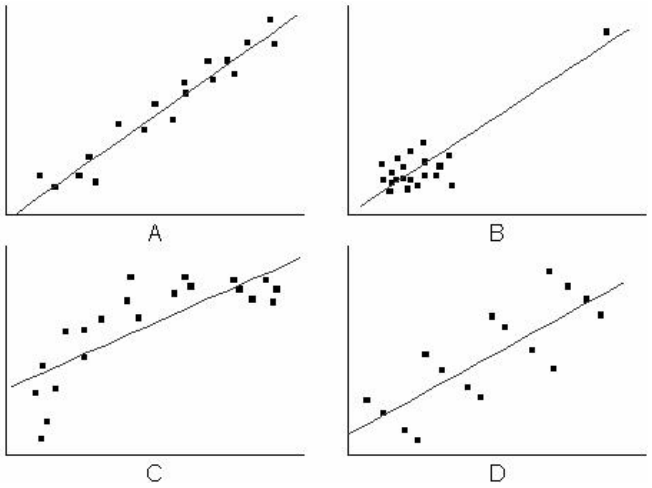


Figure 2.30: The chi-squared test of Goodness of Fit for linear regression. The value of x^2 is considerably high for A, B and D. However in the case of B and D, they do not clearly represent the condition of expected and real data. In these cases, additional analysis is needed, potentially via residual analysis.

Such prediction can be deployed to predict the stability in the computing resources, if the workloads in the VM are processed within expected consistency and accuracy. In such cases, if the value of x^2 is high, the hardware performance is deemed consistent and optimal. Such observation is part of the proposal in this thesis.

In addition to the above clustering methods, Kernel Canonical Correlation Analysis (KCCA) has also been used to characterize a workload. Ganapathi et al. (2009) (2010) proposed a method to predict the workload performance, by employing the KCCA method (Bach & Jordan, 2002). This method strives to find the maximum correlation between 2 vectors. The first step in their proposal is dealing with converting the each query in the synthetic workload from TPC-DS benchmark (TPC, 2013b) into a vector, where this vector contains the "query features", for instance the type of joints in the SQL, whether they are of hash, nested loop, inner or outer join. Each baseline query will have a set of query features that are almost unique for the query. The second step is to create vector that contains "performance features" for each query. The chosen performance metrics are disk I/O, SQL processing time, number of returned rows, number of bytes in the returned rows, the scanned tables' records and the used tables' records. Assume there are *N* training queries involved, so there are *N* pairs of query and

performance vectors. The "distance metric" between these *N* pairs of vectors is computed. With this information, KCCA is employed to find associated pairs of clusters in the performance vector and query vector space. The query plan and performance projections are generated as the result, as in figure 2.31.

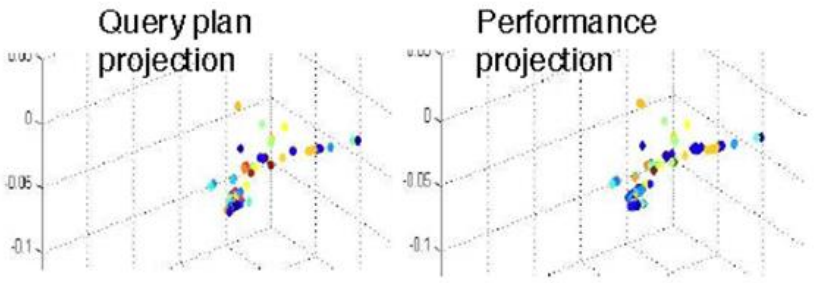


Figure 2.31: Query plan and performance projections as the result of KCCA computation on the 'distance metrics' of *N* training queries. Adapted from (Ganapathi et al., 2009)

With these 2 projections, query performance prediction can be conducted. First the query feature vector of the new query is computed; then its coordinate in the query plan projection is located. The same coordinate is subsequently matched to the performance projection, to find the performance characteristics of the new query. The authors' proposal is different from the query performance prediction from *query optimizer*, in the sense that the commercial *query optimizer* predicts the query performance which then generates the perceived most optimized execution path, by mean of statistical performance data which does not involve training the queries in prior. The technique utilized by the *query optimizer* has its advantage, as the application transactions in the user database does not need to be halted to make way for *machine learning* to train the training queries. The *query optimizer* can continuously '*learn*' the parameters and condition in the database online without incurring much system overhead. However, the proposals in this thesis are employing the same methodology as in (Ganapathi et al., 2010; Ganapathi et al., 2009), by relying on the benchmarked data as it is closer to the actual desired values. The general probing of the database states by the *optimizer* is

sometimes deemed not comprehensive enough in certain aspects in providing input to the control or prediction systems.

2.4.5 Workload characterization

As mentioned, the above clustering methods are meant to improve the understanding of particular workloads. With this understanding, many improvements at the host level can be accomplished with this new capability. Chen et al. (March 2011) (May 2012) studied the workload characterization in *MapReduce* framework, and envisaged that effective workload characterization can provides following benefits, which are also applicable for *Parallel Database* architecture:

- The growth anticipation of workloads in respect to amount of data and transactions volume can be predicted more accurately.
- The computing resource requirements can be precisely estimated.
- Resource provisioning activity can be performed accurately for specific workload types, based on the benchmarked workloads.
- The effect of clustering multiple workloads into same groups can be forecasted.
- The superposition of clustering multiple workloads into the same cluster can be visualized.
- The clustered workloads can be baselined and extended to future workloads, to serve the above functions.

Furthermore, in order for the threshold-based or Markov Chain-based scaling algorithm in section 2.4.1 to work in real environments, it is imperative that the workloads are performing in most efficient manner. The effort to tune the workloads is the pre-requisite to resource management in the virtual hosts. The authors provided insights on how intricate workloads can be apprehended using live data and transactions. The

objective is to enable better cluster provisioning and management. Motivated by similar approaches, In (C. H. Tan & Teh, 2013a), a statistical model is proposed from the aggregation of metadata from real database workloads and operating system variables, to deliver for the *resource planning* purpose, as this collection of data from the database itself rightly described the real users' experience.

To ensure efficiency in workload processing, Mateen et al. (2011) proposed autonomic workload management that encompasses self-optimization, self-configuration, self-inspection, self-prediction, self-organization and self-adoption. Self-optimization is the characteristic in the database, where minimum resources are utilized to process a group of transactions in an organized way. For instance, the *query optimizer* in Oracle RDBMS inspects and parses the SQL statements, and discovers the most optimized execution path to produce the output for the statements. Self-configuration is the responsibility to arrange for the best possible condition in the database, so that the database engine can have a more conducive environment to process the transactions efficiently. For example, the SQL Tuning Advisor (Yagoub & Gongloor, 2007) tool offered by Oracle is able to propose needed indexes and SQL profiles, so that the *query optimizer* can have a greater variety of choices to produce the most efficient execution paths. Self-inspection is the capability where the database engine is capable of discovering and visualizing abnormalities in the database environment. Oracle RDBMS provides such utility called Automatic Database Diagnostic Monitor (ADDM), where the data characteristic is examined periodically to spot the need for additional memory allocation, objects' statistic update, indexes rebuild, details of executed queries etc. Self-prediction anticipates the amount of resources and time needed to complete the workload processing. The Oracle *query optimizer* is equipped with such capability, where the cost and duration of the SQL processing can be estimated prior to the real executions. Self-organization is another great feature, where the database is able to

position itself in the most optimized condition for SQL processing. For instance, the Oracle Automatic Storage Management (ASM) utility can self-collect the statistic information on the tables and indexes, or rearrange the data blocks on different disks to alleviate I/O contention. Self-adoption is the characteristic where the database is able to take full advantage of the condition in the host, to benefit the workload processing. For instance, if the database detects that there are additional *x* number of virtual processors provisioned to the host, it should be able to increase its processing parallelism *x* amount of times. In many RDBMS system, the aggressive pattern of memory consumption in the database is very common. However with proper adaptation strategy, this memory exhaustive attribute should allow for co-location of other important operations in the host, for example operating system security auditing, monitoring of overrun background processes, backup and recovery operations etc.

In a large part of this research as well as in the literature studies, the monitoring parameters are often mainly comprised of the CPU run queue length or memory I/O usage. However, in data intensive operations, it is critical to take the disk I/O into consideration, as there will always be limited memory resource to contain the whole data sets needed for workload processing. Particularly in cloud, this parameter can affect the transactions' performance by a few orders of magnitude. The issue here is the data locality. In standalone server, local disks are normally associated with the server, hence the computing resource does not need to travel far to fetch or write to the disks. Nevertheless, as cloud promotes distributed computing architecture, data can be relative far from the computing resource. Particularly in Public Cloud, this phenomenon is not favorable for data intensive application, for instance data mining in Data Warehouse. For such data analysis work, it will be more logical to host the database in Private Cloud where data co-locates with the other computing components. However, there are researchers who study and remediate the issue on this data locality problem in the case

where data hosting in large data center is unavoidable, similar to the Public Cloud model. Raicu et al. (2008) developed a resource provisioning and task dissemination system called Falkon, mainly serving data intensive processing in NASA. The Falkon architecture is depicted in figure 2.32. The task dispatcher delivers the tasks to be processed by the executors, which in this case are the computing nodes that contain the data, via pre-computed algorithms. These algorithms retrieve their input from a set of indexes that hold the information of tasks and location of data for the particular tasks, much like the key-value pairs in the states and transition probabilities of Markov Chain diagram. The dispatcher delivers the task to the node that contains the most data to be processed by the task.

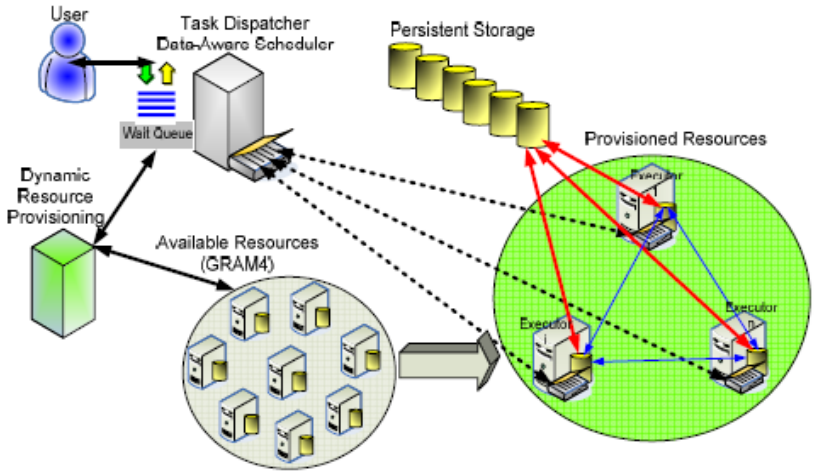


Figure 2.32: Data Diffusion model. Task is dispatched to the node (executor) where majority of the data resides. Adapted from (Raicu et al., 2008)

The interesting fact produced by these authors in their experiments, is the relationship between the throughput and the number of nodes, with the I/O size remains as constant. As demonstrated in figure 2.33, in distributed computing, the throughput is saturated at certain number of nodes, and subsequent increment in horizontal scaling does not improve the capability of the cluster. In another experiment, it is discovered that the bigger the single I/O stream, the higher the throughput of the system. This can be attributed to the fact that each I/O incurs overhead which becomes significant when

many smaller I/O streams are running for the distributed cluster. This is depicted in figure 2.34.

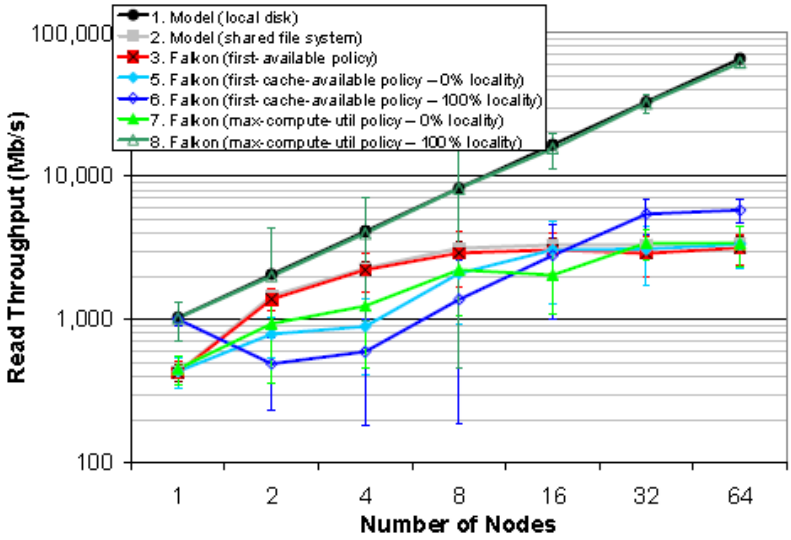


Figure 2.33: Throughput vs # of nodes in a cluster. Local disks reads capability increases in tandem with the horizontal scaling. However, distributed shared demonstrates a much degraded throughput comparatively, including the limitation of hitting throughput constraint at certain number of nodes in the same cluster. Adapted from (Raicu et al., 2008)

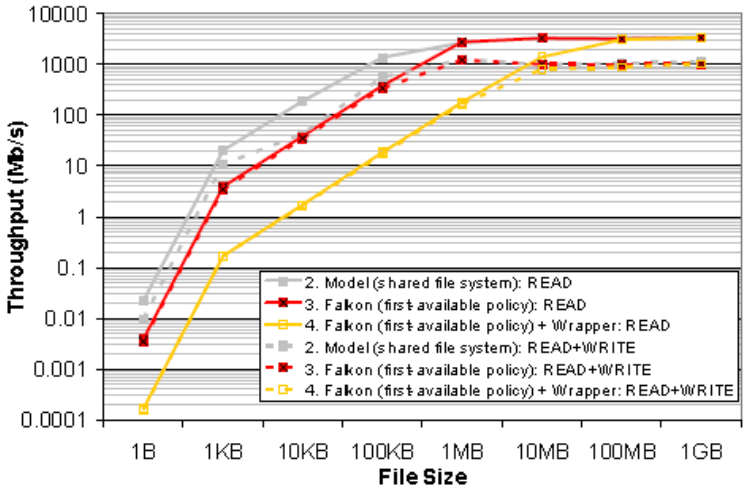


Figure 2.34: Throughput vs file I/O size. Each I/O incurs overhead which becomes significant when large number of transactions is carried out in the distributed computing cluster. Adapted from (Raicu et al., 2008)

The above findings conclude that in distributed computing, such as cloud, the number of nodes in a cluster should be controlled. Vertical scaling should be considered instead of unnecessarily spanning the resource horizontally. Regardless if the nodes are hosting the

data or not, the throughput of distributed transactions will always be much lower than the throughput in single server, as experimented by Curino et al. (2010). In the case for the nodes that do not host the data, the latency in response time is caused by tables or rows contention as multiple nodes are trying to access the same data, distributed deadlocks and complex SQLs that need to draw the computing power from multiple servers. To alleviate one of these drawbacks which relates to the issue with I/O overhead in the distributed computing, Curino et al. (2010) proposed an algorithm to partition and replicate the database so that workloads can reside in minimum number of nodes to reduce the overhead of distributed transactions, at the same time balance the workload across all available nodes. The rule of thumb to replicate a partition is that if the tables in the partition are not updated frequently, then the data in the tables are replicated to multiple nodes to take advantage of the additional computing resource, by taking the distributed transactions overhead into consideration. Depending on the memory size of the nodes, the partitions' size is chosen to fit as much as possible into the memory. The partitions are cut equally based on 2 criteria: their data size or access frequency on the data. At high level, the workload traces are collected, and subsequently the SQL and tables involved in the workloads are mined and analyzed. Based on the analysis on the *where* clause and access frequency, the data in single or joint tables is partitioned as shown in figure 2.35.

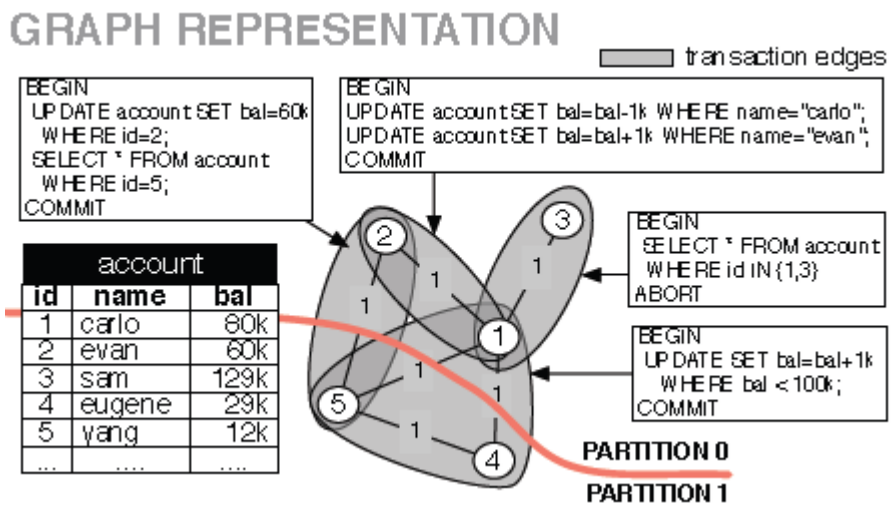


Figure 2.35: The graph representation of partitioning activity. The nodes in the diagram are the tables, while the greyed edges denotes the transactions, with the weightage implies the frequency of the transactions. Adapted from (Curino et al., 2010)

Even though the authors never envisaged cloud hosting in their proposal, their suggested partitioning and replicating algorithms are very useful for hosting of large databases with intensive I/O in cloud, particularly in Public Cloud. Furthermore, many *Parallel Database* architectures already have the partitioning option built-in; hence it is not necessary to reinventing the partitioning technology. It is believe that these replication and partitioning mechanisms will become popular in tandem with Public Cloud hosting for database operations.

Table 2.4: Summary of studied researches with critical comment on sub-themes 'statistical modeling and workload characterization'.

Scholars	contribution	comment
(Y. P. Chen et al., 2011)	Explained the importance in merging both statistical analysis and system design.	As replaying the full workload in the system is not viable for mission-critical applications, a robust monitoring mechanism ensures that all aspects in the critical workloads are controlled, with performance remediation or improvement set in perspective.
(Y. P. Chen, May 2012)	Detailed into benchmarking effort currently available.	A more accurate way for performance evaluations is using realistic workloads as the input.

(Mateen et al., 2011)	Proposed autonomic workload management that encompasses self-optimization, self-configuration, self-inspection, self-prediction, self-organization and self-adoption.	Workload processing involves aggressive pattern of memory consumption. The autonomic workload characterization should allow for co-location of other important non-database operations in the host.
(Raicu et al., 2008)	Developed a resource provisioning and task dissemination system.	The paper discovered that the number of nodes in the distributed computing should be controlled. Vertical scaling should be considered instead of unnecessarily spanning the resource horizontally.
(Curino et al., 2010)	Proposed an algorithm to partition and replicate the database so that workloads can reside in minimum number of nodes to reduce the overhead of distributed transactions.	Even though the authors never envisaged cloud hosting in their proposal, their suggested partitioning and replicating algorithms are very useful for hosting of large databases with intensive I/O in cloud, particularly in Public Cloud.

2.5 Resource utilization *optimization*

In the topic of optimization in virtualized environments, Beloglazov et al. (2012) outlined 5 challenges which need to be overcome in order for the efficiency of resource utilization in cloud computing to achieve the next milestone. They are:

1) Optimization of *VM Placement*. This is also called VM migration in some literatures. It is the eventual outcome of resource scheduling algorithm, where the content in a VM is migrated to another VM in order to take advantage of larger and smaller sized hardware. The former will benefit the quest for additional resources to process heavier and more complex workloads. The latter is beneficial in the case where idle resources can be sent back to the resource pool, hence achieving saving in operational cost and reducing carbon footprints. The frequency of *VM Placement* should be reduced as much as possible to avoid hindrance to normal users' operations and lower the overhead of the migration activities. Hence, the research challenges here are:

    - How to predict peak usage of the application.

- When and where to migrate the VM to, considering heterogeneity of the virtualized hardware.
- How to reduce the VM migration duration across large-scale systems.

2) Optimization on the virtualized network. In cloud computing, VM and hardware clusters are connected via a huge network topology. The storage component for a database is usually segregated from the computing nodes. Nevertheless, both the data and computing elements are tightly coupled in order to complete the database transactions. When the VM migration takes place, the distance between the computing and data nodes might differ from the original configuration, resulting in the changes in communication latency between these 2 components. The response time from particular transactions can greatly differ due to this factor. Hence the challenge here is to ensure that the resource scheduling algorithm can guarantee similar or shorter distance of network links between the SAN storage and the computing nodes.

3) Optimization on the thermal states and cooling system in the cloud data center. The purpose of optimization is to reduce the capital and operational costs. Overheating of computing hardware can greatly increase the electricity usage in the data center, hence affecting the bottom line of the business. Michael Bell, the research VP of Gartner Inc. quoted that *"Power and cooling is a pandemic in the world of the data center"*(Botelho, 2007). Such is the extent of how much the cost to control the heat in data center is hitting the revenue. To reduce this wastage, overheated hardware needs to be shut off. Hence, besides setting resource threshold to safeguard the VM performance, thermal management is needed, which requires the setting of temperature threshold coupled with robust monitoring in the underlying hardware. When the hardware is presumed overheated, the VM placement algorithm is

triggered. To decide this threshold is a challenge which is not studied in length in the literatures.

4) Optimization on workload consolidation. Multiple workloads should be consolidated to make full use of the available VM. For instance, if the CPU and memory utilization thresholds for a particular VM are set to 70%, it is normally difficult to have one workload that can consume all these resources. For example, multiple applications that serve the OLTP type of transactions can use up the CPU cycle, but they are not I/O intensive operations. In this case, data analytical transactions that consume a lot of I/O can be mixed into the VM so that the memory component can be utilized. The challenge here is to identify which workload to combine to which, in order to achieve the ideal resource usage optimization.

5) Achieving the equilibrium of conforming to SLA and maximizing resource utilization. A lot of the past literatures dealt with the task and resource scheduling algorithms that are focusing on achieving optimization in resource usage. However recent papers (Fito, Goiri, & Guitart, 2010; Iqbal et al., 2010) started to realize the importance of combining the SLA factor into these algorithms, as the business can only profit if their clients are satisfied with the service offerings. Hence the challenge is to discover the ideal point where both SLA conformance and saving in operational cost are meeting the expectation.

Some of the optimization methods, particularly task and resource scheduling are discussed in length in the earlier section. Apart from optimization in resource utilization, this section also details into fault analysis to discover future failure. This topic is important as optimization cannot be achieved with faulty hardware in the systems. In fact, all the 5 challenges outlined above need the hardware to perform to its optimal condition in order for subsequent feasible analysis and improvement to take place.

2.5.1 Fault analysis and failure prediction

Future prediction of workload and resource requirements has been discussed in previous section. In this section, the prediction on system state is of interest. This topic attempts to find out the condition in the VM, if the environment is conducive for critical transactions processing, by ensuring consistency and optimality on the hardware resource performance. The hardware failure is commonly a result from hardware wear out, deterioration in the hardware material, malfunction of the hardware components or any combination of these 3 culprits. Such occurrence in the VM will result in failure to achieve desired computing capacity to service the needed transactions. In cases where mission critical applications are required to be processed, such fault and failure in the hardware could result in the breach of SLA. Before further explanation is provided, the terms *fault* and *failure* need to be defined to ease the interpretation of system state. *Fault* is the hypothetical assumption that the system is going to fail, and it could remain dormant for a while before it causes the failure. While *Failure* is the event where the observed services in the VM do not meet the desired computing condition promised in the initial stage of hardware provisioning.

This section discusses the researches studied by scholars in discovering the fault and predicting failure occurrence, hence achieving minimum outages to the intended operations in the hosts, by ensuring remediation actions are taken as soon as possible. The fault analysis on the hardware condition usually involves following efforts:

1) To monitor the hardware performance periodically. Such monitoring involves capturing substantiated historical data of the system performance into a repository.
2) To understand the sources and logics that lead to the undesired event/state. In order to provide for such understanding, the undesired condition in the VM needs to be defined.

3) To create a baseline state in the VM, when the performance is at expected level. Such baseline will be compared to running condition during steady state operations.

4) Make reference on benchmarks established by scholars and industrial players, in order to understand and compare the performance standard of similar systems in the industry to the VM's performance.

The caveat to the effectiveness in the many proposals and solutions to fault analysis and failure prediction is that human error is not taken into account in composing such mechanisms. The following describes such mechanisms.

Salfner et al. (2010) conducted detailed survey on computer system failure prediction methods. They studied on *online failure prediction*, a term that describes the assessment of current system state during runtime of the computing hosts. Such prediction allows for decision to be made if there indeed is going to be a failure, based on short-term assessment, which is confined to within few minutes to an hour before actual failure. The quality and confidence level of prediction provide to the functions of the different categories of actions to be performed. For instance, less confidence prediction outcome may warrant just a system reboot, whereas a high confidence prediction may result in extensive system diagnostic or part replacement. The data sets are trained, validated and projected to estimate the useful information. In order for the proposed *online failure prediction* to produce high quality outputs, the time relations of before, current and future state of the system are planned and defined precisely. Figure 2.36 illustrates time relations for typical online prediction methods. t is the current time. Δt_d denotes the duration when the system is accessed. From this assessment, the prediction mechanisms anticipate that the system will have potential occurrence of a failure in Δt_l. Another parameter, Δt_w represents the minimal warning time before failure. The prediction is having an interval of Δt_p for it to be valid. To gauge the quality of failure prediction algorithms, 4 metrics are defined; they are true positive (TP), false positive (FP), false

negative (FN) and true negative (TN). TP happens when failure occurs within the prediction period, with warning raised in prior. FP denotes the situation where failure does not occur, but warning is raised. If the prediction system fails to notify on a true failure, FN flag is raised. In TN, no failure and no warning is given. These 4 metrics are used to determine the accuracy of the predictors. The prediction system comprises of 3 types of data sets: training, validation and testing data sets. Training data sets are the data, either from real transactions or synthetic data that is used to simulate the optimized condition in the VM. Validation data sets are derived from the training data, where it contains the baseline configuration regarding the desired condition in the VM. Testing data sets are the data that is used for subsequent test in the VM, and they are compared to the validation or training data sets to determine the hardware states.

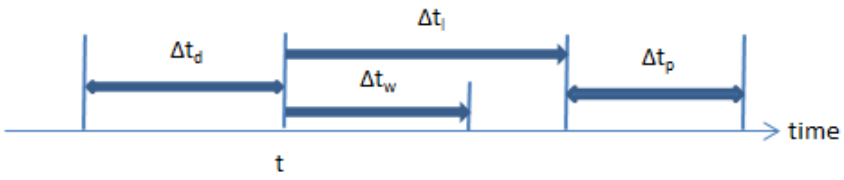


Figure 2.36: Time relations in *online failure prediction*. Adapted from (Salfner et al., 2010).

As the spectrum of *online failure prediction* is wide, Salfner et al. (2010) created a taxonomy to organize the structure of various approaches proposed by scholars. 4 main categories have been identified which are related to this research; they are briefly explained below.

- Category 1: Failure tracking. The potential failure in the future is predicted based on past failure which occurred on the same system.
- Category 2: Symptom Monitoring. Such failure prediction draws its input from the behavior of the system components, for example memory leaks, unusual high CPU usage, exceptionally high I/O etc.

- Category 3: Detected Error Reporting. This failure prediction methods deal with data from event-driven algorithms. For instance, the administrator can decide for a particular transaction, if the response time breaches certain threshold for *x* duration, the future failure is likely to happen.
- Category 4: Undetected Error Auditing. This is different from category 3, as the methods employed here aggressively scan the system for potential abnormality, instead of targeting certain parameters.

The resource *optimization* proposals in this thesis draw the motivation from the first 3 categories. The relevance of these structured methods in this taxonomy, to the proposals suggested in the thesis is exhibited below.

From category 1, the *Regression* and *Machine Learning* are the 2 techniques employed in the thesis's proposals. Figure 2.37 illustrates the way these 2 methods are utilized to determine the potential failure. The failure-prone and non-failure areas are determined after discounting the outliers and illegitimate data points, which are explained in subsequent chapters.

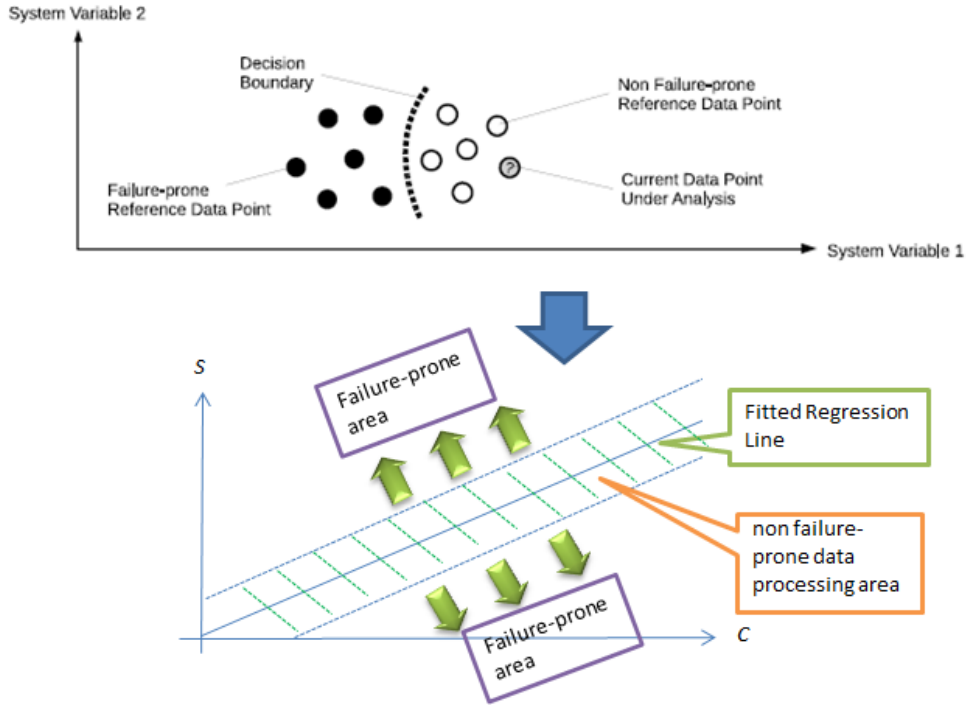


Figure 2.37: Online failure prediction method based on classification of system variable observations. Such observations from the authors are translated by the proposals in this thesis, as shown in the bottom figure. To arrive at the fitted regression line, both *Regression* and *Machine Learning* techniques are utilized. Adapted from (Salfner et al., 2010).

In Category 2, the methods in the taxonomy that are relevant to the proposed mechanism in this thesis are *Fuzzy Classifiers* and *graph models*, as in figure 2.38. *Fuzzy computing with words* (Zadeh, 1996) method is employed here to segregate and classify the observed patterns for failure prediction.

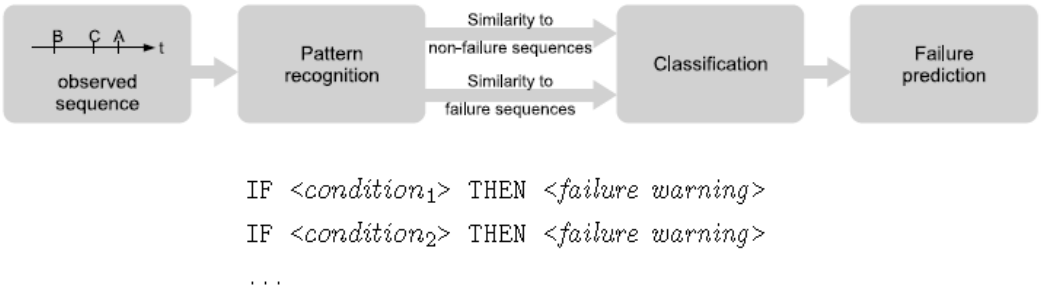


Figure 2.38: Online failure prediction method based on pattern recognition. *Fuzzy Computing with Words* (Zadeh, 1996) method is employed by the proposals in the thesis to classify the different patterns in failure prediction. Adapted from (Salfner et al., 2010).

The *graph-models* as illustrated in figure 2.37, is generated by the mechanisms as depicted in figure 2.39. The measurement values are obtained from the training and

validation data sets, and subsequent comparisons are made between the testing data sets with the validation data sets.

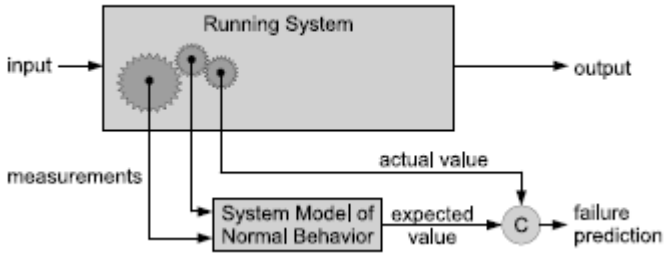


Figure 2.39: Online failure prediction method based on *system models*. The *graph-models* is categorized in this category. Adapted from (Salfner et al., 2010).

There is a time difference between the *online failure prediction* topics discussed by the authors and the proposals in this thesis. The authors envisaged that the prediction duration to actual failure for their discussed methods should be within few minutes to an hour. However this timeframe to predict future failure is longer for the failure prediction mechanisms proposed in this thesis, which could range from few hours to few days. This is because the results from the proposed *graph models* are sensitive to abnormalities; hence it can detect faults much earlier and allow for longer time-to-remediation solutions.

In the same topic, Gainaru et al. (2012) combined signal analysis concepts with data mining techniques (Teh, 2006) for the OS event log analysis to achieve the same objective of analyzing faults in the systems before failure. Signal analysis allows the characterization of the events that affect the systems. In this context, the normal behavior of a system is described and employed as baseline to subsequent collected traces. Then, Data Mining technique is engaged to extract the patterns in the collected data sets, search for correlations in the suspected events log and provide an adaptive forecasting method to predict the failures. The authors' proposal is depicted in figure 2.40. The normal signals are gathered and filtered. Instead of filtering out the outliers,

they are instead retained, and normal signals are gotten rid of. Subsequently the analysis is performed in the outliers, by scrutinizing on the outliers' pattern. In general, the authors found out that the longer the duration between outliers, the lower the chances for similarity, which lowers the 'confidence' level of the sequence of event. Such observation is exhibited in figure 2.40. In the diagram, the initial 4 outliers are almost having the same distance measurement. However the last outlier, which has a time lag much greater than the earlier outliers, is measured differently in its distance, hence is having lower confidence level of 68%.

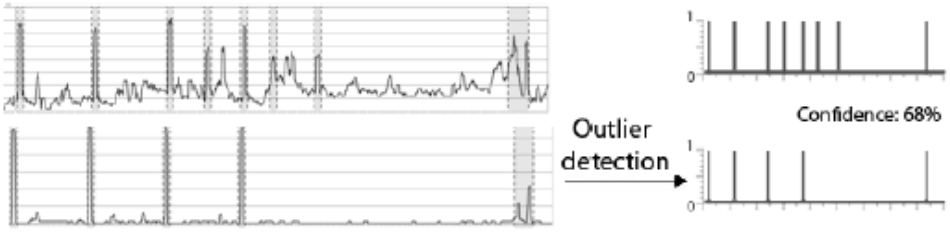


Figure 2.40: Fault detection strategy that filters out the normal signals, and leaves the outliers for analysis. Adapted from (Gainaru et al., 2012).

The probability of hardware component failure increases as the number of systems and hosts grows. Hardware component failure is the norm rather than an exception in cloud environments (Vishwanath & Nagappan, 2010). To compute the reliability level of particular cloud data center, Vishwanath et al. (2010) studied on failure trend on hardware components, data centers and hardware manufacturers. From hardware perspective, they found out that the component that is most vulnerable to failure is the hard disks. Using the data gathered and mined from 100,000 servers in a large data center, it is discovered that the annual failure rate (AFR) of the hard disks is about 2.7%, followed by memory module at 0.1%. Thus, the disks failure is the dominant issue in affecting hardware performance. However the type of hard disks, whether it is of SAS or SATA type does not have effect on the failure potential. Even though the price difference between these 2 classes of product is wide, the reliability characteristic

between them cannot be ascertained. The authors also uncovered that the age of the machines or any hardware components does not have correlation to the failure opportunity. The same is true for the configuration of the servers, location of servers in a rack and the type of workloads run on the servers; none of these contributes to the failure potential. However, the gathered data reveals that there is a failure correlation in the machines that have experienced failure in the past, that the chances for failure are higher in the machines that failed before, as compared to the machines of the same configuration but have not encountered any breakdown in prior. In addition, the location of the data centers and manufacturers of the hardware are having effects on the possibility of failure.

Table 2.5: Summary of studied researches with critical comment on sub-theme 'fault analysis and failure prediction'.

Scholars	contribution	comment
(A. Beloglazov et al., 2012)	Outlined 5 challenges to increase efficiency of resource utilization: 1) Optimization of VM Placement 2) Optimization on the virtualized network 3) Optimization on the thermal states and cooling system in the cloud data center 4) Optimization on workload consolidation 5) Achieving the equilibrium of conforming to SLA and maximizing resource utilization	Optimization in hardware performance is virtue, as with the other optimization criterions.
(Salfner et al., 2010)	Conducted detailed survey on computer system failure prediction methods.	The authors' focus in on short term assessment. The studied methods may be applied for longer term prediction.
(Gainaru et al., 2012)	Combined signal analysis concepts with data mining techniques on the OS event log to analyze faults.	The interesting method is that the analysis is performed in the outliers, by scrutinizing on the outliers' pattern.
(Vishwanath & Nagappan, 2010)	Studied on failure trend on the hardware components, data centers and hardware manufacturers.	The research discovered that the disks failure is the dominant issue in affecting hardware performance.

2.5.2 Resource utilization *optimization* models

In this section, the maximization of resource usage is discussed. The resource utilization rate in traditional servers in general is only about 8 – 10% (Gmach et al., 2008). Such utilization percentage is deemed wasteful, and affects the bottom-line of the application service offerings. However, this kind of server configuration is unavoidable, as the architecture of the system needs to consider burstiness in resource requirement. Particularly in mission critical situation, such configuration is important to ensure critical transactions can be processed without resource constraint. However, as mentioned earlier in the chapter, virtualization is addressing this resource optimization issue with the capability of VM placement, workload migration together with horizontal and vertical scaling of resources. To maximize the usage of the allocated resources, following studies have been conducted by scholars.

2.5.2.1 Task scheduling

To ensure optimization of resource utilization, efficient task scheduling is an important component in cloud computing. Task scheduling is also called job scheduling by some researchers. Here, the name of such mechanism is generalized to tasks scheduling. At high level, task scheduling involves the process to map particular set of tasks to available resources in the virtualized server cluster. Such mechanism is more suitable for web applications; however it is envisaged to benefit database operations in the near future. In RDBMS world, such scheduling mechanism is already made possible by Oracle, where job can be executed in any node which is part of the Oracle Real Application Cluster (RAC) (Oracle_RAC, 2013). However, such architecture is much different than available database hosting in cloud, as Oracle RAC requires homogeneous hardware in the RAC configuration. Nevertheless, it is believed that task scheduling paradigm will become viable in cloud in the very near future, as the segregation in the layers of computing and data nodes becomes more transparent. The

recent introduction of Oracle 12c has made available the pay-per-use licensing concept. Such facility will encourage aggressive task scheduling for database transactions very soon.

Jangra et al. (2013) provided a definition on tasks scheduling. Basically 3 steps are involved:

1) The broker discovers the available resources in the provisioned pool of VM, together with the detailed information regarding the percentile availability of these resources.
2) Tasks are mapped to the appropriate resources. This step is the most studied area by scholars, where efficient mapping of task-resource ensure optimization of hardware usage.
3) Tasks are assigned to the resource slice based on decision in step #2.

The challenges of task scheduling in cloud environment are mainly due to the high heterogeneity of computing resources, as well as high heterogeneity in the arriving jobs (Xhafa & Abraham, 2010). In their paper, Xhafa et al. (2010) proposed heuristic and meta-heuristic methods to address the requirement for good quality task scheduling mechanism. Their works focused on Grid computing. As Grid computing is a type of distributed computing, some of the discoveries in the paper can be assimilated in cloud environment. For instance, the heuristic method to determine the utilization of resources for particular job can be defined as

$$avg\_exec\_time(job\ A) = \frac{\sum_{i=1}^{n} exec\_time_i}{n},$$

where *n* is the total execution of *job A* in the particular VM.

In the virtualized cluster, *job A* could have been scheduled to various VM. The scheduling algorithm for new job can be of FIFO type, as typified in (Xhafa &

Abraham, 2010). Subsequently when the same job is run more frequently, more sophisticated scheduling can take place. If the job had run in *m* number of VM in the cluster, the execution time of *job A* can be arranged in an array, in the ascending order of execution time, such that the execution time of *job A*, $E_A = \{(e_1, VM_1), (e_2, VM_2), \ldots, (e_m, VM_m)\}$. Thus, the scheduling priority will be from VM_1 to VM_m. There are still a number of considerations, for example the condition in the VM at particular scheduling time, other scheduled jobs in the VM, threshold in the VM etc. Such scheduling mechanism for database transactions will be researched in the future works, not included in this thesis. These research works will address the heterogeneity of the resources and tasks.

In order to maximize profit from application service offerings, Li et al. (2010) proposed an optimization method to choose the VM where the tasks are to be scheduled. They applied the *Grobner bases* theory to solve the *stochastic integer programming*, which is a type of optimization programming method similar to *linear programming* proposed in this thesis. Figure 2.41 illustrates their envisaged scheme. The 'Abstract service' denotes a functionality that is similar to partitioned workload in a business process, which is independent in nature and can be allocated to particular VM for processing. The *resource-i-j* represents the VM slices. Each *resource-i-j* has certain capability that is capable of producing *x* amount of throughput with *y* amount of latency, at the expense of *z* cost. With a *p* probability of SLA fulfillment by each *resource-i-j*, the challenge is hence to find the most cost effective *resource-i-j* that can meet the required SLA. The proposal looks perfect in the ideal world. However the problems observed here are:

1) How to provision the standby *resource-i-j* to service the 'abstract service'.
2) How to arrive at the probability distribution of QoS, *q* in the figure for each *VM* for each 'abstract service'.

3) How to calculate the latency and throughput with the *q* percentile.

The above 3 questions were not detailed in the paper. The authors' proposal is incomplete. However the idea of employing the optimization algorithms once these parameters can be discovered is very exciting.

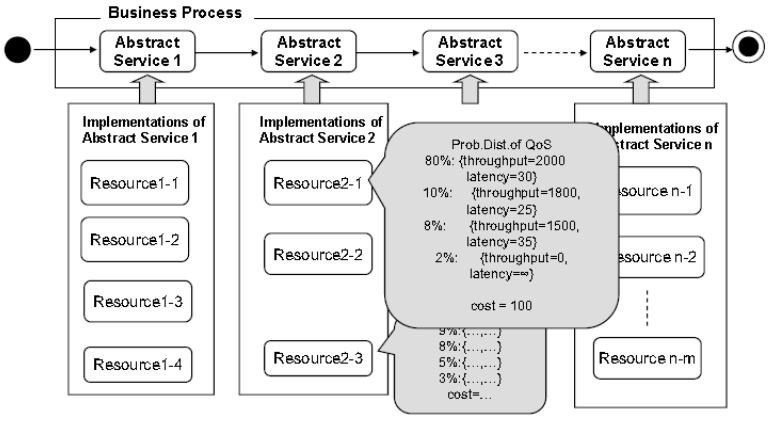


Figure 2.41: Task scheduling problem, with standby VM to service the 'abstract services'. The goal is to find the VM that is most cost effective in serving these 'abstract services', at the same time complying with the required SLA. Adapted from (Q. Li & Guo, 2010)

In the same topic of maximizing resource utilization, Hsu, Chen, and Park (2008) proposed a task scheduling method called Extended Smallest Communication Ratio (ESCR) for grid computing environment, which is also applicable to other distributed systems, especially cloud. In their proposal as depicted in figure 2.42, there is a master server serving as the "broker" to disperse tasks to various nodes in the cluster. This broker functions similar to load balancer in conventional web service architecture. However the difference is that this broker considers more parameters in deciding which nodes to send the tasks to. Moreover, instead of a single node, the tasks can be sent to a cluster of nodes.

The tasks are sent to the nodes to be processed, based on the availability of the CPU resource, as shown in figure 2.43. The authors claimed that their proposed ESCR resource allocation algorithm is able to optimize the CPU resource by minimizing the 'idle' time as much as possible. Lin, Liang, Wang, and Buyya (2012) improved the task

scheduling methods by incorporating network bandwidth into consideration in determining the VM resource state. The authors claimed the novelty here, that the resource depiction only from CPU and memory perspective is deemed insufficient. They employed the nonlinear programming to solve the task distribution problem, via heuristic approach. The authors also considered the scheduling problem as NP-complete.

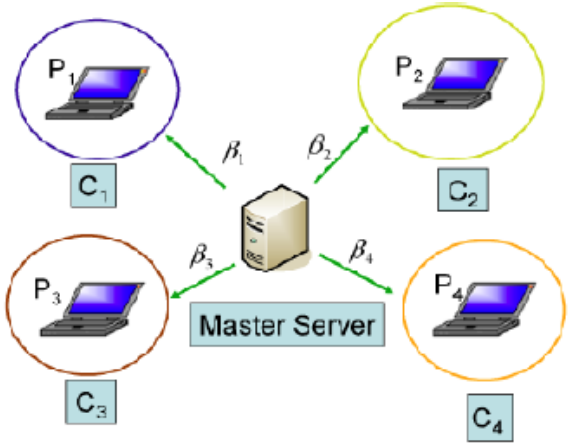


Figure 2.42: Task scheduling system. The C_1, C_2, C_3 and C_4 denote the nodes where tasks are to be processed. β_1, β_2, β_3 and β_4 are the criterions that help the broker to disperse the tasks. Adapted from (Hsu et al., 2008).

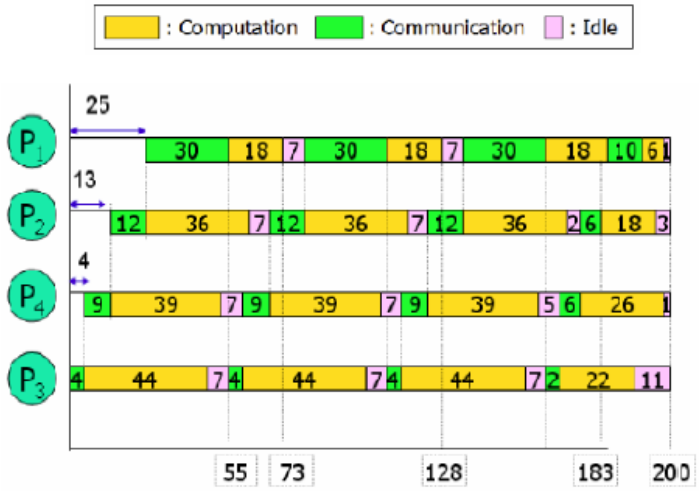


Figure 2.43: ESCR task allocation algorithm. P_1, P_2, P_3 and P_4 are the processors where tasks are processed. The initial delays (4, 13 and 25 units of time) are caused by the communication issue due to the distance between the broker and the nodes. In this diagram, the deadline to complete the tasks is set to 200 units of time. Adapted from (Hsu et al., 2008).

Table 2.6: Summary of studied researches with critical comment on sub-theme 'task scheduling'.

Scholars	Contribution	Comment
(Xhafa & Abraham, 2010)	Proposed heuristic and meta-heuristic methods to address the requirement for good quality task scheduling mechanism.	Apart from execution time criteria, there are other considerations, for instance the condition in the VM at particular scheduling time, other scheduled jobs in the VM, threshold in the VM etc.
(Q. Li & Guo, 2010)	Proposed an optimization method to choose the VM where the tasks are to be scheduled.	Some shortcomings not covered in the paper are: 1) How to provision the standby resource-i-j to service the 'abstract service'. 2) How to arrive at the probability distribution of QoS, q in the figure for each VM for each 'abstract service'. 3) How to calculate the latency and throughput with the q percentile.
(Hsu et al., 2008)	Proposed a task scheduling method called Extended Smallest Communication Ratio (ESCR).	The resource allocation mechanism is translated to resource management in 1 VM, which is illustrated in subsequent section for future research.
(Lin et al., 2012)	Incorporated network bandwidth in scheduling problem.	The authors did not detail the complexity of combining CPU, memory and network bandwidth variables in the scheduling problem. Potentially such aggregation of input to depict the VM's resource state can boost the accuracy of the task scheduling problem.

The above resource allocation mechanism can be translated to resource management in 1 VM. The following mechanism is envisaged for future work in optimizing the resource utilization in particular VM, especially on CPU cycle. It will be detailed and refined in future publication of journal paper.

Proposal to optimize resource utilization – a high level view for future research

During the steady-state database operation in a VM, the interest is to find out resource utilization pattern by the database processes. In particular time interval, histogram is employed to record the resource utilization. The samples of SQL processing time, *S* and server load, *C* are gathered at 5 minutes interval. Subsequently 1 dataset is defined as

sample of data points collected in 3 hours timeframe, *T*. Hence there are 36 data points for analysis. The choice of 3 hours duration is also due to the fact that it can comfortably accommodate most long running data analytical processes in entirety. However this timeframe can be varied as needed. In real world production mode, this frequency of data collection can be adjusted to accommodate the allowable overhead in the system, for instance, 1-minute or 2-minute interval. This increase of sample collection frequency can produce more accurate results for analysis as the visibility into the resource condition in the VM is increased.

Data analytical processes, unlike OLTP processes, often can be adjusted to fit into timeframe where the processes can run best without hindrance from resource constraint or OS noises. The activities in each time slot are presented by first defining some parameters as follows:

D_i = Day of the week when the datasets are collected. *i* represents values from 1 to 7, denoting 7 days in a week.

T_j =3-hour time-block in D_i. *j* represents values from 1 to 8, denoting 8 blocks of test duration in a day.

d_j = Dataset where samples are contained.

C_k = 1-minute average server load at particular point in time in T_j. *k* represents values from 1 to 36, as there are 36 data points in 1 dataset.

S_k = total SQL Elapsed time in the database at particular point in time in T_j.

S_T = Corresponding SQL Elapsed time at C_T.

$\hat{S}_h$ = *70^{th} percentile SQL Processing time* (C. H. Tan & Teh, 2013a), measured from 0 to S_T.

C_h = Corresponding server load at $\hat{S}_h$.

Using *Fuzzy Computing with Words* to characterize the relationship (Zadeh, 1996):

- *If a lot of data samples exceeds server load, C_h* ***OR*** *many occurrence of database processes holding S_k > Total SQL Processing time, $\hat{S}_h$ for more than q minutes, the test block, T_j is busy with database processes.*
- *If T_j is busy with database processes, additional database processes cannot be scheduled in the timeframe.*

To explain the logic, additional 3 parameters are introduced, *m*, *n* and *q*. So If *m* percentile of data samples exceeds server load, C_h **OR** more than *n* occurrences of database processes holding S_k > Total SQL Elapsed time, $\hat{S}_h$ for more than *q* minutes, the test block, T_j is busy with database processes. Assume, C_h =4.4, *m*=20, *p* =0.8, *n*=2 and *q*=10 in the explanation here. If 20% of data samples exceeds C_h, **OR** with more than 2 occurrences of continuous data processing hold Total SQL processing time, S_k > $\hat{S}_h$ for more than 10 minutes, T_j is deemed busy and not suitable for additional scheduling of database jobs. Else more database maintenance jobs can be added to T_j.

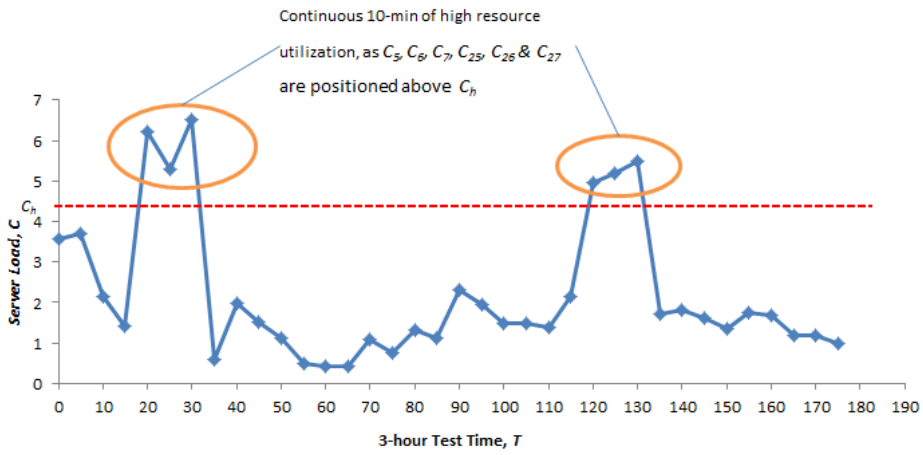


Figure 2.44: Proposed resource optimization by introducing more efficient task scheduling in a particular VM. This proposal is to be refined in subsequent journal paper.

Figure 2.44 shows the server load status in the *VM* for a dataset d_1 in T_1. As depicted, there are 2 continuous-10-minute high resource utilization blocks in the dataset. The data points at C_5, C_6 & C_7 and C_{25}, C_{26} & C_{27} are running maintenance jobs that constantly hitting the *VM* for more than 10 minutes over the boundary of C_h.

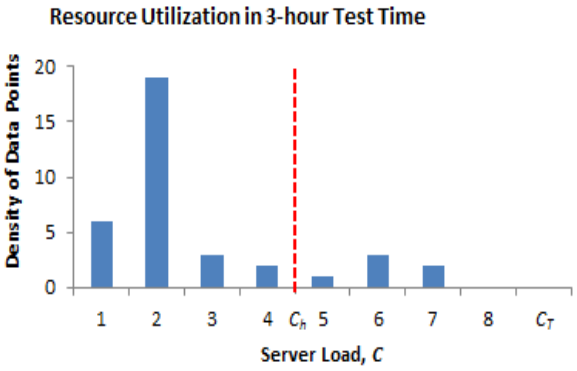


Figure 2.45: Server busyness using server load limit as gauge.

Figure 2.45 shows the 'busyness' of the *VM* in T_1. 6 out of 36 data points are beyond the red line. Hence as < 20% of data points are > C_h, the condition is also not met to deem T_1 as busy. The theory thereon allows more database maintenance jobs to be scheduled in T_1. As illustrated in Figure 2.45, the server load, *C* has been grouped into classes, and the height of each bar indicates the density of the data points collected in the *Workload Repository*. Values which fall on a boundary are counted in the upper class. The width

of the bar is the same, and they are chosen to reveal the resource utilization in that duration.

The above task allocation proposal in a single VM is simplistic; however a successful implementation of such mechanism can greatly increase the utilization rate of hardware resources in the virtual environment, thus achieving great saving in capital and operational costs.

2.5.2.2 Auction-based resource scheduling

Optimization can be achieved from another aspect in cloud. Wang et al. (2013) proposed a way to allocate resources to the cloud consumers, via a reverse auction based mechanism. In such scenario, it is assumed that the providers are having limited resources to be allocated, which is rarely the case for today's commercial cloud providers. However if resources are to be sourced from the smaller providers with consumers rallying to take advantage of the cheaper offering from such providers, this Auction-based allocation mechanism will be beneficial. The model works, starting with the potential clients tender their resource requests to the providers. Such requests will contain the information of needed Quality of Service (QoS) and amount of resource blocks. These requests are delivered to multiple providers, which in return will establish the associated cost with the tendered requests. This costing information will be submitted to an intermediary broker (IB), who will make decision, on which clients to get how much of the resources. The authors proposed to utilize the *Vogel's Approximation Method* (VAM) (VAM, 2013) to optimize the resource allocation problem at the stage when the allocation is to be made by the *IB*. They didn't detail how the VAM can be deployed for such allocation mechanism, however the provisioning algorithm can be envisage as follows. In figure 2.46, the blue cells denote the cost of providing the resources from the providers. In deriving this cost structure, the providers

will decide based on the QoS and amount of resources needed by the clients, plus any other criterions deemed essential by the providers for particular clients. The green cells contain the resource slices owned by each provider, which are readily to be sold to potential consumers. The purple cells show the resource requirements from each client.

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	10	30	25	15	14
Provider 2	20	15	20	10	10
Provider 3	10	30	20	20	15
Provider 4	30	40	35	45	12
requirement	10	15	12	15	

Figure 2.46: Optimization for supply and demand Auction-based resource allocation. The optimization is to be conducted by an intermediary broker based on the information in this table.

The next step is to apply the VAM to the table's information. In this algorithm, the supply and demand amount must be added up to have the same values. If they are not the same, then a dummy consumer or provider will need to be added to the table, as in figure 2.47. In this case the demand is greater than the supply; hence the clients' requirement cannot be fulfilled in total. However the consequence of such scenario is not to be discussed here.

	0	15	0	5		
	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply	
Provider 1	10	30	25	15	14	5
Provider 2	20	15	20	10	10	5
Provider 3	10	30	20	20	15	10
Provider 4	30	40	35	45	12	5
dummy	0	0	0	0	1	0
requirement	10	15	12	15		

Figure 2.47: First step in the VAM optimization. The red cell indicates the highest value derived by comparing the values of differentiation between the lowest 2 cells' values in each row and column. The yellow cell is the identified cell to have maximum resource allocation into it.

The VAM computation is to be carried out in following steps:

1) The difference in value between the lowest 2 cells in all columns and rows, excluding the dummy's, is to be computed.

2) The values in step #1 are compared, and the highest value is to be identified.

3) With the value obtained from step #2, the cell that has the lowest value in the row or column associated with the highest value as per step #2 is to be allocated with maximum possible resources. If there are same computed values in step #2, the row or column can be arbitrarily chosen.

4) Step 2 and 3 are repeated. If any row or column has the supply and requirement value maximized, the other cell in the row or column is marked 'x', as in figure 2.48.

5) The optimization steps are complete when all the cells contain either the allocated resources or 'x' value.

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	10	30	25	15	14
Provider 2	x	10	x	x	10-10=0
Provider 3	10	30	20	20	15
Provider 4	30	40	35	45	12
dummy	0	0	0	0	1
requirement	10	15-10=5	12	15	

Figure 2.48: The supply of resources from provider 2 has been allocated in full to consumer 2. The other consumers will not get any more resource from this provider, hence their cells are marked 'x'.

The rest of the optimization steps are depicted in Appendix A. The end result is an allocation as shown in figure 2.49. With this result, the *intermediary broker* achieves its objective to optimize the resource allocation based on the best pricing that can be offered by the providers together with the quality and quantity of demand from the clients.

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	x	4	1	0
Provider 4	x	5	7	x	0
dummy	x	x	1	x	0
requirement	0	0	0	0	

Figure 2.49: The green colored cells indicate the resource allocation by the providers to each consumer.

A similar Auction-based resource allocation mechanism is proposed by Buyya et al. (2009). The authors quoted that the hurdle to universal embracement of such Auction-based system is the non-standard interfaces which are needed by the consumers to migrate their applications. Due to this issue, it is difficult for the consumers to interact with each provider to discover the best pricing and services that are offered by the providers. Realizing this shortcoming, the authors proposed Meta-Negotiation Middleware (MNM), which is a prototype to enable global exchange of cloud services. The objective is to ease the interpretation of services provided by each cloud provider. As depicted in figure 2.50, the consumer sends the request for resource to the MNM, so that the information can be 'normalized' to fit the interface standard of particular provider. The provider will take the input from the received translated information from MNM, process and submit the pricing and service offering to the potential client. In such model, the consumer will initiate multiple negotiation sessions with many providers, by tagging along the MNM agent. Eventually the consumer will choose the best offer in the market based on the feedback received from various providers.

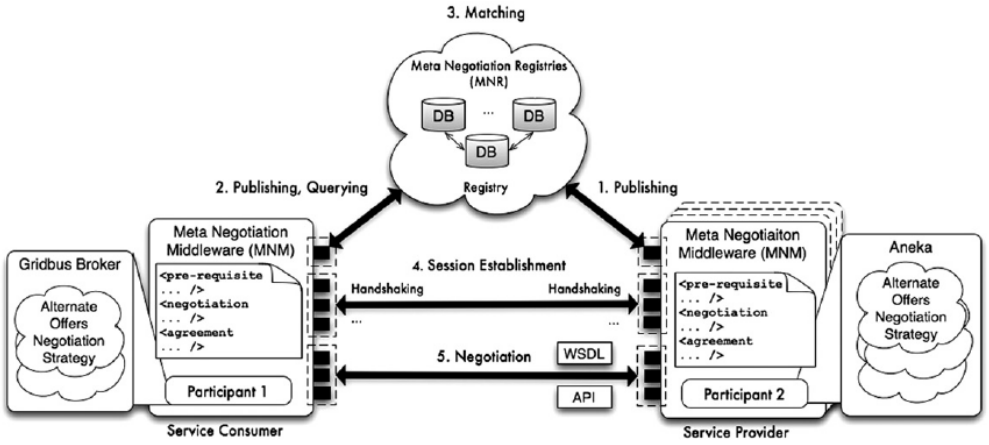


Figure 2.50: Negotiation of services and pricing between the consumer and cloud provider. The Meta-Negotiation Middleware translates the information for the consumer to arrive at the best market offer. Adapted from (Buyya et al., 2009)

An et al. (2010) considered the uncertainty in the negotiation process between the consumers and providers. During the negotiation stage, a particular consumer can bid for resources from multiple providers. However, the consumer can only consume a finite number of resources offered by 1 or more providers, but it cannot take on the amount of resource beyond what is needed for the application. At the same time, a provider may receive multiple resource requests from consumers. The provider can only offer resources to the potential clients with the resources it has on hand. In such scenarios, the authors proposed a mechanism for the consumer to quickly decommit or cancel the agreement entered with a provider, by paying a certain amount of penalty, when a late but more favorable offer is offered by another provider. The same mechanism is applied for the provider, where it can cancel a deal entered with a client when a higher bid for resource is tendered by another client. An agent is envisaged to detail out the decommitment and cancelation of such agreements. Figure 2.51 shows the high level explanation of the prototype.

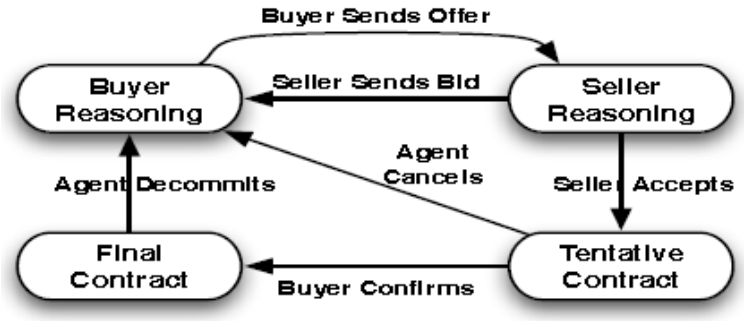


Figure 2.51: The negotiation process for finite state of buyer's request and provider's resources. The agent is engaged to negotiate, confirm, decommit or cancel the deal. Adapted from (An et al., 2010)

From commercial perspective, a lot of companies are engaging cloud billing company, for example Aria Systems (Aria, 2013) to automate the billing management of rapid changes in the resource allocation. Such systems can be altered to accommodate the Auction-based mechanism, both from providers and consumers. With such systems in place the focus is shifted towards generating the most efficient supply-and-demand pricing procedure.

The Auction-based resource allocation optimizes the resource provisioning mechanism in the Public Cloud environment. The future database operations can make use of such model to ensure reduction in capital and operational cost, particularly from *cloud bursting*. The studied literatures provide a fundamental idea how the resources can be provisioned from the Public Cloud vendors in the most optimal manner, from the perspective of costing and service offering.

Table 2.7: Summary of studied researches with critical comment on sub-theme 'Auction-based resource scheduling'.

Scholars	Contribution	Comment
(Wang et al., 2013)	Proposed a way to allocate resources to the cloud consumers, via a reverse Auction-based mechanism.	It is assumed that the providers are having limited resources to be allocated, which is rarely the case for today's commercial cloud providers. However it is beneficial for smaller cloud providers.

(Buyya et al., 2009)	Quoted that the hurdle to universal embracement of Auction-based resource allocation system is the non-standard interfaces which are needed by the consumers to migrate their applications. Proposed Meta-Negotiation Middleware (MNM), to enable global exchange of cloud services.	The introduced middleware needs to be embraced by all cloud providers in order to serve its purpose.
(An et al., 2010)	Considered the uncertainty in the negotiation process between the consumers and providers.	The introduced agent-based system is attractive in perform resource negotiation.

2.5.2.3 Resource brokering – the essence of *cloud bursting*

When the number of nodes in the *VM* or the duration of service time breaches a threshold, the workload is sent to be processed by Public Cloud as the resource is more reliable. Until this point in this chapter, the determination of resource adequacy is based on threshold limit from operating system parameters, by taking feedback from end users or administrators. Nevertheless, these inputs are often not accurate as human interpretation of the resource requirement tends to be overblown. Hence, scholars have conducted studies in *resource planning* by analyzing the real or synthetic workloads. The notable method frequently studied is autonomous resource brokering. Even though Private Cloud is the focus in this survey, this resource brokering in Hybrid Cloud is also studied as this technology has matured in Grid platform, which will eventually benefit the cloud deployment. IT infrastructure has evolved in the way Cloud is being utilized. Instead of application hosting solely in public or Private Cloud, IT architects have combined the hosting in both. This is to take advantage of the massive scalability and cost overhead reduction, couple with more stabilized hardware in Public Cloud; whereas some classified transactions can still be preserved to run in Private Cloud.

The predecessor to above *cloud bursting* brokering scheme is detailed in (Javadi, Kondo, Vincent, & Anderson, 2011). Before the topic of *cloud bursting* is elaborated, it

will be interesting to take an insight on how the research evolved from the objective of discovering availability and unavailability services in the host to resource brokering. In this paper, Javadi et al. (2011) first examined the problem of discovering availability and unavailability models for hosts running in a large distributed system. In this particular study, the CPU component's state is measured instead of the entire system as a whole. The traces for realistic scenarios are gathered via BOINC (Anderson, 2004), which is a middleware for publicly volunteer distributed computing. The data collection randomly identifies subsets of hosts whose availability have similar statistical properties and can be modeled to other larger systems with similar probability distributions. The test runs for close to 2-year duration and captures 57,800 years of CPU time and 102,416,434 continuous intervals of CPU availability in 230,000 hosts. From the data, probability distributions are modeled from the hosts that have truly random availability and unavailability intervals. To group and cluster the hosts based on their similar trace distribution, two standard clustering methods are employed; they are k-means (Elkan, 2003) and Hierarchical clustering (Manning et al., 2009). These 2 clustering methods have been discussed earlier in this chapter. The distance metric from Cramer-von Mises (Laio, 2004) is found to be the more suitable distance measurement algorithm for the host clustering effort by these 2 methods. Distance measurement using Cramér–von Mises method can be explained as follows.

Consider a group of parametric criteria to measure the CPU availability, x_1, x_2, ..., x_n, which is arranged in ascending order. The Cramér–von Mises statistic for this group of value is

$$T = \frac{1}{12n} + \sum_{i=1}^{n} [\frac{2i-1}{2n} - F(x_i)]^2,$$

where,

n is the total number of samples.

x_i is the *i-th* order in the smallest value in the sample.

F is the perceived distribution function of the host availability.

In this case, the distance measurement is provided by value T. If T is within a tabulated value for a cluster, the x_i can be grouped in 1 cluster. Subsequently the host clusters can be aggregated via different value of T.

It is also found that combining availability and unavailability greatly reduce the confidence and accuracy of the clustering methods; hence they are segregated during clustering. After the clusters of hosts are determined, parameter fitting for various distributions is conducted via the maximum likelihood estimation (MLE) (Myung, 2003). Goodness of fit (GOF) (Narsky, 2003) of the resulting distributions for each cluster via standard probability-probability (PP) plots using a visual method or quantitative metrics are subsequently carried out. It is found that Gamma distribution is more suitably representing availability distribution, while unavailability distribution in large distribution systems can generally by represented by hyper-exponential distribution. With these 2 distributions plotted and the mean and variance values obtained, the authors applied them into the resource brokering mechanism as described below.

With these establishments of the distribution models, they are applied to the resource brokering problem. Figure 2.52 depicts the resource brokering model envisaged by the authors.

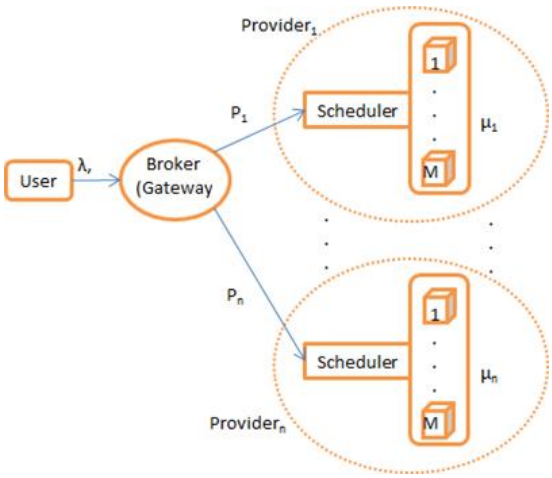


Figure 2.52: Resource brokering model envisaged by Javadi et al.. Adapted from (Javadi et al., 2011).

In this context, a broker is responsible to route a series of incoming jobs to a set of schedulers, one for *n* number of clusters, in order to service the requests. The scheduler in turn dispatches the tasks to the providers within the cluster. The broker determines which schedulers to send the workload requests to, by the possible completion time of the jobs. This is the part where the availability and unavailability distributions play their roles. They employed the probability distribution function (PDF) proposed by Kleinrock et al. (1993) for job completion time in aggressive distribution systems to compute the hypothetical job completion time, as input to determine which scheduler to send the workloads to. The usage of availability and unavailability distributions in PDF is depicted below:

The PDF of time *t*, with workload *W* and *M* processors, $f(t) = \frac{W}{\sqrt{2\pi\sigma_b^2 t^3}} \exp[-\frac{(W-bt)^2}{2\sigma_b^2 t}]$,

where,

$$b = \frac{t_a}{(t_a+t_u)} M \text{ and } \sigma_b^2 = \frac{\sigma_a^2 t_u^2 + \sigma_u^2 t_a^2}{(t_a+t_u)^3} M.$$

The mean of the PDF, $\bar{f} = \frac{W}{b} = \frac{W}{M}\frac{(t_a+t_u)}{t_a}$, and the variance of the PDF, $\sigma_f^2 = \frac{W}{b}\frac{\sigma_b^2}{b^2}$

In these formulas, the t_a, t_u, σ_a^2 and σ_u^2 are the means and variances of the availability and unavailability distribution proposed by Javadi et al. (2011). Such resource brokering proposal paves the way for further enhancement. In the above model, the computed job completion time is fed to the broker for routing decision. From figure 2.52, the *Cloud Bursting* phenomenon can be visualized. In this case the $provider_1$ represents the internal Private Cloud, whereas $provider_2$ depicts the cluster in Public Cloud.

The authors further extended their works to include cost and performance aware provisioning policy in (Javadi, Abawajy, & Buyya, 2012; Javadi, Thulasiraman, & Buyya, 2012). The InterGrid gateway (IGG) as depicted in figure 2.53 is acting as the broker in figure 2.52. It is responsible to route the request to either the public or Private Clouds. In this case the users submit the requests to IGG, by providing the information regarding the duration of the workload processing, the required VM to process the workloads and the QoS of the required computing services. In addition to the brokering service, the IGG is also capable of performing the scheduling algorithms, by interacting with another IGG or Virtual Infrastructure engine (VIE). The VIE component is unique for Private Cloud, as it is capable to start or shutdown the VM in the cluster as needed. Figure 2.53 shows how workload request is sent for processing in this Hybrid Cloud architecture. This architecture is designed and implemented by Cloudbus research group (Costanzo, Assunção, & Buyya, 2009). The darker greyed text boxes in this figure indicate the route taken for Public Cloud resource provisioning, while the lighter grey boxes denote the Private Cloud route. Resource provisioning policies are built into this model. Their inputs are either based on the workload model known beforehand or from failure correlations established by Failure Trace Archive (FTA) (Kondo, Javadi, Iosup, & Epema, 2010), in order to fulfill a common QoS requirement. The measurements are established from deadline violation rate, job slowdown, and performance–cost efficiency. When the number of nodes in the VM or the duration of servicing time

breaches a threshold, the workload is sent to be processed by Public Cloud as the resource is more reliable, hence avoid job resume or restart scenarios due to spatial (Fu & Xu, 2010; Gallet et al., 2010) or temporal correlation (Yigitbasi, Gallet, Kondo, Iosup, & Epema, 2010) in hardware failure events. Spatial correlation denotes multiple failures that happen in multiple nodes in the cluster within short timeframe. This normally is caused by environmental factor, for instance the change in temperature in the data center. Temporal correlation is the failures that occur by not adhering to random order, where skewness in the distribution is observed over time.

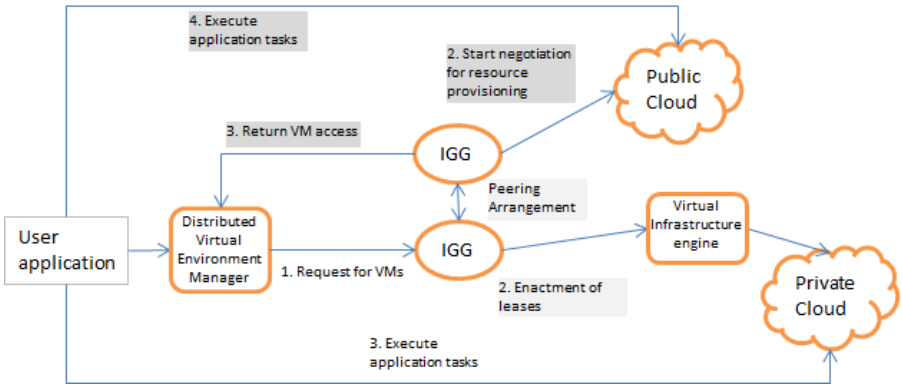


Figure 2.53: Resource brokering model envisaged by Javadi et al., with *cloud bursting* mechanism incorporated. Adapted from (Javadi, Thulasiraman, et al., 2012).

The computation of routing possibility, P_i by the broker to the providers, based on cost and performance criteria of the resource providers, is exhibited by the authors via mathematical models. To illustrate these mathematical models, figure 2.52 is made reference in the following explanation.

The cost influence to the broker routing decision

To explain this, following sequence will bring reader to understand how the hosting cost is affecting the routing path:

1) λ = job arrival rate from the users to the broker. Given *I* as the distribution of job arriving at the broker, the mean of this distribution, $E[I] = \lambda^{-1}$ and variance, $V[I] = \sigma_I^2$.

2) Furthermore, assume distribution of service time of queue *i* in provider *i* is S_i, and this distribution has the mean value, $E[S_i] = \mu_i^{-1}$, and variance, $C_{Si} = \sigma_{Si} \cdot \mu_i$.

3) Another assumption needs to be made for distribution of arrival time of jobs in queue *i*. The mean is given by $E[I_i] = \lambda_i^{-1}$, with variance,

$$V[I_i] = \frac{\sigma_I^2 P_i + \lambda^{-2}(1-P_i)}{P_i^2}, \qquad (1)$$

4) Assume K_i is the cost to be paid to the provider *i* for the usage of resource per unit of time.

5) $E[T_i]$ is the expect response time of the job serviced in queue *i*. The authors derived this value as

$$E[T_i] = \frac{1}{\mu_i} + \frac{C_{Ii}^2 - C_{Si}^2}{2(\mu_i - \lambda_i)}, \qquad (2)$$

where C_{Ii}^2 is the square coefficient of variance on arrival time of jobs in queue *i*. From equation (1), C_{Ii}^2 is derived as $C_{Ii}^2 = 1 + P_i(\lambda^2\sigma_I^2 - 1)$.

6) The objective function of the broker is to minimize the cost associated with the services provided by the providers and the expected response time, so optimization algorithm can be applied here. In mathematical form, it is to achieve

$$min \sum_{i=1}^{n} (K_i \cdot E\,[T_i]), \qquad (3)$$

7) The authors applied the *Lagrange multipliers* method to optimize the broker's objective function in equation (3). From equation (2), they derived that the optimized routing probability of jobs by the broker to the providers,

$$P_i = \frac{\mu_i}{\lambda} - \frac{\sum_{i=1}^{n} \mu_i - \lambda}{\lambda} \cdot \frac{\sqrt{K_i \eta_i}}{\sum_{i=1}^{n} \sqrt{K_i \eta_i}}, \quad (4)$$

where $\eta_i = \lambda(\lambda^2 \sigma_I^2 + \lambda^2 + C_{Si}^2 - \lambda \mu_i)$.

8) Hence, from equation (4), the broker routing decision is taking the cost charged by the providers as a criterion.

The performance influence to the broker routing decision

From another perspective, the routing path can be guided by the performance model. The authors employed the "Average Weighted Response Time"(AWRT) (Grimme, Lepping, & Papaspyrou, 2008) and "bounded slowdown" (Feitelson, Rudolph, Schwiegelshohn, Sevcik, & Wong, 1997) parameters. They computed that for *N* number of requests, the AWRT is defined as the average time that the user must wait in order for the request to be completely processed. Mathematically it is defined as

$$AWRT = \frac{\sum_{j=1}^{N} d_j \cdot v_j \cdot (ct_j - st_j)}{\sum_{j=1}^{N} d_j \cdot v_j}, \quad (5)$$

where,

d_j is the run time of request *j*

v_j is the number of VM requested by the user to service request *j*

ct_j is the completion time of request *j*

st_j is the submission time of request *j*.

The job slowdown is the total response time plus other overheads, for instance the queuing in the provider end or delay in other component involved in the job scheduling. It is defined as

$$\text{Slowdown} = \frac{1}{N} \sum_{j=1}^{N} \frac{w_j + \max(d_j, bound)}{\max(d_j, bound)}, \quad (6)$$

where,

w_j is the waiting time of request j

The bound is set to 10s to prevent short requests from hindering the parameter.

Both AWRT and "bounded slowdown" parameters are computed for each workload, which are stored in a repository. When similar workloads are routed to the broker, it can make the routing decision based on the historical result of the workload models. Hence the goal of using performance metric to influence and optimize the job scheduling mechanism is achieved.

Fito et al. (2010) studied on the economics of Hybrid Cloud. Their works focused on web applications, and the benefit of cloud elasticity is expressed in monetary unit. Figure 2.54 illustrates the components in a typical modern web hosting architecture that makes use of cloud computing. In this depiction, the Cloud Hosting Provider (CHP) denotes the VM which is readily available in the Public Cloud, whereas the Web Hosting Provider (WHP) represents the in-house web servers. When WHP is overloaded, the resource needs is '*burst'* to CHP. The Web Service Monitoring (WSM) monitors the resource utilization in the whole system, and it provides input to the Scheduler to decide the routing of further tasks from the front end clients via the proxy server.

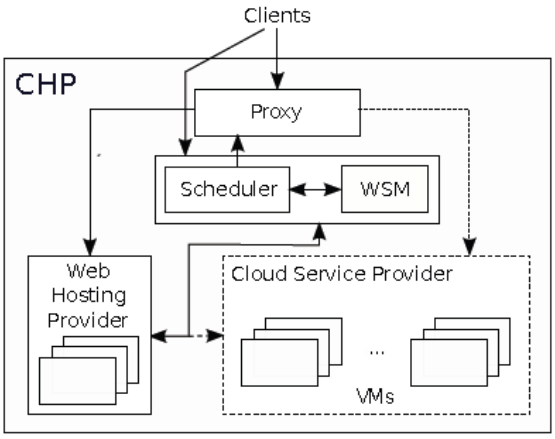


Figure 2.54: *Cloud Hosting Provider* architecture envisaged by Fito et al. Such system will become typical web hosting model in the very near future. Adapted from (Fito et al., 2010)

Most of the details in the components have been discussed in literature reviews in above sections. The novelty of this paper is the study conducted to exhibit the monetary benefit of using cloud services, compared to web hosting in static servers. Figure 2.55 illustrates such comparison. The monetary unit in the bottom figure is formulated by considering the SLA violation cost versus hardware cost and the maintenance cost of the hosting systems. The authors showed that cloud hosting outperforms the static hosting in many orders of magnitude as shown in the bottom figure.

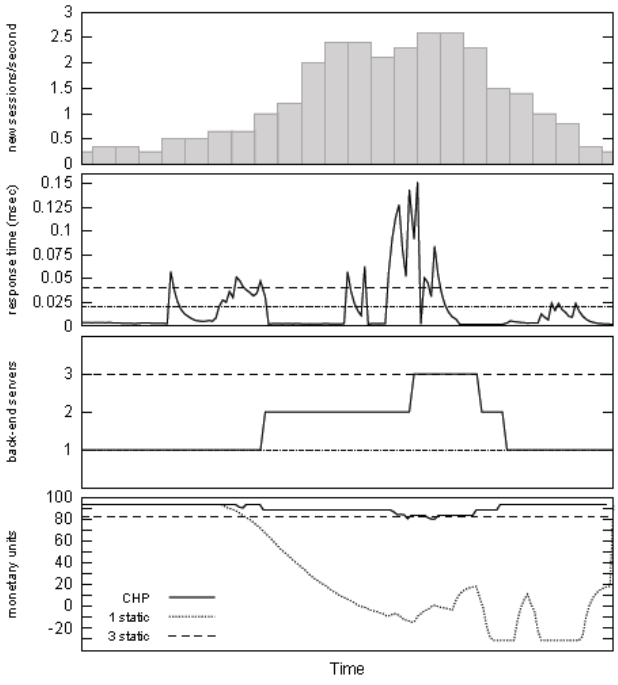


Figure 2.55: Revenue benefit in deploying Hybrid Cloud computing compared to static servers. Adapted from (Fito et al., 2010)

All the discussed Hybrid Cloud computing in above literatures are studied on web application domain. There is not any convincing literature that discusses such '*burst'* of resource to the Public Cloud in the *Parallel Database* domain. Nevertheless, with the overcome of hurdles in security concerns and some limitations in the RDBMS technology, such architecture will be very beneficial to the wide industry as it promises optimization in hardware usage, which can significantly boost the bottom line of the business by greatly reducing the IT spending on hardware.

In the proposals in this thesis, *optimization* is performed using the *linear regression and machine learning methods*. Synthetic TPC-H queries are utilized to load the VM into certain resource limit in the VM, and the relationship between the SQL processing duration versus CPU run queue size is compared to the baseline data, to determine if the hardware is performing to its best optimized condition.

With the performance assurance on the hardware, the subsequent section can convincingly deal with the response time verification. In the next topic, resource utilization *affirmation* is scrutinized, with the target to create stress-testing scenario in the VM in order for critical transactions' response time to be tested.

Table 2.8: Summary of studied researches with critical comment on sub-theme 'resource brokering – the essence of cloud bursting'.

Scholars	Contribution	Comment
(Javadi, Abawajy, et al., 2012; Javadi, Thulasiraman, et al., 2012)	Proposed resource brokering that includes cost and performance aware provisioning policy.	Cloud Bursting phenomenon is incorporated in the brokering model. The proposal can be extended to Hybrid Cloud model.
(Fito et al., 2010)	Studied on the economics of Hybrid Cloud, by focusing on web applications. The benefit of cloud elasticity is expressed in monetary unit.	The envisaged Hybrid Cloud architecture may potentially become typical web hosting model, in view of the capability of 'bursting' to Public Cloud, which provides different hosting benefits as compared to in-house hosts.

2.6 Resource utilization *affirmation* – stress testing

Stress testing, also known as performance testing, is one of the very important components in application service offerings. Problems after the applications have gone live and started serving the wide user base are rarely due to functionality issues. This can be understood as any functionality bug would have been identified during the construction phase. During steady state production, the most encountered issue is performance degradation due to sudden surge in transaction volumes, or uncharacterized usage patterns from the end users. When these happen, stress testing is employed to re-verify the resource adequacy to accommodate such changes. Generally, these deviations from predicted volume of transactions and changes in application usage patterns are a norm rather than an exception. Hence the stress testing will be needed frequently during the tenure of the application services.

2.6.1 Conventional stress testing

Commercially available load testing software, for instance *HP Load Runner* (HP, 2007), *Rational Performance Tester* (IBM, 2013c) and *Microsoft Visual Studio Ultimate* (Microsoft, 2013c) are capable of serving the industrial application performance verification need. Undeniably, these load testing software are essential for the operational continuity of the applications. However the performance modeling using these tools are time consuming, and in most cases, not suitable for applications that have gone live, as there are very few opportunities for lengthy outages. Figure 2.56 displays the components of the *HP Load Runner* utility. The load testing conducted using this tool can last from several hours to a few days, depending on the intensity of the tests. As such, there is great value in producing load testing mechanisms that can shorten the testing time, as well as the effort needed to compile the test cases.

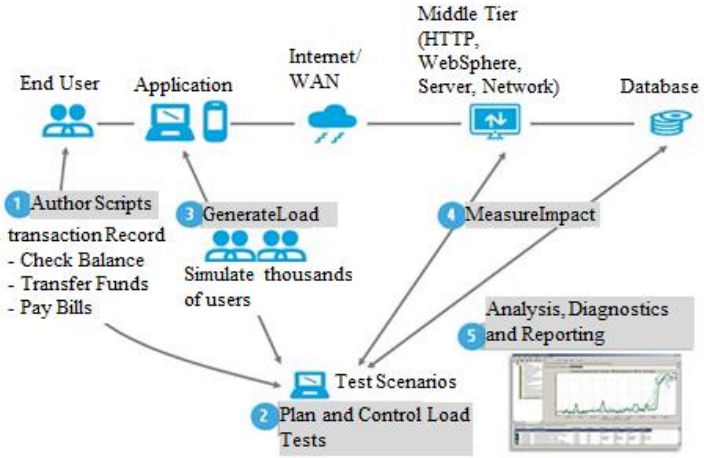


Figure 2.56: *HP Loadrunner* Components. Full-fledged load testing together with comprehensive analysis can be delivered by the utility. Adapted from (HP, 2013a)

A typical load testing constitutes of 3 phases:

1) Ramping up phase – During this time the test is building up the loads to reach the desired workload level.

2) Steady state phase – in this phase, the workload reaches the designated level. Performance metrics are gathered, and the system is monitored if it can sustain the workload for *x* amount of time.

3) Ramping down phase – the load generator gradually reduces the load to the system, until the machine is completely idle with no more load injection.

Thakkar et al. (2008) detailed out the challenges of utilizing the complex load testing tools, and proposed a method to shorten the testing timeframe. The outlined challenges are:

1) Large number of test cases needs to be prepared and executed to cover all possible workloads.

2) Limited allowable time for testing. Particularly for mission critical applications, lengthy load testing is almost impossible to be conducted in production instance.

3) Error in creating the test cases. As test cases are manually created, there are chances of faulty test cases which lead to re-execution that wastes precious production time.

The authors envisaged that an ideal performance model is capable of predicting the resource condition with variable loads. Such vision is illustrated in figure 2.57. With such capability, the amount of performance tests can be reduced significantly. The hurdle to overcome in this case is to discover the most suitable workload samples, which can delineate the complete set of workloads during production mode.

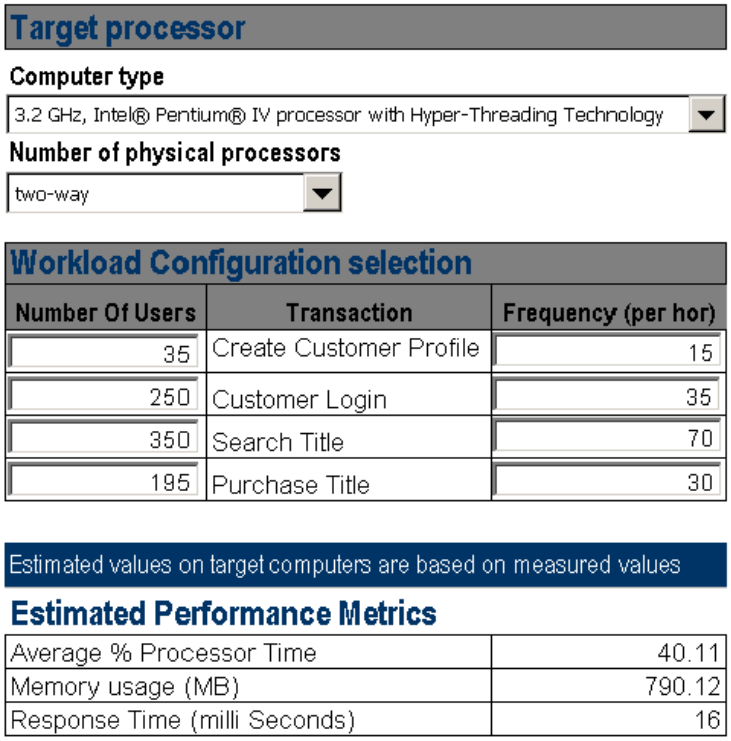


Figure 2.57: Capacity calculator. The idea is to perform load testing on a particular sample of workload, which is representable for all the workloads. Adapted from (Thakkar et al., 2008)

The authors proposed static and dynamic test reduction to try to arrive at the ideal workload samples. The static test reduction requires strong knowledge regarding the functional transactions, where transactions that can complete within short timeframe with less expected executions from end users are filtered out. In addition, similar transactions can be moderated, by including only a subset of the transactions in the representative workload. Dynamic test reduction involves the identification of large and resource exhaustive workloads during the initial test. These transactions are

subsequently ranked by their impact level to the system performance. Depending on the size of the eventual desired workload sample, the load testing will choose only certain percentage of the ranked workload for execution.

The authors also discovered that the resource utilization metric oscillates quite substantially, which makes the harvesting of accurate performance metrics more difficult. The same observation is noted in the experiments which are carried out to bolster the proposals in this thesis. The above load reduction methods are practical, but they do not eliminate the requirement to manually create the test cases. Furthermore, even with such reduction, it will still require few hours of testing time. The proposed stress-testing mechanism in this thesis strives to improve these shortcomings, by setting up the stress-testing scenario in the VM within 15 minutes in general, which corresponds to the steady state phase mentioned above. Moreover, test cases are not needed, but the proposal will mimic as close as possible to the actual scenario in the database as though real transactions are executed in the database.

For the goal of prohibiting access to sensitive data, in servicing environments with stringent data protection, synthetic workloads often need to be fabricated for load testing purpose. Krishnamurthy et al (2006) studied on the creation of synthetic workloads with the intention to match these workloads to the real workloads. A collection of *sessionlets*, with each *sessionlet* represents a sequence of requests from historical real workload traces, is served as input for the construction of the synthetic workloads. These *sessionlets* are built from TPC-W benchmark queries and data which are suitable in this case as web-based applications are in focus. The request length is taken as the criterion to match the real workloads. These *sessionlets* are matched to the desired workload by minimizing their request length difference by employing *linear programming* (Vandenberghe et al., 2002) method iteratively on the workload traces. Another discovery of their approach is that they considered the inter-request

dependencies, where the experimented test cases yield lower hypothetical resource utilization compared to the real situation, hence renders the inaccurate identification of the limit of resource threshold in the host. Thus, the conventional workload partitioning which utilizes Markov Chain model is deemed unsuitable for this reason. The authors implied that the Markov Chain model is rigid in relating the 'states' between the workload partitions hence it is not suitable for mixed and ever-changing workloads. Casale et al (2009) continued the work and experimented using TPC-W benchmark queries and data, to simulate burstiness in workloads, and deliver a mechanism to size resource adequacy in virtualized environment. In this context, CPU time is taken as measurement criterion in this burstiness characteristic. The authors found out that by considering the burstiness criteria, the discovered resource threshold is generally about 30% higher than the threshold value generated by experiments without burstiness consideration.

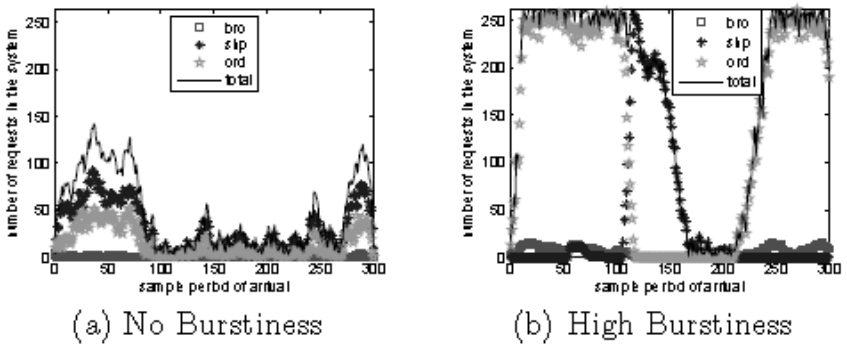


Figure 2.58: CPU utilization on the web and database servers from sampling of load testing carried out on a hypothetical online store. The linear correlation between the load and CPU utilization at the beginning correspond to the observation in the experiments in this thesis. Adapted from (Casale et al., 2009)

In this thesis's proposal on constructing light-weighted stress testing scenarios, the burstiness of particular workload can be accounted by incorporating this element into the representative workload that depicts the full workload in the VM. However, such incorporation is subjected to the SLA of the application offering, as burstiness might be an undesired outlier that does not need to be accounted for. Instead in most cases, it

needs to be gotten rid of as it wastes resources in the VM as the tenure of application service offerings is predominantly serving workloads which are processed in normal fashion. Hence, to account for such criteria in workload partitioning is quite subjective.

In systems which do not permit lengthy outages, a full-blown load testing scenario is not suitable. In view of this, Barna et al. (2011) proposed a framework to explore the real workload space, and discovered the points which cause bad performance in the software and hardware components in the hosting environment. Subsequently load testing is conducted by taking these workload mixes as the stress vectors and starting point for the stress cases. Subsequently these cases are exploited to stress the host to the breakpoint for transactions' response time verification. The interesting proposal in this paper is that the authors strived to uncover the different 'stress vectors' in the workloads, which correspond to a class of similar transactions that can be grouped together. Each stress vector is further divided into few segments, based on number of users or response time during the stress test. With this information, these stress vectors can be utilized to predict and plan for additional resources for expansion of customer base. Similar to the experiments carried out in this thesis, the CPU parameter is taken as measurement to represent the host's resource adequacy. In most real production systems, as the CPU is the most expensive component, the resource constraint will normally happen on the CPU resource. Nevertheless, the constraining factor can happen on other hardware component, producing the scenario as in figure 2.59. In such situation, particularly in virtualized cloud hosts, the usual remediation is to fix this constraint until it does not post as the break point to the systems. As such, the CPU will soon become the constraint which is desired, as it is more cost effective to utilize the CPU cycles as much as possible.

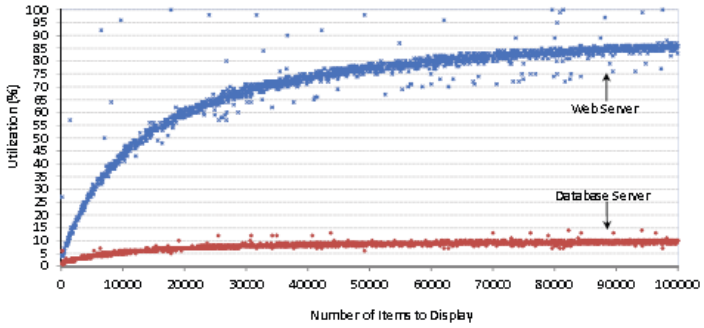


Figure 2.59: CPU utilization on the web and database servers from sampling of load testing carried out on a hypothetical online store. The linear correlation between the load and CPU utilization at the beginning correspond to the observation in the experiments in this thesis. Adapted from (Barna et al., 2011)

2.6.2 I/O parameter

I/O parameter is scrutinized in this section in creating the stress-testing scenario in the VM. The perception, or rather the myth regarding database hosting in cloud in this sense, is that cloud is not conducive in processing database transactions that involve large physical I/O. Ghoshal et al. (2011) surveyed the I/O throughput in local clusters and Public Cloud. To discover the I/O performance in local clusters, experiments are carried out in readily available large clusters called *Magellan* from NERSC (NERSC, 2013). To simulate the Public Cloud I/O performance, Amazon EC2 instances are employed.

Comparisons are made between filesystems in these private and public virtualized environments, with analysis on the differences. The I/O performance is analyzed based on benchmarked experiments from Interleaved or Random (IOR) and *Timed Benchmark*. IOR generates the I/O usage patterns based on various interfaces. The block size and transfer size for reads and writes are set to 100G and 2M respectively. The *Timed Benchmark* also utilized the same block-size and transfer-size. In addition, the results are continuously collected over a period of time with sample collection in certain frequency interval.

The summary of the causes that induce the I/O variation are:

1) Direct and buffered I/O. Such difference in I/O is only observed in the NERSC clusters. Amazon EC2 is not having the I/O buffered as according to the authors, and only direct I/O is configured in this Public Cloud. In the NERSC clusters, experiments are conducted on 3 types of filesystems, as depicted in figure 2.60. *Global Scratch* is a high speed global shared filesystem that often deployed in *High Performance Computing* (HPC) applications, which has a peak performance of 15GB/s. Local disks are attached to the nodes, without needing network connection between the computing and data nodes. Elastic Block Storage (EBS) (Wolloch, 2013) volumes are the network attached storage (external disks) storage arrays which are often used in traditional data center. The peak throughput for this kind of storage varies, depending on the type of network connection between the data and computing nodes. The properly configured EBS volumes should have peak performance between 500 to 2000 Mbps. From the graph, it is observed that cached I/O provides much higher throughput for *Global Scratch* filesystem. This condition can be devoted to the high speed network connecting the filesystems and the computing nodes, as it is not posing as a constraint; hence data can flow freely from the cache in the storage node to the computing node. In the local disks and EBS shared-storage, the throughput from buffered I/O is slightly higher than direct I/O. The authors concluded that in this case, the buffered data is having minimal or no effect on virtualized resources. For the EBS storage in Amazon EC2, it is observed that the Amazon EC2 does not provide for buffered I/O capability.

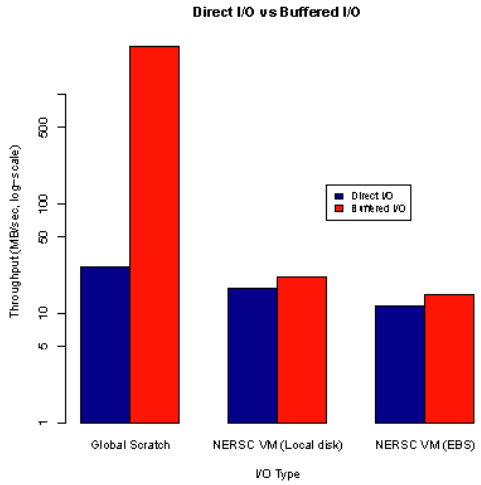


Figure 2.60: Comparison of I/O performance between direct and buffered I/O in virtualized clusters in Private Cloud. *Global Scratch* is optimized to provide high throughput when the data is cached. The local disk and EBS shared-storage are not showing significant difference between the 2 types of I/O. Adapted from (Ghoshal et al., 2011)

2) The effect of instance type in Public Cloud - Amazon EC2 instance type architecture is configured in such a way that for larger instance, the I/O throughput is higher. Hence for clients that pay more to acquire bigger cloud instances from Amazon, they are also provided with better I/O throughput rate.

3) Regional effect – Amazon EC2 instances are observed to perform better in certain region in the United States. Such observation can be concluded for other cloud providers where different data centers in various locations can have different configurations, which result in dissimilar performance throughput.

Figure 2.61 is the result from the IOR tests. Surprisingly, the Amazon EC2 outperforms the NERSC clusters in the I/O throughput. Generally local disks perform better than EBS shared-storage, however the difference is not in order of magnitude as initially perceived.

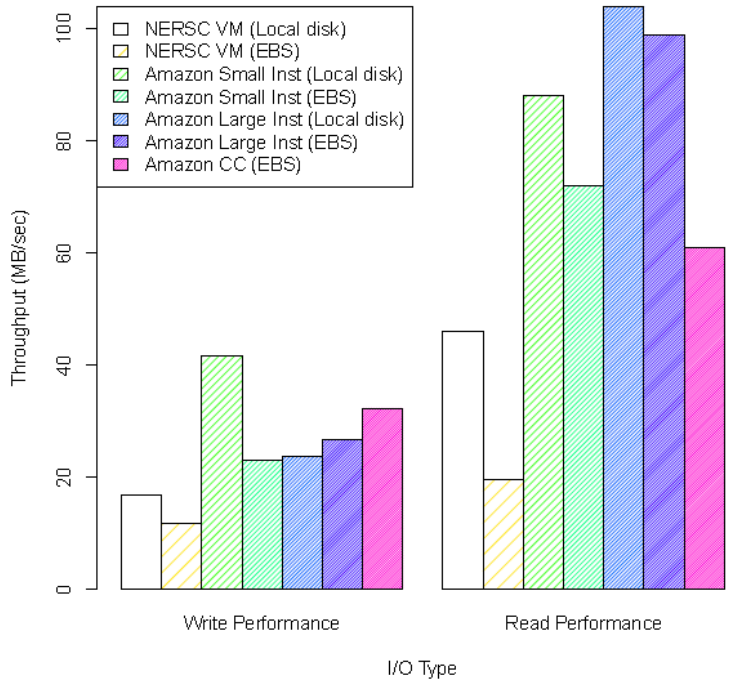
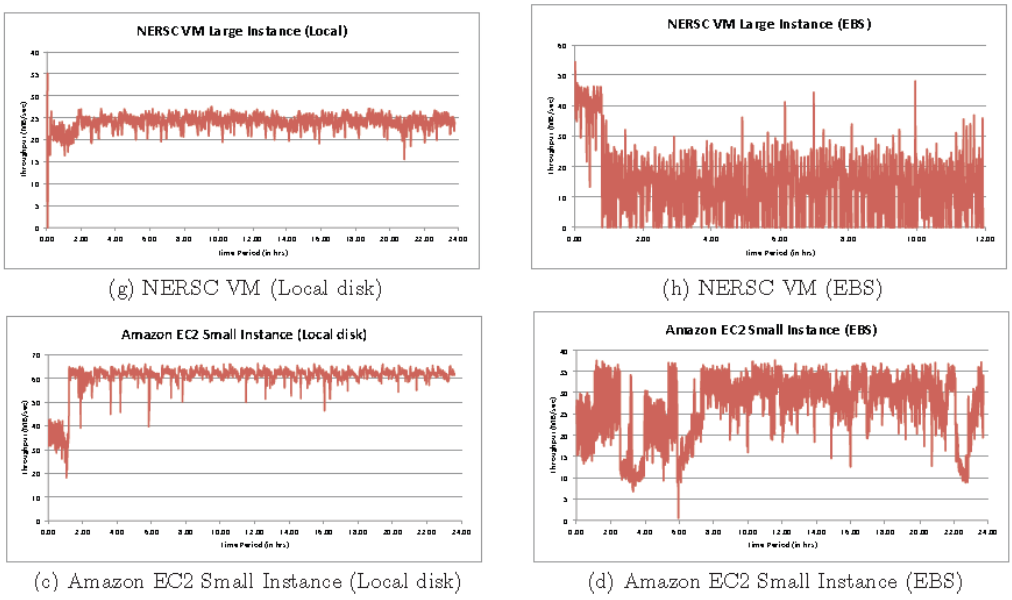


Figure 2.61: Comparison of I/O performance between direct and buffered I/O in virtualized clusters in Private Cloud. *Global Scratch* is optimized to provide high throughput when the data is cached. The local disk and EBS shared-storage are not showing significant difference between the 2 types of I/O. Adapted from (Ghoshal et al., 2011)

It is also observed that the filesystems on local disks are having a more consistent throughout as compared to shared filesystems, both in private and Public Cloud. Figure 2.62 depicts such scenario. The experiments were carried out in a continuous 24-hour timeframe.



(g) NERSC VM (Local disk)

(h) NERSC VM (EBS)

(c) Amazon EC2 Small Instance (Local disk)

(d) Amazon EC2 Small Instance (EBS)

Figure 2.62: Throughput as a function of time. Local disk is observed to perform more consistently, as it does not depend on the network to deliver the data. Adapted from (Ghoshal et al., 2011)

Table 2.9: Summary of studied researches with critical comment on sub-theme 'conventional stress-testing and I/O parameter'.

Scholars	Contribution	Comment
(Thakkar et al., 2008)	Detailed out the challenges of utilizing the complex load testing tools, and proposed a method to shorten the testing timeframe.	The authors noted that the resource utilization metric oscillates quite substantially, which is the same observation as in the experiments in this thesis.
(Krishnamurthy et al., 2006)	Studied on the creation of synthetic workloads with the intention to match these workloads to the real workloads.	The authors utilized TPC-W benchmark in their proof-of-concept experiments. Similar approach is taken in this thesis.
(Barna et al., 2011)	Proposed a framework to explore the real workload space, and discover the points which cause bad performance in the software and hardware components in the hosting environment.	The interesting proposal in this paper is that the authors strived to uncover the different 'stress vectors' in the workloads, which correspond to a class of similar transactions that can be grouped together. These stress vectors can be utilized to plan for resource requirement.
(Ghoshal et al., 2011)	Surveyed the I/O throughput in local clusters and Public Cloud.	The study on the I/O characteristic is essential in the research proposal to produce the affirmation model.

2.7 Summary and discussion

Computing hardware resource management is very critical to many organizations, in order for them to stay relevant and profitable in their respective business domains. To survive in the stiff competitive market, the cost of computing needs to be optimized by employing the latest technology that allows for significant reduction in workforce and infrastructure spending. Cloud computing is such paradigm that promises great saving in capital and operational cost. It provides the platform for enterprises to pay only for the resources that they use. Couple this with the flexibility in the computing resource allocation, the consumers needlessly procure large and expensive servers to cater for the predicted future computing requirement. The magnitude of IT saving resulted from this is extremely significant. For instance, instead of spending few millions of dollars in procuring top-notch enterprise-class Unix servers to host the mission critical ERP databases, with the cloud clusters, the hardware investment can be as little as less than a hundred thousand to deliver infrastructure that is capable of running the same databases. Amid the skepticism in the cloud privacy and security, it is inevitable that the cloud will flourish and become the predominant standard in all IT infrastructure hosting.

Hence, for the virtualized cloud to deliver its full potential, resource management play the most important role. Without proper resource utilization *monitoring*, suboptimal hardware resource scaling results in considerable overhead in the system which discounts the purpose of elastic scalability benefit. If the hardware is not scrutinized on its performance, substandard delivery of transactions' executions result in breach of SLA, and wastage in the computing resources. Hence, both the resource utilization *monitoring* and *optimization* topics are studied in length in this research. Subsequently, resource utilization *affirmation* which concerns with the verification of the transactions provides a platform for swift stress-testing that is different from cumbersome

conventional load testing. With the convincing delivery of these 3 themes, cloud adoption will surely be more efficient and profitable.

In this chapter, the surveys are conducted by adhering strongly to these 3 proposed themes. The sub-themes from each are elaborated in length. In the *monitoring* scheme, the surveyed sub-topics are:

1. Monitoring models deployed in current wide industry as well as those envisaged by scholars, which have great value in propelling the cloud paradigm.
2. Resource scaling technology, which tightly coupled with the monitoring models. The improvement in this mechanism strives to boost the level of efficiency in cloud offering.
3. Statistical modeling and benchmarking intend to characterize a greater visibility on transactions that occurs in the VM.
4. Workload characterization has similar aim as the studies in Statistical modeling and benchmarking. The target workloads are from both the real and hypothetical scenarios.

In the *optimization* scheme, following topics are scrutinized:

1. Fault analysis and failure prediction in the computing hardware. Such studies ensure that sub-optimal performance in the computing hosts is avoided.
2. Task scheduling is surveyed to discover the current and potential methods to ensure computing tasks can complete without constraint in resources.
3. Resource scheduling is reviewed next. Its objective is also to ensure that resource constraint does not occur in the computing hosts. The difference is that it targets the optimization at the hardware level instead.

4. Resource brokering. It has similar objective as resource and task scheduling. However the target optimization area is at the cloud providers and consumers segment.

After the above 2 schemes are studied, the *affirmation* scheme targets following topics to provide for its verification objective on application transactions:

1. Stress testing. The conventional methods are reviewed, together with scholarly proposals which are impactful in the evolution of this technology.
2. I/O influence on workloads is analyzed as the experiments that serve the *affirmation* scheme largely utilize the I/O component in the VM as input to produce the desired output.

Some of the studied literatures are not directly applicable to the prototypes developed in this thesis. Nevertheless, their influence on the outcome of this research is significant and relevant. From the detailed scrutiny on these literature reviews, the general trend of resource management in the industry, particularly in cloud platform is perceived. Subsequently the generated proposals are originated from these roots. Besides reviewing these literatures, foreseeable future developments are also indicated in this chapter.

3. RESEARCH METHODOLOGY

3.1 Introduction

There are 3 themes in this research. The first concerns the resource utilization *monitoring* in the cloud virtualized environment. Subsequently, resource utilization *optimization* is targeted. Following these 2, resource utilization *affirmation* is scrutinized.

Firstly, the first topic is important as an effective monitoring system is required to provide clear visibility on the resource state in the VM, for resource planning and scaling purposes. It requires the metadata of the real workloads, particularly on the SQL processing time, to be mapped to the corresponding CPU run queue size, to produce the resource state depiction. The relationship between the SQL processing time and CPU run queue size is assumed linear in nature, in order for this monitoring mechanism to deliver its objective. The workloads in focus do not necessarily filtered; hence the burstiness condition is also established by the monitoring mechanism, which allows for resource planning in case burstiness situation in resource utilization is included in SLA calculation. The research work here has been published in (C. H. Tan & Teh, 2013a).

Secondly, optimization in the hardware is targeted, as it is crucial to safeguard the hardware performance in the VM, in order for the computing power to be reflective of the true capability of the hardware, as well as to hinder any potential failure in the near future. To construct the mechanism to examine the hardware state, TPC-H data and queries are employed to load the VM gradually and steadily to the run queue threshold. Once convincing baselines are obtained, they are recorded as benchmark. Subsequently, the same TPC-H queries are run periodically, and the results are plotted into linear regression graphs. The gradients and y-intercepts of these graphs are compared to the

baselines, to arrive at the conclusion if there is any concern with the hardware performance in the VM.

Thirdly, a mechanism is proposed to create stress-testing scenarios in the VM, in order for the SLA-bound transactions to be verified. In this case, the proposed mechanism is similar to the conventional stress testing utility. However the proposed model requires much shorter timeframe to arrive at the steady state stressed level. TPC-H queries and data are employed to load up the VM. In doing this, the memory reads/s of the real workload is matched with the memory reads/s of particular TPC-H query or queries. Once this match is found, the TPC-H query or queries are executed iteratively in the VM to reach a steady run queue level, for subsequent verification of SLA-bound transactions. It is to note here that the caveat for the successful implementation of this model is that the I/O must be predominantly performing memory reads in the database. The memory writes, physical reads and physical writes are assumed not significantly influence the real workload. Due to this presumption, the representation of the workloads is generally applicable for OLTP transactions that do not induce significant I/O write to the storage subsystem. This scenario is common in properly architected production environments. As the memory modules are getting cheaper, this motivates the utilization of memory reads/s parameter as the primary variable in the database together with the run queue size, for the construction of the stress-testing environment. The TPC-H queries are also utilized to discover new CPU run queue value when the hardware change is taking place in the VM. The research works for the second and third themes have been published in (C. H. Tan & Teh, 2013b, 2013c).

In order for the above 3 themes to deliver their objectives, the resource utilization must only be constrained at the processor level in the operating system. Such scenario is also common in most hosting architecture, as CPU power is generally more expensive than

any other component in the hosts. With this sanctioned, CPU run queue can be comfortably applied in the linear relationship envisaged in the proposed mechanisms.

3.2 Approaches to research

In producing the resource management proposals, the caution is to avoid access to real data, in order to protect the data privacy. Hence, metadata from real workload is mined to produce the *monitoring* model. Thereafter, TPC-H is employed to formulate synthetic workloads in the *optimization* and *affirmation* models. TPC-C was considered before; however the queries in TPC-C are lighter, hence harder to stress the VM to the resource limit compared to TPC-H queries. Hence even though TPC-C is established as a common industrial standard, it is not utilized here. However it is considered for future works in replacing performance data widely employed in this thesis with total-transaction-capable measurement in particular VM.

The conventional silo monitoring of the resource condition in the VM is assumed inadequate. This research combines the database parameters with the OS parameter to deliver its resource management objectives. From the database end, the SQL processing time is taken as the primary variable in the first 2 themes. In the first theme, this parameter is further segregated into 2 types. The first one is called DB CPU Time (DCT). It is the processing duration that is required solely by the database. The second is named SQL Elapsed Time (SET). This duration constitutes of all the timing parameters involved in delivering the result of the query. In other words, DCT is a subset of SET. The magnitude of difference between DCT and SET is taken as the barometric gauge to determine the resource adequacy in the VM for database operations. The proposal in the second theme takes only SET as reference when database parameter is combined with the OS parameter in the linear regression analysis. Then, the suggested stress-testing scenario built for the third theme utilizes the memory

reads/s parameter in the database. From the operating system perspective, all the 3 themes are employing CPU run queue size as the parameter from OS to be combined with the database variables.

From statistical modeling standpoint, *linear regression analysis* is employed in the first 2 themes to depict the resource utilization condition in the VM. It is to note that any data point beyond the linear plots is not to be considered in the construction of the *monitoring* and *optimization* mechanisms. Further to this *linear regression analysis* technique, *machine learning* is applied to periodically learn the behavior of the hardware system. In the third theme, *linear programming* and *simplex method* are envisaged to discover the TPC-H query that has the closest match to the I/O throughput and CPU run queue combination.

3.2.1 Definition of research objectives

The interest in the resource utilization *monitoring* is to provide a clear visibility of resource state in the VM. Such information is subsequently relayed for resource *planning*, or *scaling* purpose. In many of the surveyed studies, this topic is part of many proposed resource management methodologies. It serves as the critical pre-requisite for many resource or task scheduling proposals. The success in deployment of efficient and autonomous monitoring system is the goal in this research.

The definition for resource utilization *optimization* is broader. Many scholars' researches focus on optimizing the resource usage in the hosts, by attempting to increase the percentile of utilization in the VM, via the popular task and resource scheduling models. There are also suggestions to optimize from the monetary investment perspective, by proposing supply and demand on hardware resources via Auction-based algorithm. In this research, the optimization is viewed from the perspective of hardware performance. This aspect is critical in complementing all the other optimization

approaches. Only with proper delivery of hardware performance, the other avenues can become meaningful.

Resource utilization *affirmation* concerns with creating a stress-testing scenario for verification of SLA-bound transactions. The objective for the proposal is to augment the application service offering, where the transactions can be tested and the associated response time verified. In a way, its function is similar to conventional load testing; however the duration needed to create the stress-testing scenario is much shorter. Hence, it should be viewed as a light-weighted version of load testing utility that complement the conventional load testing tools, where comprehensive but cumbersome load testing is undesired in some situations. Furthermore, it also serves the objective to discover new value of CPU run queue threshold value in case there is change in the hardware configuration in the VM.

3.2.2 Proposed models

3.2.2.1 *Monitoring* model

The model envisaged for resource utilization *monitoring* theme is depicted in figure 3.1. The data points from the test will produce the 2 linear plots. S' denotes the DB CPU Time (DCT), which is a subset of S, which is the SQL Elapsed Time (SET). SET involves all the processing time to complete the SQL, from database as well as operating system perspectives. C''_T is the 100 percentile value of the run queue threshold in the particular VM. Data points beyond this threshold are not guarantee to conform to the linear relationship, hence are not serving the interest of the proposal. C'_T and C_T are the 80 and 90 percentile values, and a 5% zone is visualized between these 2 edges. ΔS is the value different between the S and S'. To conceptualize these parameters, take figure 3.1 as the ideal test result from the real workload. In this case, at point C''_T, $S' = 2S$. This can be deciphered as to process the workload in the VM at

point C''_T, the database needs only S amount of time to process the workload, but due to constraints internal or external to the database, the result of the workload can only be returned after spending additional 100% of time in the VM. This 100% additional time is taken as the near-threshold for workload processing. From this explanation, it can be known that C''_T is not determined in prior, but obtained its value from ΔS. From C''_T, C_T and C'_T are deduced. Hence, this monitoring system is also capable of inducing the resource threshold in the VM.

The real insightful information to determine if the resource adequacy in the VM is characterized by the data points in the 5% zone. Now, when 5% of the total data points fall into this zone, the VM is deemed saturated in the utilization of its resources. The burstiness condition can also be delineated by narrowing the 5% zone in the model.

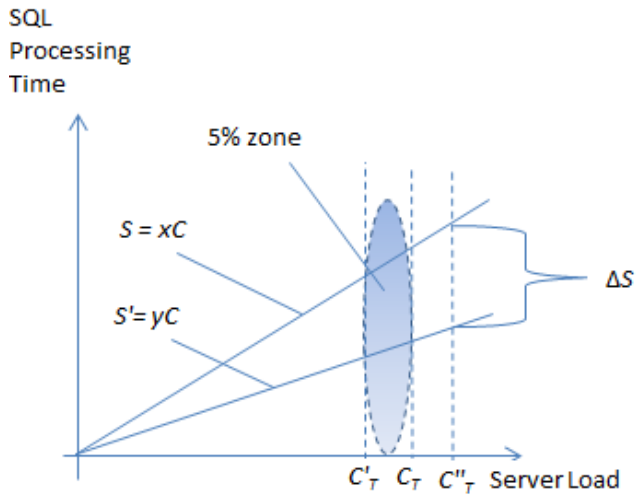


Figure 3.1: Resource utilization *monitoring* model. It is capable of providing clear visibility on resource usage, as well as determining the run queue threshold in the VM.

3.2.2.2 *Optimization* model

The resource utilization *optimization* model is as illustrated in figure 3.2. In this diagram, the TPC-H query #8 is utilized to steadily load up the VM, with the corresponding CPU run queue size recorded at each interval. Again, any data point beyond the resource threshold, a value that can be obtained from the first and third

research themes, is discarded. This is because the linear relationship needs to be adhered to in order for this model to deliver its objective. From this diagram, it can be observed that that there are 2 directly obtained values from the linear plot: the gradient and y-intercept of the linear graph. If the hardware performance is optimal, periodic tests conducted will yield almost identical values for these 2 parameters. Besides these, the correlation coefficient of the linear plot must also be taken into account during the analysis, as it represents the relevance between the best-fitted linear plot and the testing data points. There are 2 phases in this model. Firstly, initial test is carried out to register the baseline values for the gradient and y-intercept. Subsequently during steady state operations, periodic tests are executed to produce the linear plots, and the associated gradient and y-intercept are matched to the baseline. If close match is found, the hardware performance is deemed consistent and identical to the initial state. The tests are to be carried out during maintenance window, where there is not any noise in the database and operating system that hinder the accuracy of the result.

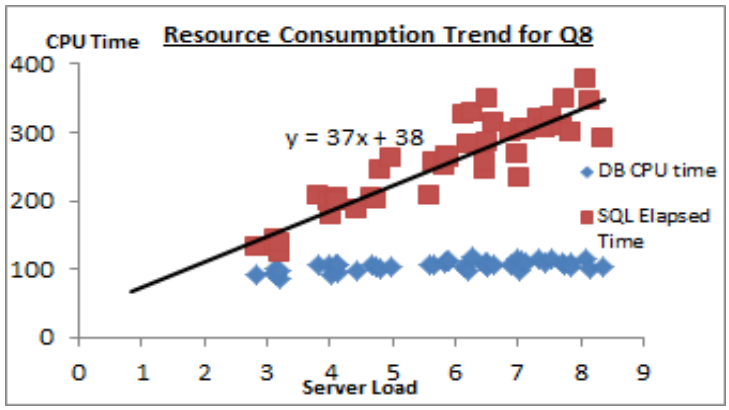


Figure 3.2: Resource utilization *optimization* model. The linear graph is capable of characterizing the hardware performance in the VM.

3.2.2.3 *Affirmation* model

The modeling of resource utilization *affirmation* is similar to the *optimization* model, where benchmarks are established before the actual tests are conducted. As depicted by figure 3.3, during the initial benchmarking effort, TPC-H queries are run against the

VM, by maintaining the CPU run queue at a level which is close to the threshold value in the VM. When the test is stabilized, the values for memory reads/s that correspond to each query are recorded in an array as below:

A[Query][memory reads/s] = {Q16, Q9, Q13, Q21, Q3, Q7, Q5, Q11, Q1, Q22, Q4, Q10, Q14, Q6, Q18, Q19, Q20, Q12, Q15, Q17, Q2, Q8}, {7000, 11000, 10000, 10000, 11000, 11500, 12500, 12500, 13000, 14000, 14500, 14500, 15000, 16000, 16000, 16500, 16500, 17000, 17000, 19000, 22500, 16000};

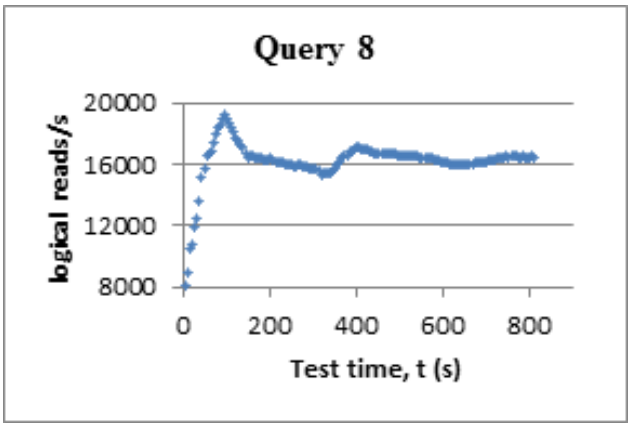


Figure 3.3: Resource utilization *affirmation* model. Initially a set of benchmark data is needed, by experimenting on various TPC-H queries. The goal is to discover the corresponding memory reads/s for each TPC-H query.

Subsequently, when the stress-testing scenario is needed, the real workload traces are analyzed, and the corresponding memory reads/s in the workload is retrieved. This value is matched to the array *A*. When identical memory reads/s value is found, the associated TPC-H query is employed to load the VM, which creates a stressed condition similar to the effect of the workload in the VM. It is to note that the workload traces need to be obtained during steady state, with the CPU run queue size to stay constant at the level similar to the benchmarking stage of the TPC-H queries.

Besides the stress testing scenario creation, the TPC-H queries are also used to discover the run queue threshold in the VM in this theme.

3.2.3 System design

3.2.3.1 The *monitoring* scheme

The steps towards the build of the *monitoring* scheme are depicted in figure 3.4. As shown, the monitoring process is a continuous effort that can be performed without incurring downtime to the database operations. It strives to discover the threshold point in the VM, as well as the visibility on the resource adequacy. The result from the mechanism is then fed into resource planning or scaling exercises. The discovery of the threshold point in this case is resulted from the input of metadata from the actual workload; hence it is closer to the user experience as compared to the threshold discovery method in the *affirmation* scheme which will be discussed in a while. However it is not as proactive in comparing to the proposal in the *affirmation* model as the *monitoring* model needs real workload to be executed before the new threshold can be established.

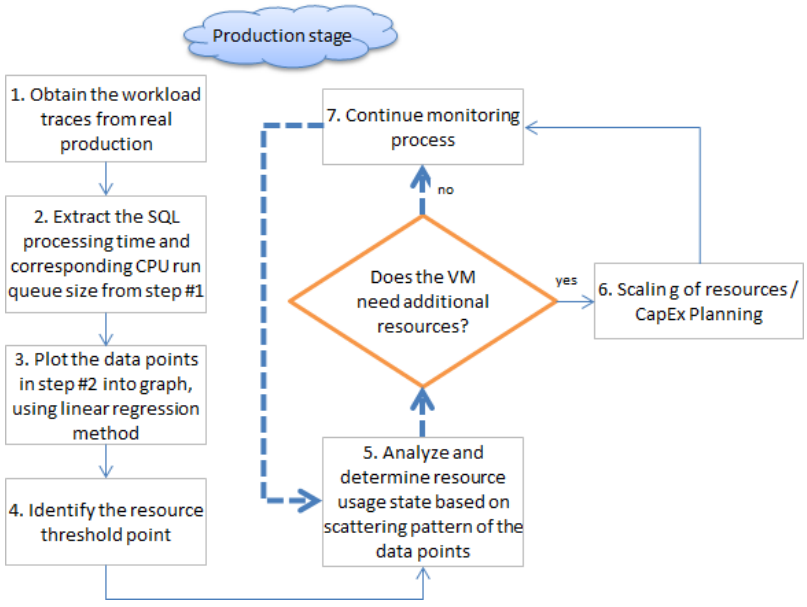


Figure 3.4: The creation flow in the resource utilization *monitoring*. The eventual objective is to provide for clear visibility of resource usage in the VM, to serve the planning or scaling purposes. Dotted lines depict the monitoring and analysis tracks.

3.2.3.2 The *optimization* scheme

As illustrated in figure 3.5, the *optimization* scheme is built from 2 stages. First, the TPC-H queries are executed in the VM, to produce the baseline reference. Subsequently, the same queries are executed in the VM during production stage, and the test results are compared to the baselines to determine the hardware consistency and optimality. During the second phase when the VM is running in the production mode, the tests are carried out periodically to probe the hardware condition.

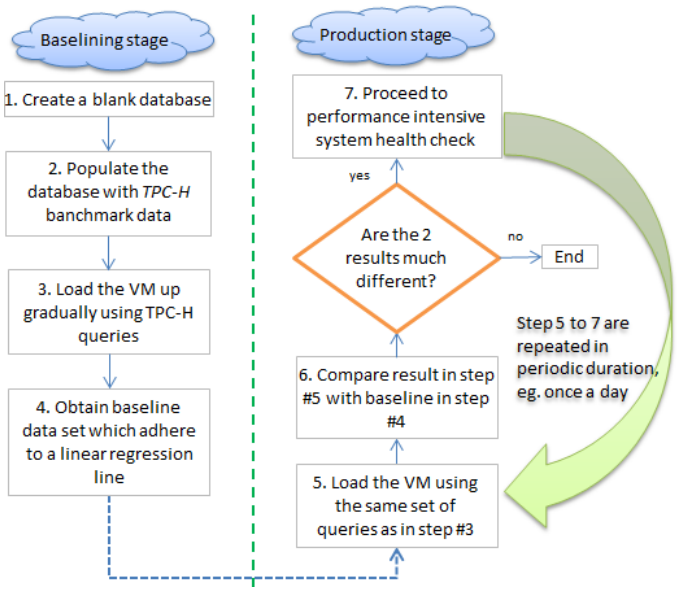


Figure 3.5: The creation flow in the resource utilization *optimization*. There are 2 stages in the model. The initial stage creates the baselines for subsequent reference in the production stage.

3.2.3.3 The *affirmation* scheme

The benchmarking phase in figure 3.6 produces an array of value pairs. The pairs consist of TPC-H queries and associated memory reads/s. These data is obtained from the iterative execution of the queries by maintaining the CPU run queue size at certain level, which is close to the resource threshold in the VM. The creation of the stress-testing scenario during production phase is carried out during maintenance window, as the VM should be free from noises before the environment can be stress-tested. In order to mimic the real environment, the TPC-H query chosen to load the VM is matching the

same memory reads/s value of the real workload, at CPU run queue level that is similar to the benchmarks. When the stress- testing scenario is stabilized in the VM after a brief duration of time, SLA-bound transactions can be executed to have their response time verified.

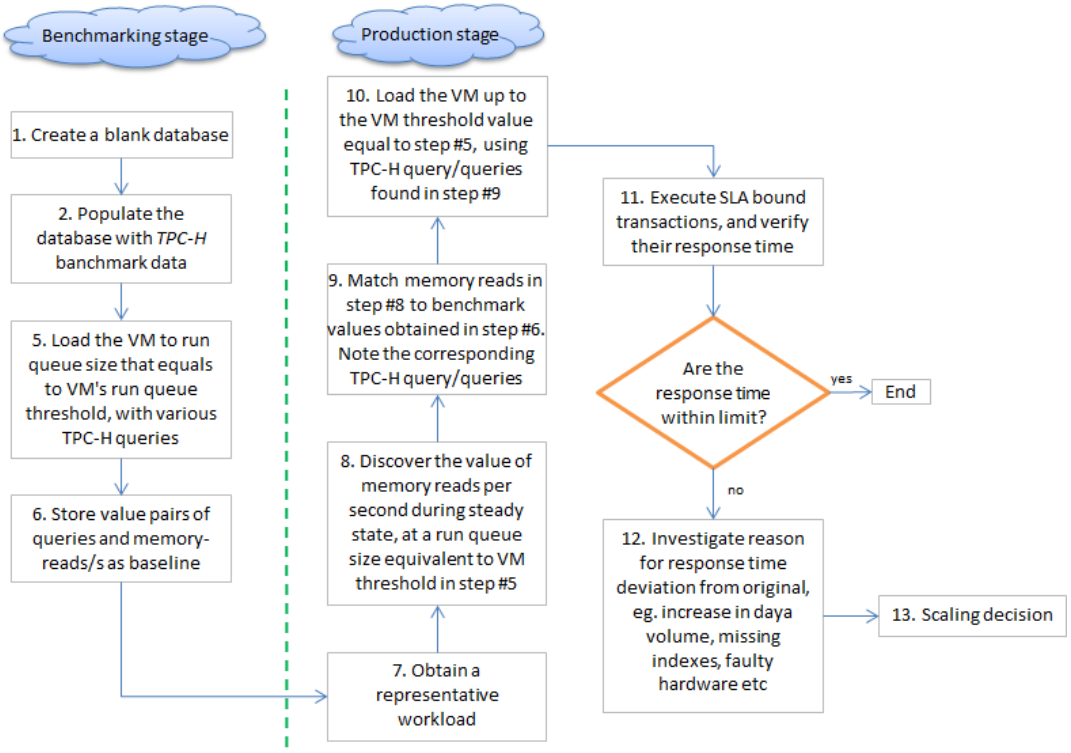


Figure 3.6: The creation flow in the resource utilization *affirmation.* The objective is to build an instant of stress-testing scenario for transactions' response time verification. Such verification result will subsequently be fed as input for scaling decision.

The next purpose of the *affirmation* scheme is to create a platform to discover the new CPU run queue threshold when the VM undergoes hardware change. Such method is deemed more proactive than the threshold discovery in the *monitoring* scheme, as this model does not require metadata from real data. Hence it can be put in use as soon as the hardware change takes place. Figure 3.7 illustrated the steps towards achieving the setup. The TPC-H queries are executed gradually and discretely when loading up the VM. For such mechanism to take place, it has to be run during maintenance window to avoid noises from the database operations and operating system.

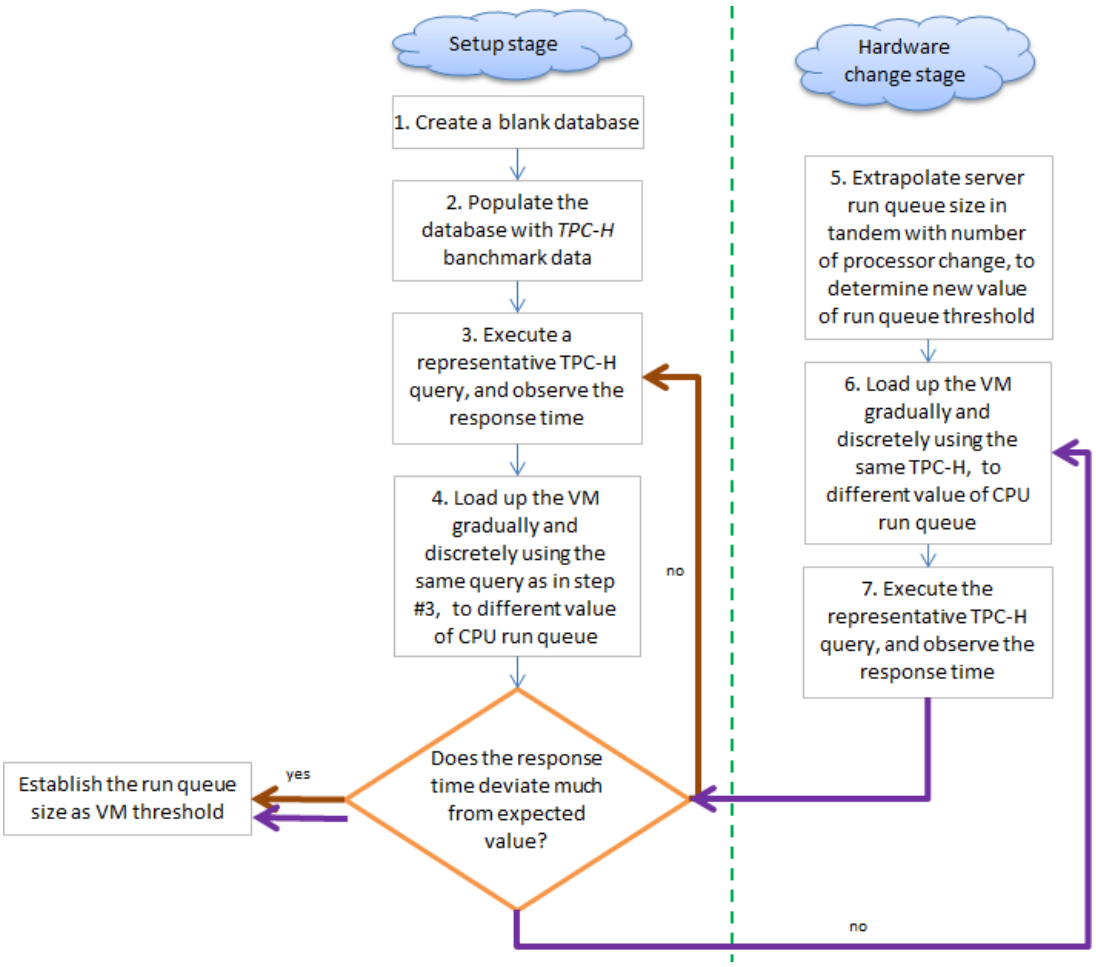


Figure 3.7: The second creation flow in the resource utilization *affirmation*. The goal is to discover the CPU run queue threshold based on TPC-H query's response time. The colored arrows depict the iteration of the tests.

3.2.4 Analysis of methods

3.2.4.1 *Monitoring* model

The effectiveness in the resource utilization *monitoring* model very much depends on the representability of the workload which is used as input. A general Healthcare application, for instance a system that allows query on patients' medical records, typically will have the access and usage pattern recorded in a week. In this case, the metadata collection for 1 week is sufficient to characterize the resource usage in the VM. However certain applications, for example the performance management portal in Human Resource Management System (HRMS) becomes active only during certain

duration in a year. In such cases, the workload needs to be captured in that particular timeframe in order for the proposed *monitoring* model to be effective.

As the resource adequacy is determined by the accumulation of data points in the 5% zone, such monitoring method can also account for burstiness in the workload. In this case, the zone's position can be adjusted and narrowed. Hence the monitoring model is capable of catering for SLA that requires accommodation on either the normal or burstly transactions.

The data collection and analysis efforts do not incur downtime on the database; hence they can be performed at any instance and frequency. Furthermore, from the experiment, it is observed that the metadata collection on the identified workload does not induce high overheads. It is to note that the analysis on the *monitoring* model valid only at the linear portion of the graph. Any data points beyond the CPU run queue threshold are not serving the interest in this proposal.

In the above explanation of figure 3.1, $\Delta S = S' - S = S$. As explained, S is the processing time required by the database to completely process the workload. It is a subset of S', where S' includes all the waits in the database and VM. As mentioned, for this model to work, the resource constraint must be on the processor component. Hence if $S' = 2S = \Delta S/2$, that means on average, when the SQLs are processed at this run queue level, it will have 1 queue at each of the processors in the VM. Generally this would be the constraint point in the VM as queuing for computing power is often undesired. So as in figure 3.1, the VM resource threshold is fixed at C''_T where the value of $\Delta S = S$ corresponds to.

In producing this diagram, real data is not engaged. Hence the objective to avoid access to private and sensitive data is achieved. Only the metadata of the workload is required

to produce the data points needed in the model. It is also to note that outliers in the data points need to be gotten rid of, so that the 2 linear plots can become more accurate.

3.2.4.2 *Optimization* model

As explained and illustrated in figure 3.5, outage on the database is needed to compute the baseline data. The test for each TPC-H query requires 25 minutes to complete. This duration includes 10-minute stabilization timeframe, and 15 minutes of data collection. The 10-minute initial stabilization wait time is crucial as the SQL processing time and CPU run queue parameters oscillate quite substantially at the beginning of the test. This baselining activity is normally carried out at the beginning of the VM build or immediately after the hardware is provisioned. At any 1 time, 2 baselines will be sufficient for reference by the subsequent testing data in production mode. Each baseline data can consist of only 1 TPC-H query or a combination of the queries.

The baselining activity is required after each instance of hardware change. Subsequent testing data harvesting activity in production phase will also need the same 25-minute outage window. In view of this, such model is not suitable for VM that undergoes aggressive on-demand resource scaling. It is primarily designed for database hosts that experience static and staged resource scaling activity. Similar to the model in *monitoring* scheme, the model is applicable only in the linear correlation in the SQL processing time and CPU run queue plots. The gradient and y-intercept values obtained from the testing data and baselines are compared, and uniformity between them will indicate consistency and optimality in hardware performance. However it is to note that due to the nature of the parameters employed to construct the graphical representation, to obtain exactly the same values between the baselines and testing data are improbable. Hence a close match between them will suffice. As the parameters oscillate quite substantially, the gathered results need also be validated using the correlation

coefficient, *r* in the linear regression analysis. In addition to this, the outliers of the data must be removed and noises blocked during the test to increase the accuracy of the graphical output.

3.2.4.3 *Affirmation* model

Similar to the *optimization* model, a separate timeframe needs to be provisioned in order to gather benchmarking data to be used for the construction of the stress-testing scenario. Each TPC-H query will require 25-minutes time block in order to obtain the consistent memory-reads/s value. A typical benchmark will contain close to 20 tests from different individual or combination of TPC-H queries, so the length of outage window required is quite substantial. Furthermore, such benchmarking effort is also needed whenever the hardware configuration is changed in the VM; hence its design will be more suitable for hosts that do not undergo too elastic and frequent resource scaling. However the benchmarking sequences are automated, hence manual intervention can be avoided.

The future work in this area will involve finding out the possibility of extrapolation to reduce the benchmarking timeline. With this timeframe reduced, it is more practical for deployment in real world applications.

The second proposal in this *affirmation* model involves the discovery of CPU run queue threshold when the hardware configuration is changed in the VM. The setup here is rather straight forward comparatively. However, the output gathering at each staggered run queue level must allow a timeframe for the parameter oscillation to stabilize. The threshold discovery proposal in this section serves the same purpose as in the *monitoring* model, amid different methodology and input values. Here, synthetic workload using TPC-H data and queries are employed, whereas the *monitoring* model is obtaining insightful input from the metadata of the real workload.

3.3 Summary and discussion

This chapter provides a high level explanation on the conducted research, and the proposed outcomes. The research is segregated into 3 themes. The first and third themes have 2 proposals in each scheme; whereas the second theme contains 1 proposal.

The *monitoring* scheme strives to introduce a way to monitor the resource consumption in the VM via metadata harvesting from real representable workload. In addition, it allows for resource threshold to be discovered via real users' experience. The *optimization* scheme proposes a method to examine the consistency, stability and optimality on the hardware performance. Subsequently, the *affirmation* scheme suggests a way to compute a stress-testing scenario that allows for the SLA-bound transactions to have their response time verified. Furthermore, a method to realize the CPU run queue threshold is envisaged in this section.

In all the 3 themes, the real data access is avoided. Hence this achieves the objective of allowing effective database and VM administration without compromising on data privacy and security. The proposals address the shortcoming of the commercially deployed hosting architecture, with the prospect of encouraging cloud adoption by organizations with skepticism on cloud safety, particular on the data security issue.

The next chapter will explain the detailed designs of each scheme.

4. SYSTEM DESIGN

4.1 Introduction

In this chapter, the detailed design of the models is explained. As mentioned in chapter 3, there are 3 themes involved in this research. First theme strives to examine the resource usage state in the VM, by proposing a graphical depiction of resource utilization *monitoring*. The source of input to the mechanism is described in section 4.3.1. Subsequently, the construction of the model is demonstrated in section 4.3.2. The method to remove the outliers is detailed in 4.3.3.

The second theme attempts to produce a mechanism to investigate and probe the hardware condition in the VM, so that computing performance consistency and optimality can be achieved with the set of hardware configuration. Section 4.4.1 describes the setup of the model, by scrutinizing the role of *machine learning* in the model construction. Consequently, section 4.4.2 explains the applicability of the mechanism in production environment.

With the design of the *monitoring* and *optimization* models explained, the third theme dwells on the *affirmation* model. In this section, 2 mechanisms are described; one that creates stress-testing scenarios in the VM, and another one which aims to discover the resource threshold in the VM. Section 4.5.1 illustrates the configuration of the environment. This is followed by the application of the model in production mode in section 4.5.2. Section 4.5.2.1 describes the usage design of the stress-testing scenario, and 4.5.2.2 outlines the threshold discovery method by using the model.

4.2 Overview of proposed models

The monitoring model is based on graphical depiction of the relationship between the SQL processing time and CPU run queue length. The manipulation is performed against

this linear relationship between these parameters. With the linear plots established, the monitoring outcome is produced by the locality of the data points which correspond to the transaction processing condition in the representative workload. There is little resource overhead incurred by the data collection, hence it is ideally applicable throughout the tenure of the application offering. Furthermore, the setup of the model does not require outage from the database operations.

The setup of the *optimization* and *affirmation* models requires a separate database in the VM, which solely host the TPC-H data. However, the TPC-H data can also co-exist with an existing database in the VM. Nevertheless such configuration might produce noises in the environment as it is often difficult to control the end users' activities even outage is scheduled. Hence it is simpler from management perspective to create a separate database for the construction of these models. The TPC-H data and queries are downloaded from Transaction Processing Performance Council website (TPC-H, 2013). The proof-of-concept environment is utilizing VMWare Virtualized Infrastructure which has been elaborated in Chapter 2. In each database VM, a separate TPC-H database is built in as indicated in figure 4.1. Each TPC-H database consumes approximately 5GB of SAN space and 2GB of shared memory.

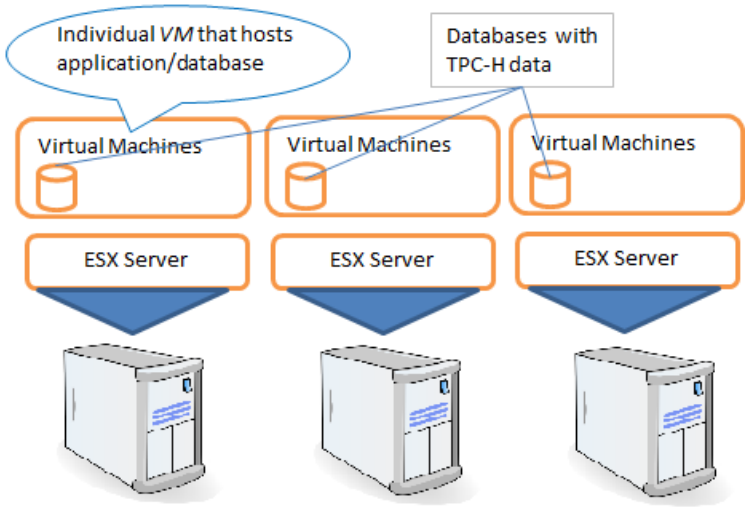


Figure 4.1: Layered depiction of VMWare Virtualized Infrastructure for database hosting. The separate TPC-H database is built in each database VM.

The envisaged *optimization* and *affirmation* algorithms are run against the TPC-H database. As the proposed models are very sensitive to noises in both the database and operating system (OS) environment, outage windows are required during both setup and steady state phases. During the setup phase, the *optimization* scheme requires 2 hours of 'quietness' in the VM to construct the baselines. Subsequently when it is deployed in the steady state production phase, each test will consume 30 minutes. As it is a cyclical probing of the VM during the production phase, it is recommended that this 30-minute timeframe to be provisioned weekly. The *affirmation* scheme requires a lengthier downtime on the VM. Its setup phase needs between 5 to 6 hours, whereas its application for production phase calls for between 30 minutes to1 hour of outage window.

Nevertheless, the implied 'outages' for both the *optimization* and *affirmation* models do not actually require all the user databases to be shutdown in the VM. It is suffice to ensure a state of dormancy in the databases and VM, without needing to bring down any of the software components in the VM.

4.3 Theme 1: *monitoring* scheme

The *monitoring* scheme strives to produce a model that increases the visibility to the resource consumption state in the VM. The output from it will aid in resource planning decision as well as determining the resource scaling opportunity. This model will complement existing commercial resource monitoring tools, and provide a different monitoring perspective as compared to these tools. Most of the existing monitoring utilities focus on either the database or OS, rarely a strong combination of both. Even if there is linkage between the 2 areas in these products, the proposal here can deliver a stronger binding to produce a more cohesive monitoring output.

4.3.1 Workload traces

To explain the design of the model, the input is elaborated in this section. The experiment was conducted using a Sun Solaris server powered by 4 Sun Solaris SPARC64-VII CPU with 4-core architecture, 64GB RAM and external SAN running on ZFS File system. When this experiment was conducted, the VMWare VM had not been available; hence the tests were not performed on a virtualized machine. Nevertheless the experiments produced the same concept that can be proliferated to database operations on Cloud VM without much variation. The application was running the SAP Enterprise Resource Planning (ERP) software, on ECC6 Human Resource Management (HRM) Module. The application is OLTP in nature, servicing HRM System for a large organization. The database that hosts the application data and transactions is an Oracle 11g database. In a lot of organizations, it is almost impossible to send all the tables' data into memory, as the typical implementation of ERP application is normally having database size which is at least 500 GB. However, a sizable shared memory of 20 GB will ensure most frequently used tables' data resides in the memory.

In chapter 2, it was mentioned that the reason for this research is to produce mechanisms that steer away from real data access, on databases that host sensitive data particularly in healthcare organization. The selection of HRM module here is due to the fact that generally the databases that host this application are having quite stringent data security requirements; hence the choice here is almost equivalent to the security demand in Healthcare sector.

The collection of the identified workload's metadata spans for a week. The goal here is to find a workload that can be representative for the critical transactions in the applications, as well as the decisive timeframes that heavily load up the server. A week of data here is assumed adequate in delineating the nature of the resource consumption pattern of the application. At this juncture, it is noteworthy to mention the characteristic of different application processing, to serve as input in identifying the most appropriate and relevant timeframes to exemplify the particular applications. From HRM perspective, the general applications will have their busy computing duration during normal working hours. Hence in this case, even if the data collection timeframe is to span for a week, the daily data collection may only account for 8 hours duration when the employees in the company are officially working. On the other hand, in typical performance management systems, they are mostly active only during particular timeframes in a year, for instance, at the beginning of every quarters. For such applications, the workload data capturing activity should only focus on these busy timeframes. Furthermore, in many human resource departments particularly in United States, there is a period of time in a year when the systems are opened for employees to add, change or delete their fringe benefit options. Such timeframe is generally called Open Enrollment. Hence, the workload capturing activity should target only this duration. The collected workload can be partitioned further if required, in case where the resource planning aims to provision additional resources to the entire hosting

environment. Workload partitioning is not researched in length here, but it will become an important topic in the case when VM expansion subject is detailed later.

The metadata required as input to the model is collected at every minute interval in the database. When this metadata collection engine runs, the resource overhead observed is less than 2% of the total computing capability in the server. Hence, it is considered negligible in overall effect on the server performance. The collected data comprises of DB CPU Time (DCT) and SQL Elapsed Time (SET) from the SQL processing in the database, and the corresponding CPU run queue length from the OS perspective. These data are stored in a custom table. For a SQL, SET denotes the time needed to completely return the result queried by the SQL. It considers all variables affecting the SQL's runtime, i.e. time required by database engine, server condition, network latency, memory and disk I/Os etc. The time spent by the database engine is the cumulative non-idle CPU time to process the SQL. The server condition contains the noises which delay the execution of the SQL. Network latency delay is caused by the transportation of query result from the database to the application servers or end users' terminals. The memory and disk I/Os are the result of fetching the query outputs from data in memory or SAN storage. DCT is a subset of SET, where it represents the non-idle duration needed only by the database engine to process the SQLs. These 2 parameters can be retrieved from the database dictionary.

As mentioned, these metadata can be partitioned to serve various purposes. Asides from capacity planning, the partitioned workload can be utilized for task scheduling algorithm if ever desired. Subsequently the *monitoring* model proposed here can be employed to analyze the feasibility of moving the partitioned workloads to different VM. In this thesis, the workload is not partitioned. The workload partitioning will be covered in future works.

At this point, with the identified workload to deliver as input to the *monitoring* model, there seems to be a potential possibility that it can serve as benchmark for similar transaction processing in other environments that administer similar transactions. However, there are distinct differences between different set of workloads. Each workload is unique as the volume of data differs. Because the data volume is not the same, even the same SQL with identical syntax does not yield the same DCT and SET. This can be explained by considering the following SQL:

select department_name, sum(salary) from departments

where num_employee>100;

If this statement yields 300000 logical reads and 5s of DCT from the real data in the memory, in another environment where the table *departments* has 100 row in it, the logical reads will be much less than 300000 and the DCT could be 0.01s, as different execution path is taken by the RDBMS engine due to the change in data values that changes the predicate clause. With this characteristic, the workload processing pattern obtained from 1 environment can be used the most as reference instead of treated as benchmark in another instance.

4.3.2 Graphical representation – linear regression

Following the data capturing phase, the data is interpreted by presenting the data points in graphical fashion. In this case, *linear regression* is employed to depict the relationship between the SQL processing time, which includes DCT and SET, with CPU run queue. This correlation is illustrated in figure 4.2. The collected data translation to these linear plots is straightforward. However as the data is gathered during production phase, in some cases, the outliers are significant. The ideal situation is to have only the workload to be executed in the VM, without any other computing tasks, or in other words, the noises incurred from the OS. For instance, the gathered data

must avoid the timeframe when the OS backup is taking place. In other circumstance, there might be overrun daemon processes at the OS level which significantly affect the CPU run queue reading. The outlier removal process must account for such condition. The perfect linear plots will yield the *correlation coefficient* in *linear regression* analysis, *r* to have value of 1. However as this is non-achievable in real situation, *r* should be more than 0.7 in order to produce a convincing output for further analysis.

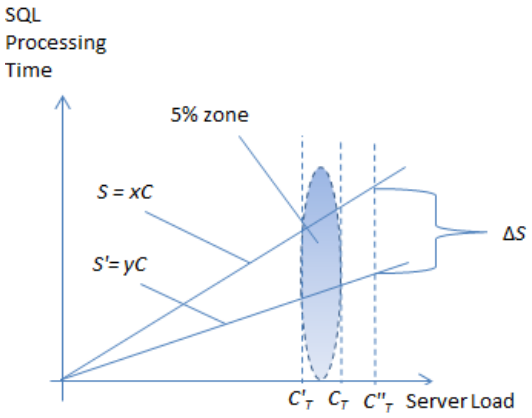


Figure 4.2: The relationship between DCT, SET and CPU run queue. With the 2 linear plots obtained, further analysis produces 80%, 90% and 100% of run queue threshold values.

The calculation of *r* is as follows:

$$r=\frac{\sum_{i=1}^{N} y_i x_i-\frac{1}{N}(\sum_{i=1}^{N} y_i)(\sum_{i=1}^{N} x_i)}{\sqrt{\left[\sum_{i=1}^{N} y_i^2-\frac{1}{N}(\sum_{i=1}^{N} y_i)^2\right]\left[\sum_{i=1}^{N} x_i^2-\frac{1}{N}(\sum_{i=1}^{N} x_i)^2\right]}},$$

Where,

N = Total number of legitimate data points. These points are obtained after removing the outliers.

y = The CPU run queue size.

x = DCT or SET, which corresponds with the *y*.

The computation of this value *r* can be performed easily using MATLAB software (MATLAB, 2013). As this calculation is non-iterative, it can also be done using regular

Microsoft Excel worksheet. However, as mentioned this value must be greater than 0.7 to yield any meaningful output, there is a cyclical effort to remove outliers on the gathered data, and then recalculate the *r* value until the satisfactory value is obtained.

In the graph as in figure 4.2, S' and S represent the DCT and SET of all the SQL processing in the workload. C'_T, C_T and C''_T are the 80%, 90% and 100% of the run queue thresholds in the VM. The derivation of C'_T, C_T and C''_T values depend on the pre-determined ΔS value, which normally is set as $S - S' = \Delta S$, and ΔS is ideally set to equate to value of S. If this is translated to the processing of a SQL, that means if the SQL needs 5s to be processed without any resource constraint in the VM, at the point C''_T, it needs 10s to be processed. This occurs because in this case there is always 1 queue in all the processors at C''_T. As queuing exemplifies delay in the VM, it is undesired if there is occurrence of data points beyond C''_T. When this happens, it should warrant replenishment of resources to the VM. In the published journal paper (C. H. Tan & Teh, 2013a), ΔS is visualized at point C'_T. This is also a valid assumption, amid a more cautious approach to the resource threshold.

With the linear plots and threshold points identified, the imaginary 5% zone is envisaged in the graph in figure 4.2. For a typical workload in a database, generally the response time of a particular transaction fluctuates amid different level of resource adequacy in the VM. Figure 4.3 illustrates such condition. This visionary 5% zone is envisaged to cater for such expectation. As the database transactions intensify, the resource consumption will increase. This will push the data points towards the 5% zone. As manifested by the name, when more than 5% of the total N number of legitimate data points accumulates in this zone, the resource scaling algorithm should be triggered.

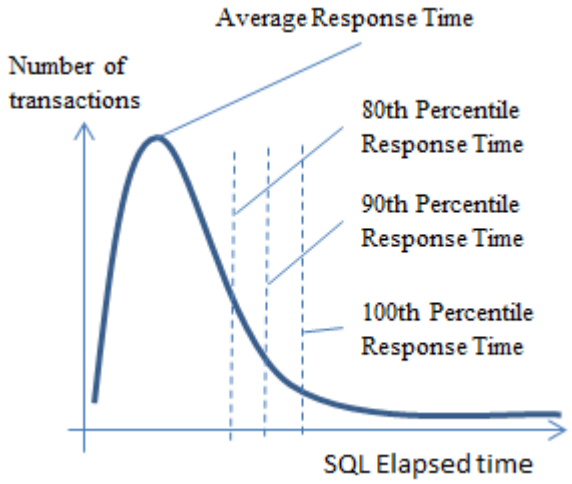


Figure 4.3: Response time fluctuation of a transaction in the VM. Such behavior is expected in real environments as it is often difficult to achieve a constant response time due to various condition in the VM.

As mentioned in earlier chapter, this *monitoring* model is also capable of accommodating burstiness in transactions processing. In this case the 5% zone can be pushed towards higher CPU run queue values to depict the threshold of the burstiness.

In the case where the model is to be employed for resource planning purpose, which normally requires a few months to have the capital expenditure approved in most organizations, the 5% zone can be adjusted at lower CPU run queue length. However, for resource scaling purpose where the computing resources are readily available in the resource pool of the virtualized clusters, this 5% zone can be shifted as far as beyond the point C''_T.

This data collection and monitoring process is iterative and it is to be repeated at weekly interval for typical general purpose applications. The envisaged cyclical monitoring mechanism is illustrated in figure 4.4.

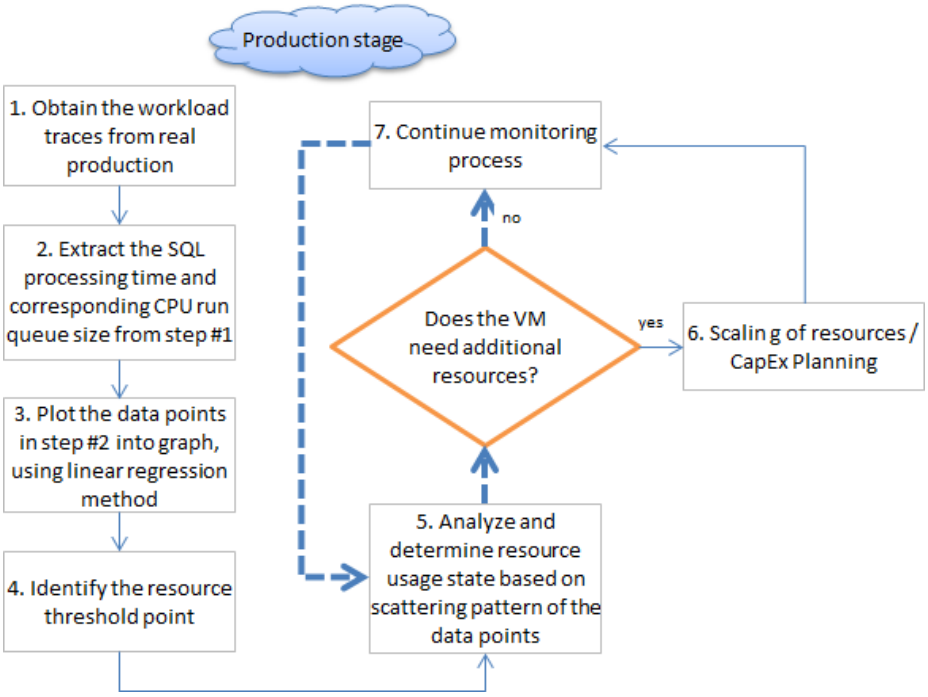


Figure 4.4: Steps towards achieving the proposed monitoring scheme. The dotted lines indicate the cyclical monitoring and analysis tracks.

4.3.3 Outliers effect

Outliers are largely unavoidable in the linear graphical plots. These are caused by noises in the OS. As mentioned, the data collection should avoid the timeframe when the OS undergoes maintenance, for instance the OS backup, auditing and anti-virus scanning. However there are scenarios which are unexpected and unavoidable. An example is the occurrence of overrun daemon processes in the OS which consume significant resources in the VM. In other situation, the OS administrators might be doing ad-hoc OS logs housekeeping which consumes the I/O bandwidth in case the data is sharing the same LUN with the OS logs. As in many other industries (Amos, 2013), getting rid of outliers is critical to protect the legitimacy of the output.

For this purpose, the Fourth-Spread *(fs)* method(Devore, 2008) to exclude the outliers in the scatter plots is employed. To accomplish this, the *25th and 75th* percentile quartile boundary values in the collected data sets need to be determined, using following sequential method:

Order the data set values in ascending order.

$X_1, X_2, \ldots, X_n,$

For even sample size, find the median of the data set as $M = \frac{X_{\left(\frac{n}{2}\right)} + X_{\left(\frac{n}{2}+1\right)}}{2}$,

25th and 75th percentile boundaries of the ratios of *S/C* from the collected samples are then defined as:

25th percentile boundaries, Q_{25} = Median of $X_1, X_2, \ldots, X_{(\frac{n}{2})}$,

75th percentile boundaries, Q_{75} = Median of $X_{(\frac{n}{2}+1)}, X_{(\frac{n}{2}+2)}, \ldots, X_n,$

For odd sample size, find the median of the data set as $M = X_{\left(\frac{n+1}{2}\right)}$.

25th and 75th percentile boundaries of the ratios of *S/C* from the collected samples are then defined as:

25th percentile boundaries, Q_{25} = Median of $X_1, X_2, \ldots, X_{(\frac{n+1}{2}-1)}$,

75th percentile boundaries, Q_{75} = Median of $X_{(\frac{n+1}{2}+1)}, X_{(\frac{n+1}{2}+2)}, \ldots, X_n.$

With Q_{25} & Q_{75} values discovered, value of Fourth-Spread, $fs = Q_{75} - Q_{25}$. Using this value together with the median, *M* of the data set, values for *Upper Outlier boundary, UO* and *Lower Outlier boundary, LO* are determined by following:

UO ➔ $M + 1.5*(fs)$,

LO ➔ $M - 1.5*(fs)$.

Using both the *UO and LO*, for each data point, if its *S/C* value is lower than *LO* or higher than *UO*, they should be discarded from the legitimate data set.

4.4 Theme 2: *optimization* scheme

The *optimization* model aims at complementing the conventional hardware health check which often requires lengthy downtime. Such thorough hardware probing effort will discover detailed hardware state; however it normally requires a full work day of outage window each time, with a substantial amount of manual intervention in the probing process and outage management. This proposed mechanism strives to provide a heads-up before thorough system checking is triggered. The suggested probing method is closer to end users' experience as it employs database transactions, by mean of iterative execution of TPC-H queries to discover abnormalities. The input parameters from this model is different from the *monitoring* scheme, in the way that only SET and CPU run queue are utilized. DCT is not involved in this probing mechanism. This research work has been published in (C. H. Tan & Teh, 2013c).

4.4.1 Setup of environment

The construction and application of the *optimization* model involve 2 stages. First stage is named the baselining phase. It includes the setup of the separate TPC-H database in dedicated VM. The hosting architecture is depicted in figure 4.1. This is a 5GB database, with shared memory sized to 1GB, which in this case is sufficient to avoid excessive I/O reads from SAN storage. In fact, as the TPC-H data is hosted in its own database, this mechanism is applicable even for application VM that do not host any database transactions. This is because the test data crunching process happens only in the TPC-H database, without dependency on any other software components in the VM.

The TPC-H data and queries are downloaded from TPC website (TPC-H, 2013). The build steps of the TPC-H database are illustrated in Appendix B. The main contribution of TPC-H benchmark in the industry is to allow commercial software vendors to showcase the capability of their products by running the standard queries against the

standard data. The output from the tests can then be published in the knowledgebase pertaining to the relevant industry, so that consumers can make meaningful analysis and comparison with products or service offerings from other vendors. In academic point of view, this benchmark can also be employed to serve similar purpose, where it is utilized for comparison between an envisaged proposal and another from other scholars. Nevertheless, the purpose of engaging such benchmark in this study is not meant for comparison with any of current available standards. This benchmark instead is used as synthetic workload in the proposals.

There are 22 queries in TPC-H benchmark. The response time of these queries, after they are cached, ranges between 1s to 20s, except query #21 where it takes 164s to return the rows in the testing environment. The VM utilized as test bed is running on Suse Linux OS, provisioned with a quad-core Intel processor, with 1GB allocated for shared memory. The queries' response time, memory reads and physical reads when run in this test bed are as illustrated in table 4.1. For the selected queries executed in the experiments, this 1GB of cache in the database incurs quite significant disk I/O reads, however it is not posing as constraint in the VM. The disk I/O reads is required to probe the condition in the SAN storage. The selection of queries and sizing of shared memory in this case is made carefully so that the processor component will be the constraining factor when the VM is stressed to its limit. Because the choice of synthetic workload is using the TPC-H data and queries, the similar adjustment can be applied to other hardware configuration, instead of reworking the memory sizing and selection of queries when non-standard synthetic workload is used.

Table 4.1: Corresponding response time, memory and disk reads of TPC-H queries. These values are obtained in a VM with a quad-core CPU, running on an Oracle database with 1GB of shared memory, with the data stored on *EMC* SAN storage.

TPC-H queries	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
response time (cached)	17	1.5	2.6	5.4	7.4	3.6	6.3	5.5	15	2.4	1.6
memory reads	374634	24073	479701	465965	480577	374894	480579	494141	546761	133667	67236
disk reads	374623	13479	479210	465559	479210	374623	479210	492689	548617	133764	66249
TPC-H queries	#12	#13	#14	#15	#16	#17	#18	#19	#20	#21	#22
response time (cached)	5.6	9.1	4	4.1	2.3	4.2	13	4.4	5.9	164	2.7
memory reads	465967	92524	388458	375766	23643	388458	434989	388458	493626	1216625	118543
disk reads	465559	92200	388102	374623	22317	388102	443220	388102	454357	1521057	118238

Figure 4.5 illustrates the flow of the 2 phases in the setup process. Steps 1 and 2 have been discussed in above section. In step 3, the VM is loaded up slowly and gradually using the selected TPC-H queries. The goal here is to capture the SET that corresponds to the increment in the CPU run queue size. The loading algorithm is depicted in figure 4.6. They are programmed and executed via Shell script in the VM environment. The actual program is illustrated in Appendix C.

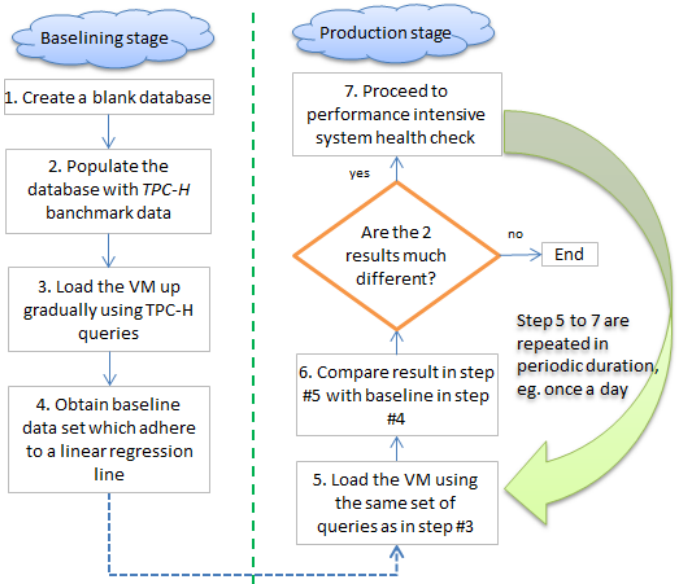


Figure 4.5: Steps to create the baselines and their applicability in production environment for *optimization* model. The green arrow depicts the cyclical test in the VM, which can be performed daily or weekly.

```
Define test time, t as 2520s
Define the set of queries to run for the duration of t
Define the stabilizing duration as 120s
    While t>0,
      If (t ==2520)
         Continuously maintain the execution of 6 sets of queries in the database
      If (t ==2000)
         Continuously maintain the execution of 5 sets of queries in the database
      If (t ==1600)
         Continuously maintain the execution of 4 sets of queries in the database
      If (t ==1200)
         Continuously maintain the execution of 3 sets of queries in the database
      If (t ==800)
         Continuously maintain the execution of 2 sets of queries in the database
      If (t ==400)
         Continuously maintain the execution of 1 set of queries in the database
      If (t ==0)
               Complete test
      t=t-1
      If (t ==2400)
         Start capturing database snapshots to collect values of S and C every 30s
    done
```

Figure 4.6: Algorithm to load up the VM. The baseling and production phases are using this same method.

The Shell program is capable of using single or multiple TPC-H queries to stress the VM. The environment is loaded with continuous execution of the queries, and they are varied in concurrent number of executions in order to produce the high to low resource consumption scenarios in the VM. In parallel to this loading activity, database snapshots are captured every 30s to provide the information regarding the state of SET, *S* and corresponding CPU run queue, *C* at each interval of interest. This snapshots capturing capability is provided by the database Workload Repository utility (Oracle, 2009). Each loading activity is scheduled to complete in 42 minutes. It is noteworthy that prolonged test time will produce better accuracy.

The CPU run queue value is taken as in 1-minute average, and it is noticed that these values fluctuate quite substantially during the initial loading. Hence some technologists term this parameter as simplistic and poorly defined in Unix environments. However as demonstrate by the experiments, that if longer test duration is allocated, this parameter can be useful in measuring the information of the queuing processes appropriately. To remediate this oscillating symptom, the stressing exercise is allowed 2 minutes of 'stabilizing' timeframe, before data collection begins.

At the end of the test, the required metadata is extracted from the *Workload Repository*. As mentioned above, the fluctuation in the values can be considerably huge at particular instances of time. Hence the implausible data needs to be filtered out. The algorithm to filter the metadata is illustrated in figure 4.7. As shown in this figure, in order to ensure accurate *S* and *C* data points, the start and end values of *C* in the 30s snapshot intervals are assessed so that they are less than 10% different from each other to ensure that consistent state is achieved before data recording is performed.

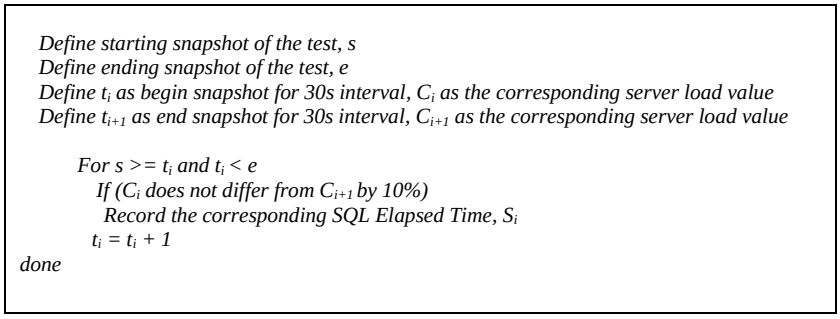


Figure 4.7: Metadata filtering to ensure stabilized condition in the VM before reliable data is collected.

The data extraction and filtering are actually performed by a Shell script. This script is depicted in Appendix D.

4.4.2 Linear regression and machine learning

With the availability of data, the linear plots can now be constructed. Similar to the *monitoring* model, the relationship between the SET and CPU run queue is linear in nature. The proof of this linear relation is explained in section 2.4.3.1. The linear plot is depicted in figure 4.8. In this figure, the DCT is shown; however it does not serve as input parameter to the *optimization* scheme. It is illustrated in the graph to depict the actual non-idle CPU time needed to process the SQL. The blue dots in figure 4.8 does not increase in tandem with the increment in the CPU run queue, because there is only 1 SQL running entirely in the database. This is different from figure 4.2, as the workload for the *monitoring* scheme contains multiple SQLs. As expected, SET increases when

CPU run queue rises. This is due to the additional waits in the VM when the processors are getting busier. Such characteristic has been explained in the *monitoring* scheme. As the interest is confined only to the linear section of the correlation between *S* and *C*, the limit of the run queue size should be set to <= 3*x*, where *x* is the number of processor cores in the VM. The values of 3*x* is obtained judging from the experiment data using TPC-H queries, where most testing results using this ballpark figure can still conform to the linear correlation when the queue length in each processor is 2. The baselines of data obtained from step #3 & #4 in figure 4.5 are called the training data sets in *machine learning*, and they are executed when the VM is newly provisioned. Subsequently these baselines are reevaluated when the hardware configuration in the VM changes.

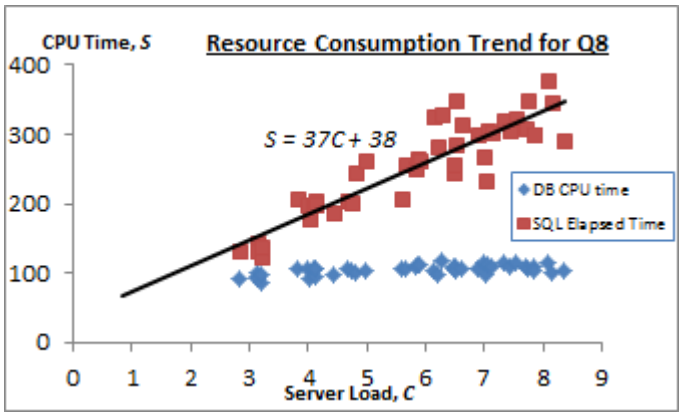


Figure 4.8: The expected output for the *optimization* scheme. The linear relationship between SET and CPU run queue is the foundation to this model.

The steps #3 to #6 in figure 4.5 can be categorized as the *semi-supervised machine learning* technique. *Semi-supervised machine learning's* concern is to infer a function from labeled training data sets (Mohri et al., 2012), and subsequently the unlabeled data is tested and compared to this induced function. In this case, the first set of data during VM initialization is taken as the labeled training sets. Regression Analysis algorithms, which are subset to the broader classes of *semi-supervised machine learning*, are employed here to generate the labeled training data sets. By definition, a training data

set, *V* draws its samples from set *X* according to distribution *D*, which in this case are the sets of relationship between SET, *S* and CPU run queue, *C*.

$V = ((c_1, s_1), \ldots, (c_m, s_m))$ *and* $V \in C \times S$,

where $Y \subseteq \mathbb{R}$ and *m*=number of training data.

Subsequently during the production phase, the test data, which is termed unlabeled data in *machine learning*, is used for testing the learning algorithms, as in figure 4.6 and 4.7. The characteristic difference between the training and testing data sets is known as the *Loss function*, *L*, and it is a measurement of closeness between these two data sets, and is having following definition:

$L(S, S') = |S' - S|^P$ *where* $P \geq 1$,

In the *optimization* model, input to *L* is characterized by the gradient of the linear regression line and *y*-intercept, as illustrated in figure 4.8. More experimental results related to the construction of baselines are displayed in Appendix E.

The outliers should be gotten rid of by employing the Fourth-Spread *(fs)* method explained in section 4.3.3.

4.4.3 Applicability to the production phase

The gradient, M_B of the linear line is obtained, together with the *y*-intercept, b_B. M_B and b_B are asserted as baseline for expected performance on allocated resource in the VM. Iterative learning process is conducted to arrive at the most accurate values of M_B and b_B. Subsequently in the production phase, testing data sets are collected and plotted into linear graph, and the respective gradient, M_T and *y*-intercept, b_T are recorded. Subsequently they are compared with M_B and b_B values from the training data sets, to gauge the consistency and optimality of the hardware performance. In comparing

between the baselines and the testing data sets, there is another critical parameter that needs to be accounted for. It is the correlation coefficient, *r*. As mentioned, the value of *r* needs to be as close as possible to 1. The details of this explanation are provided in section 4.3.2. With the information, the characteristic of the model can be represented by *Fuzzy Computing with Words*(Zadeh, 1996), as follows:

- *If values of* M_T and b_T *are much different as compared to* M_B and b_B*, hardware performance in the VM is not optimal.*
- *If value of r is not close to 1, hardware performance in the VM is not consistent.*

The above 2 conditions must be satisfied in order to guarantee fine delivery of computing resources to the hosted databases or applications in the VM.

In converting the testing data sets to the linear plots, the cautions and approaches are similar to the baselining phase. They are:

1. The outliers need to be gotten rid of using the Fourth-Spread *(fs)* method explained in section 4.3.3.
2. The noises in the OS and potentially database need to be kept at minimum. For instance, OS or database backup must not be running during the collection of testing data sets, CPU run queue must start from value 0 before the loading mechanism is triggered, OS auditing and anti-virus must not be running in the background while the data collector is working etc.
3. To compare with the baselines, same TPC-H query must be used. The statistic in the TPC-H database must be updated in case there is change of data in the database.
4. The same stabilizing duration during the loading mechanism must be observed, before data collector can start gathering the metadata information.
5. The loading mechanism ideally should be performed during outage window.

As per figure 4.5, the collection of the testing data sets is a cyclical process. Depending on the available outage window, it can be performed either daily or weekly. As the loading activities do not involve changes of data, the TPC-H database is almost maintenance-free. There is not even a requirement to intervene with the database restart in case the VM is rebooted, as the database can be stopped or started by the *init* program in Linux. The loading mechanism can also be automated almost fully. The Shell scripts involved can be triggered by Linux *crontab* scheduler. Subsequently the values of gradient, y-intercept and correlation coefficient from the collected testing data sets can be processed by MATLAB software easily.

The output from the comparison between the baselines and testing data sets is analyzed, and depending on the magnitude of difference, 2 actions can be performed:

- In the case where mild discrepancy is observed, a system reboot may be warranted. This action is taken as part of the software rejuvenation initiative discussed in (Andrzejak & Silva, 2007; Vaidyanathan & Trivedi, 2005).
- As mentioned earlier, the intention of this *optimization* model is to complement the conventional intensive hardware health checking. If serious abnormality is observed from the result, such comprehensive testing can be triggered to identify the actual culprit to the hardware problem.

4.5 Theme 3: *affirmation* scheme

The *affirmation* model strives to provide a speedier alternative to conventional load testing. These commercial load testing utilities, for example the HP Load Runner, are very comprehensive in accomplishing the objective of stressing all the components in the application hosting architecture. In these tests, test cases are created to observe the behavior of the web server, load balancer, application server and the database server. However, stress testing is normally conducted prior to the production cut-over activity.

During steady state, particularly for mission-critical applications, to locate lengthy downtime to perform such cumbersome tests are often impracticable. The suggested *affirmation* model requires a much shorter outage window in order to stress the VM to the level almost equivalent to the comprehensive load testing using the real workload. The stress-testing scenario subsequently can be utilized to verify SLA-bound transactions. It is to note that this express version of load testing serves to complement the conventional type, as it provides a quick testing platform for stress testing.

The setup of the *affirmation* model is also employed to verify and confirm the resource threshold in the VM, when the hardware configuration is changed. In accomplishing this goal, synthetic workload is utilized. In regards to the same resource threshold boundary identification effort explained in the *monitoring* model, the *monitoring* scheme is utilizing the metadata of real workload to produce the outcome. The *affirmation* model instead scrutinizes on the synthetic workload to gain insightful information on the threshold boundary. The research work of this proposal is published in (C. H. Tan & Teh, 2013c).

4.5.1 Setup of stress-testing scenario

The proposal to construct the stress-testing scenario is applicable to OLAP and OLTP applications that are dominantly performing data read operations. With this assumption, the suggested loading mechanism can have focus on the memory read constraint, and treat the physical read as negligible. Such presumption can be made, because in typical Human Resource applications, the largest table in the database seldom exceeds 20GB in size. If the shared memory in the database can be sized considerably to > 20GB, most of the database read operations can be cached. Hence the core of the proposal in this case, is to determine the amount of memory reads/s in the real workload during steady state, at particular CPU run queue size. Subsequently, synthetic workload is utilized to

fabricate the stress-testing scenario, by matching this memory reads/s value from the real workload. In doing so, the CPU run queue is taken as a constant, which must be the same as the condition in the real workload when the memory reads/s value was harvested. Nevertheless, this memory I/O operation must not pose as the primary resource constraint in the VM. Instead it should be dominantly larger than the disk reads, however the leading resource constraining factor must still remain at the processor component. As mentioned, such behavior is common in most hosting architecture. A superior IT architectural hosting model should have resource limit confined at the processor component, as it is generally the most expensive part in the hardware configuration. At the same time, the application development must ensure proper tuning of SQLs and tables' structure. These 2 pre-requisites are not only important and applicable for the proposal in this thesis, they are also essential to ensure feasibility in serving the business objective of the application service offerings.

The setup of the environment is similar to the *optimization* scheme explained in section 4.4.1. A separate TPC-H database is created in the database VM as in figure 4.1. Figure 4.9 illustrates the steps to construct the whole proposal. There are 2 stages in the mechanism. First, benchmarking data that contains the value pairs of TPC-H queries and memory reads/s is discovered by mean of iterative execution of the TPC-H queries. Ideally, the VM should be loaded to the resource threshold so that the SLA-bound transaction verification can produce meaningful information regarding the capability of the hardware in conforming to the response time requirement stipulated in the SLA.

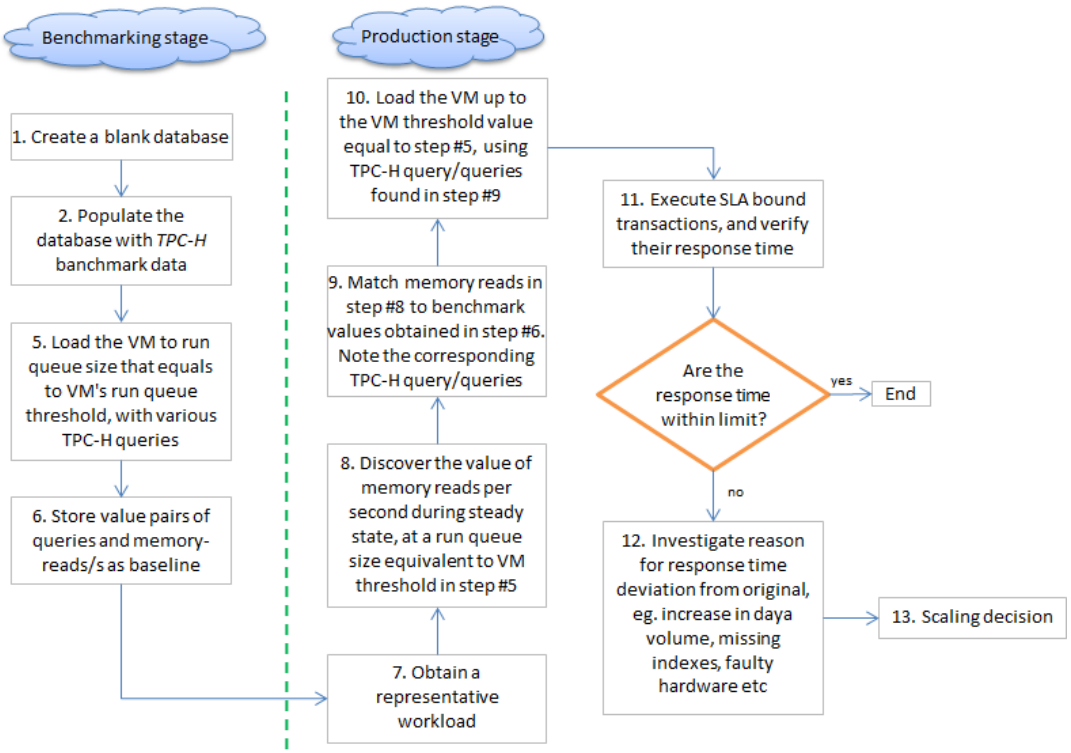


Figure 4.9: The flow of the setup and application steps of the resource utilization *affirmation* model. Similar to the *optimization* model, there are 2 phases involved.

In the benchmarking stage, to construe the setup algorithm for the stress-testing mechanism, following are the high level steps.

Step 1: Each individual query in TPC-H benchmark is iteratively executed in the VM to be tested, to obtain individual value of memory reads/s (MR) and duration of run time.

Step 2: Step #1 is also carried out for combination of queries.

Step 3: These results are listed in ascending order, to serve as benchmark to be chosen for individual testing scenario. They are stored in array format as in figure 4.10.

A[Query][MR] = {Q16, Q9, Q13, Q21, Q3, Q7, Q5, Q11, Q1, Q22, Q4, Q10, Q14, Q6, Q18, Q19, Q20, Q12, Q15, Q17, Q2, Q8}, {7000, 11000, 10000, 10000, 11000, 11500, 12500, 12500, 13000, 14000, 14500, 14500, 15000, 16000, 16000, 16500, 16500, 17000, 17000, 19000, 22500, 16000};

Figure 4.10: Array that stores the benchmark data for *affirmation* model. It is a construct of value-pairs of TPC-H queries and memory reads/s.

As mentioned, there are 22 queries in *TPC-H* benchmark. Each one of them is having distinct characteristic in term of total MR and runtime in producing the results. As depicted in figure 4.11, these queries are iteratively executed in the VM in parallel. For example if the VM is comprised of a quad-core CPU, logically the CPU run queue threshold, C_T value is 4, so that at any time there is 1 process queuing for the CPU resource. In this case the parallel execution of the query needs to be maintained at f_T which will have a value of 8, which produces average 4 queues at any time during the test. Since there are 4 cores in the processor, it means at any 1 time there is 1 queue in the virtual CPU. This explanation is similar to the description of ΔS defined in section 4.3.2. However C_T can be higher, depending on the negotiated transactions' response time requirement by client's SLA. The $C_T=4$ in this case is a ballpark figure for a quad-core VM, which should be a general guideline. This is because any higher value will incur significant waits in the transactions processing which are detrimental to end users' experience.

S = set of 22 TPC-H queries, where
S = [Q1, Q2, ... , Q22]

1. Determine number of processor cores
2. Server threshold, C_T is determined by value in step #1 OR predetermined C_T from initial load test
3. Determine run frequency, f_T of query based on step 2.
4. Determine test duration, T

while (true)
Record the start time of test, ST
For each query in S,
if run frequency, $f < f_T$ and Server load, $C < C_T$
and test time, $t < T$
Execute the query
Record logical reads/s and average runtime

Figure 4.11: Algorithm to produce the benchmark data. In this case, value pairs of TPC-H queries and memory reads/s are sought after. A Shell program is constructed to execute the algorithm, which is shown in Appendix F.

The test duration, as experimented, will need to be prolonged for approximately 15 minutes in order for the VM state to be stabilized for measurement purpose. The data collection is oscillating quite significantly, similar to the condition in the *monitoring* and *optimization* schemes. The desired result of the benchmarking experiments is exhibited in figure 4.12.

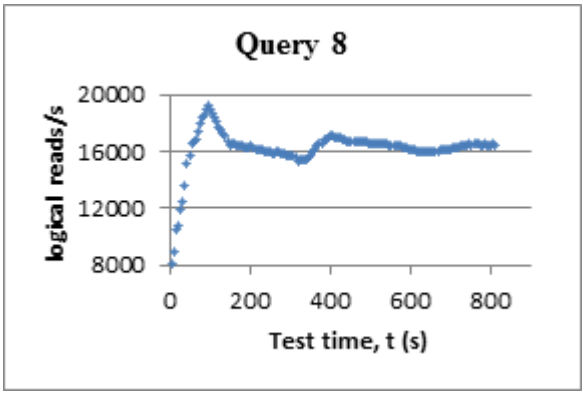


Figure 4.12: Graphical result from the benchmarking experiment. The exhibited result shows the TPC-H query utilized to load the VM to the run queue threshold, C_T=4. The obtained memory reads/s in this case is approximately 16000, after the iterative execution of the query is conducted for 13 minutes, with 8 parallel query execution.

As in figure 4.12, the fluctuation of collected data is quite significant during the first 10 minutes. Subsequently the value is stabilized and the data collector can start to work. This loading duration must be carefully and precisely determined, as the benchmarking stage is consuming quite a substantial amount of time in producing the benchmark

result. Furthermore, these experiments must be conducted during outage window to avoid noises in the VM. As displayed in figure 4.10, there are 22 queries involved in the benchmarked array. As each test consumes 15 minutes, 5.5 hours of total outage window needs to be provisioned to produce the benchmark. As some queries' executions stabilize faster, a detailed scrutiny on the required run duration for each query can save some valuable production time. However the advantage of this proposal in virtualized environment is that the benchmarking tests can be conducted in separate VM with the same hardware configuration, hence the outages needlessly occur in the real environment.

It is also to note that the combination of TPC-H queries can be utilized to enrich the benchmark in figure 4.10. The algorithm to use multiple queries is exhibited in figure 4.13. More experimental results on generating benchmark data is exhibited in Appendix H.

1. Determine number of processor core
2. Server threshold, C_T is determined by value in step #1 OR predetermined C_T from initial load test
3. Determine run frequency, f_T of query based on step 2.
4. Determine test duration, T

S'= combination of TPC-H queries, where
S' = [Q8+Q9, Q2+Q16, Q3+Q6+ Q7, ...]

```
while (true)
   Record the start time of test, ST
   For each query combination in S',
      if run frequency, f < fT/n and Server load, C < CT
      and test time, t < T
         Execute the queries
         Record logical reads/s and average runtime
```

Figure 4.13: Algorithm to produce the benchmark data using combination of TPC-H queries. The same value pairs of TPC-H queries and memory reads/s are produced. The Shell program to execute the algorithm is displayed in Appendix G.

4.5.2 Application to production phase

As indicated, the setup in this *affirmation* phase strives to serve 2 objectives. The first is to create the stress-testing scenario. Secondly, it is employed to discover the CPU run queue threshold in the VM when the hardware configuration undergoes changes.

4.5.2.1 Creation of stress testing scenario

In figure 4.9, step #7 indicates that a representative workload is to be obtained. This workload has to be the typical load in the database where most constraining situation in the application is predicted. Furthermore, it must depict a timeframe of steady state at CPU run queue threshold level, C_T. In the experiment, C_T is identified as 4 as the VM is having a quad-core processor. This information regarding the workload can be mined from *Workload Repository*, where it has a background daemon in the database that acts as the metadata collector. As this mechanism is provided by the RDBMS vendor, it is not elaborated here. The extracted metadata information will contain the memory reads information. As mentioned, this MR from the real workload is to be matched with the benchmark array, to discover the TPC-H query that displays similar computing characteristic to the real workload. Once the query or queries is found, it is use to create the stress-testing scenario in the VM. The step here is similar to step #5 in the benchmarking phase in figure 4.9. The iterative and parallel execution of the identified TPC-H query or queries must be performed in restricted environment, where the VM must be safeguarded from noises. In this case, the best possible option is to provision a maintenance window for the build phase in this stress-testing experiment.

Subsequently when the stress-testing condition is fabricated and stabilized in the VM, SLA-bound transactions are executed in the VM. As the CPU run queue is stable in this condition, the response times of the transactions are unlikely to exhibit the behavior as exhibited in figure 4.3. However, even though the incurred disk or memory I/O values are consistent, it is unlikely that the response time results will be even. Fluctuation in the response times is expected. Hence the verification of transactions should be performed in multiple iterations to guarantee convincing output. The outcome from the transaction verification yields following actions, which is best interpreted using *Fuzzy Computing with Words* (Zadeh, 1996):

- *If most of the transactions' response times that are obtained from same set of tables do not conform to initial expected response time, data volume is changed, and more resources are needed in the VM.*
- *If a subset of transactions in the pool of transactions that use the same set of tables do not conform to initial expected response time, some indexes in the query are corrupted and need rebuild.*
- *If most transactions do not exhibit consistent response time even after many rounds of testing, the hardware performance is not consistent, and further system health check is warranted.*

4.5.2.2 Threshold verification

The same setup is employed for threshold verification in this section. Contrary to the threshold verification exhibited in the *monitoring* scheme where metadata from real workload is utilized as input, the input to the threshold boundary analysis is channeled by synthetic workload using the TPC-H queries. Even though both methods strive to achieve the same goal, each contributes to different interpretation. The *monitoring* model provides a depiction of the VM resource boundary that is closer to end users' experience, as real workload is derived for analysis. Meanwhile the method in the *affirmation* model is more speculative, as synthetic workload is employed.

Figure 4.14 illustrates the steps to construct this mechanism. Most of the steps here have been elaborated in above sections. The more interesting step is in step #5. It exhibits the extrapolation to discover the theoretical CPU run queue threshold. The principle to determine the threshold is to equate the CPU run queue to the number of virtual processors. For instance, if the single quad-core processor in the test bed is increased from 1 to 2, the deduced number of cores will be 8. Hence the threshold is set to 8. The

rationale to this has been explained in section 4.3.2 where ΔS is defined; hence it is not repeated here.

The basis of construction to the model is similar to the stress-testing scenario described in section 4.5.1. However the memory reads/s parameter is not important here. This is because throughout the verification process, only particular TPC-H queries are utilized for testing, and there is not a necessity to vary the queries. The details for steps # 6 and 7 are as follows:

1) Assume TPC-H query #8 is utilized to load the VM. The same steps to setup the stress-testing scenario are employed here. However the parallelism of the query execution is incremented discretely. At each increased granularity, another TPC-H query is employed for response time verification.
2) However there is no requirement to match the memory reads/s parameter to any benchmark. Assume TPC-H query #7 is utilized as the query where response time is to be tested. The average response time of 6.3s for this query should be preserved within the threshold boundary.
3) At the point when the average response time of query #7 is not adhering to initial value of 6.3s, the CPU run queue value at that particular instance is taken as the threshold of the resource utilization in the VM.

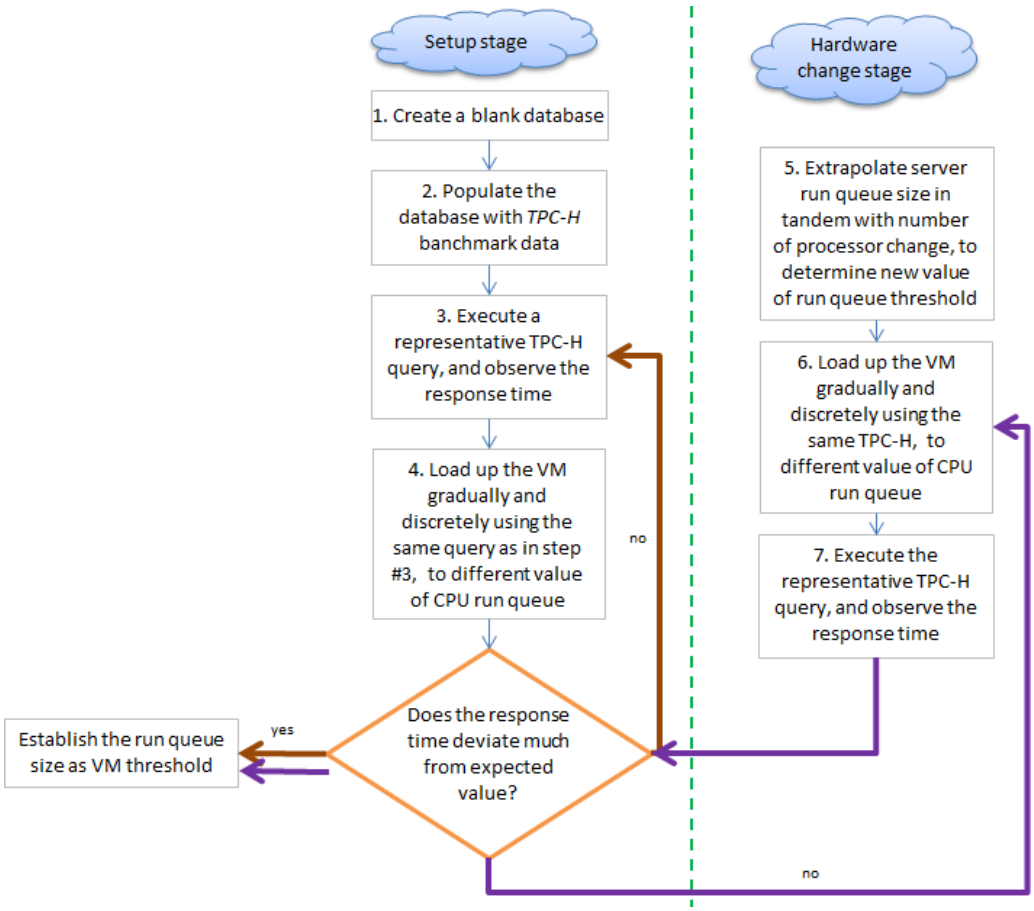


Figure 4.14: Steps to discover resource threshold in the *affirmation* model. Note that in this case, the processor is the constraining point in the VM.

4.6 Potential improvement

The construction of the stress-testing scenario in section 4.5.1 assumes that the physical reads in the database are negligible. In mission critical application where objective function is to hasten the transaction processing as fast as possible, such assumption is valid and considered a best practice in application hosting, particularly in virtualized cloud environment. Nevertheless there are situations where the stress-testing scenario is needed in the hosts, where physical reads are significant and cannot be neglected in the model construction. In such cases, the biggest challenge is to discover the memory reads (MR) and physical reads (PR) for the employed TPC-H queries, at particular CPU run queue. Such challenge is complicated as it needs to take into few considerations as follows:

1) The behavior of the database engine in processing the iteratively run queries, particularly when the shared memory is not caching the entire queries needs to be detailed.

2) Waits might be significant as SAN storage might pose as constraining factor.

3) Theoretically, if query #1 and #2 are combined and iteratively run, the total memory and physical reads incurred should be a sum of each individual query's memory reads and physical reads. Nevertheless, this hypothesis will need to be proven experimentally.

4) The data collection oscillates quite substantially. A steadier mechanism to data loading needs to be envisaged.

The exploration and validation to the above uncertainties are to be continued in subsequent research. Nevertheless, once the MR and PR can be determined convincingly, the choice of TPC-H queries to be utilized to load the VM can be computed using the *linear programming* and *simplex method*. The next section illustrates the potential solution in choosing the queries using these 2 mathematical algorithms.

To explain the suggested method to choose queries which are most suitable for data loading, following 2 rules are assumed:

1) The queries' response time should be as short as possible, so that the fluctuation of the MR and PR parameters can be kept as small as possible. For instance, a 1 query is capable of producing required MR and PR values of 10000/s and 20000/s respectively, and its response time is 20s. Query 2 is also capable of producing the same amount of MR and PR in 1 second duration, but it needs only 5s to run. In real condition, these MR and PR values are not smoothened across the whole tenure of the query execution. With this setback, the transaction verification process will not

produce desired consistent response time. To lessen the impact of such behavior, query 2 is preferred in this case, as the shorter the processing time, the more consistent the values of MR and PR are generated.

2) When 2 queries are run in the database, the total values of MR and PR is the sum of individual MR and PR values. Such assumption is made here for easy depiction of the algorithm. Future research will scrutinize on the potential improvement on this hypothesis.

To start the mathematical depiction of the proposal, assume 3 queries are involved. In actual scenario, there are 22 TPC-H queries to be chosen from. However to ease the explanation, 3 queries are selected to describe the detailed steps. These queries are having the attributes as displayed in table 4.2.

Table 4.2: Attributes of the queries potentially involve in the construction of stress-testing scenario.

	response time (s)	MR /s	PR /s
Query 1	10	10000	5000
Query 2	20	15000	20000
Query 3	15	25000	11000

In addition, assume the CPU run queue threshold is 8, which is equivalent to 2 units of quad-core processors allocated to the VM. The arrival to this value is explained in section 4.3.2, where ΔS is defined. With this information, it is derived that the total parallel run of the queries should equate to 16. The objective function of the *linear programming* here is to minimize the response time as per rule #1 above. So,

To minimize,

$Q = x_1 f_1 + x_2 f_2 + x_3 f_3$.

Subject to following constraints:

$MR_1 f_1 + MR_2 f_2 + MR_3 f_3 \geq M$,

$PR_1 f_1 + PR_2 f_2 + PR_3 f_3 \geq P$,

$f_1 + f_2 + f_3 \geq f$,

$f_1 \geq 0$, $f_2 \geq 0$, $f_3 \geq 0$ because parallel execution of queries cannot be negative,

Where,

MR_i is the individual MR of query i, S_i,

PR_i is the individual PR of query i, S_i,

M is the desired MR to be loaded in the VM,

P is the desired PR to be loaded in the VM,

f is the parallel execution of the queries, which corresponds to the CPU run queue threshold in the VM.

Hence, putting in the values, the objective function becomes

$$Q = 10f_1 + 20f_2 + 15f_3, \text{ with following constraints:}$$

$$10000f_1 + 15000f_2 + 25000f_3 \geq 150000,$$

$$5000f_1 + 20000\, f_2 + 11000f_3 \geq 120000,$$

$$f_1 + f_2 + f_3 \geq 16,$$

where,

$$f_1 \geq 0,\ f_2 \geq 0,\ f_3 \geq 0,$$

The augmented matrix corresponds to the minimization problem is as below:

10000	15000	25000	150000
5000	20000	11000	120000
1	1	1	16
10	20	15	0

The matrix corresponds to the dual maximization problem (Cengage, 2013) is given by the transpose of the above matrix.

10000	5000	1	10
15000	20000	1	20
25000	11000	1	15
150000	120000	16	0

The dual maximization problem implies that the objective value, Q of a minimization problem in standard format has a minimum value only if the objective function of the

dual maximization problem, *P* has a maximum value. In addition, *Q* is equal to the value of *P*. *Q* and *P* are defined as follows.

In the dual maximization format, the dual objective function, *P* is derived from above transposed matrix:

$$P = 150000\, y_1 + 120000\, y_2 + 16\, y_3 ,$$

Where it is subjected to following dual constraints:

$$10000\, y_1 + 5000\, y_2 + y_3 \leq 10,$$
$$15000\, y_1 + 20000\, y_2 + y_3 \leq 20,$$
$$25000\, y_1 + 11000\, y_2 + y_3 \leq 15.$$

With $y_1 \geq 0, y_2 \geq 0, y_3 \geq 0,$

With the above defined, *Simplex method* can be applied to solve the minimization problem. The dual objective function and the dual constraints are arranged in a tabular format, as in table 4.3. The vertical cells are interpreted as column, c_i while r_i represents the horizontal cells in following explanation.

Table 4.3: Tabular representation of the dual objective function and constraints. The yellow cells denote the objective function.

y_1	y_2	y_3	s_1	s_2	s_3	P
10000	5000	1	1	0	0	10
15000	20000	1	0	1	0	20
25000	11000	1	0	0	1	15
-150000	-120000	-16	0	0	0	0

The next step is to find the pivot to table 4.3. To arrive at the pivot, the most negative value from the row that stores values of the objective function is identified. Subsequently, the values in column *b* are divided by the column's values associated with the negative value. The smallest value from this computational result will affiliate with the row with the pivot. This operation is shown in table 4.4.

Table 4.4: The red colored cell depicts the pivot.

y_1	y_2	y_3	s_1	s_2	s_3	P	
10000	5000	1	1	0	0	10	0.0010
15000	20000	1	0	1	0	20	0.0013
25000	11000	1	0	0	1	15	0.0006
-150000	-120000	-16	0	0	0	0	

$\frac{b}{y_1}$

Subsequently, the other cells in the pivot row need to be reduced to 0. The right-most equations in table 4.5 illustrate how this is performed.

Table 4.5: The reduction of other cells' values to 0, in the pivot column.

y_1	y_2	y_3	s_1	s_2	s_3	P	
0	600	0.6	1	0	-0.4	4	$r_1 - 10000r_3 = R_1$
0	13400	0.4	0	1	-0.6	11	$r_2 - 15000r_3 = R_2$
1	0.44	0.00004	0	0	0.00004	0.0006	
0	-54000	-10	0	0	6	90	$r_4 - 150000r_3 = R_4$

To arrive at the final resolution, the operations displayed in table 4.3 and 4.4 are iteratively executed until all the values in the dual objective function become positive. This repeated computation is exhibited in table 4.5.

In the last table in table 4.6, $s_1 \approx 8$, $s_2 \approx 2$ and $s_3 = 6$, while $b = 44.3$. s_1, s_2 and s_3 correspond to f_1, f_2 and f_3 respectively. The value P, as mentioned, equates to Q in the original objective function. Hence, the minimum response time obtained is 44s. As the total parallel run of the queries should equate to 16, query 1, 2 and 3 are run in parallel in the VM, at the frequency of 8, 2 and 6 respectively.

Table 4.6: The continuous steps to produce the final resolution by the *simplex method.*

y_1	y_2	y_3	s_1	s_2	s_3	P	
0	600	0.6	1	0	-0.4	4	0.006667
0	13400	0.4	0	1	-0.6	11	0.000821
1	0.44	0.00004	0	0	0.00004	0.0006	0.001364
0	-54000	-10	0	0	6	90	

$\frac{b}{y_2}$

y_1	y_2	y_3	s_1	s_2	s_3	P	
0	0	0.58209	1	-0.01791	0	3.50746	$r_1 - 600r_2 = R_1$
0	1	0.00003	0	0.00003	0	0.00082	
1	0	0.00003	0	-0.00001	0.00004	0.00024	$r_3 - 0.44r_2 = R_3$
0	0	-8.38806	0	1.61194	6	134.3283582	$r_4 - 54000r_2 = R_4$

y_1	y_2	y_3	s_1	s_2	s_3	P	
0	0	0.58209	0.6	-0.01791	0	3.507462687	6.025641
0	1	0.00003	0	0.00003	0	0.000820896	27.5
1	0	0.00003	0	-0.00001	0.00004	0.000238806	8.888889
0	0	-8.38806	0	1.61194	6	134.3283582	

$\frac{b}{y_3}$

y_1	y_2	y_3	s_1	s_2	s_3	P	
0	0	1.00000	1.03077	0	0	6.02564	
0	1	0.00000	-0.00003	0.00000	0	0.00064	$r_2 - 0.00003r_1 = R_2$
1	0	0.00003	-0.00003	0	0.00004	0.00006	$r_3 - 0.00003r_1 = R_2$
0	0	0.00000	8.64615	1.61194	6	44.32836	$r_4 - 8.38806r_1 = R_4$

The above illustration only employs 3 queries. In actual situation, 22 TPC-H queries should be involved in the objective functions and constraints. With such lengthy computation, the discovery of the most appropriate value of parallel execution frequency can be obtained using the *simlp* function in MATLAB. The above work has been published as a journal paper in (C. H. Tan & Teh, 2013b).

4.7 Summary and discussion

The designs of the resource utilization *monitoring*, *optimization* and *affirmation* models have been detailed in this chapter. The designs employ statistical computation to produce the proposed mechanisms. In these cases, *linear regression analysis, machine learning, linear programming* and *simplex method* have been utilized to generate the intended and potential outcomes. Each step in the construction of the models has been accompanied by detailed explanation with logic and feasibility, regarding the choice of the related techniques.

Future works have also been covered in the latter part of the chapter. It is hoped that the models will be improved and feasibly applied in the commercial arena.

In the *monitoring* scheme, the characteristic of the workload is detailed. In this case, the workload metadata is taken as input to the model. The relationship between this information from the database and the CPU run queue parameter in the OS is exploited to depict the condition of the resource utilization state in the VM. In this case, *linear regression* technique is employed to describe the workload processing trend. Before the graphical representation is established, the noises particularly at the OS level need to be gotten rid of. In this case the *Fourth-Spread (fs)* method is utilized.

Thereafter, the design for the *optimization* scheme is elaborated. The prototyping of this scheme requires the setup of a TPC-H database. In this research, dedicated database is built for the TPC-H data, so that the noises can be contained to the minimum. The algorithms to construct the model are based on the *linear regression* and *semi-supervised machine learning* concepts. In building the prototype, 2 phases are envisaged. In the baselining phase, the training data sets are collected, usually right after the new hardware configuration is put in place. Subsequently the production phase involves the collection of testing data sets. Both training and testing data sets are compared to yield the result of hardware performance level.

Consequently, the construction of the *affirmation* scheme is detailed. There are 2 goals to be achieved here. The environment setup for the 2 mechanisms is similar to the *optimization* scheme. Both need the TPC-H database to be built. The first objective is to create the stress-testing scenario, whereby the database logical reads and the CPU run queue parameters are exploited to create the stressed condition in the VM, for verification of SLA-bound transactions. The second target is to utilize the TPC-H queries to discover the resource threshold in the VM. It is to note that both *monitoring*

and *affirmation* models are capable of recognizing the threshold value. Nevertheless the former is taking the perspective from the end users' experience, whereas the latter is more hypothetical in nature.

The next chapter explains the next phase after the design phase elaborated in this chapter. The experimental results are produced and detailed analysis is conducted.

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1 Introduction

The prototyping results of the proposed models are detailed in this chapter. The results are critically analyzed, and their advantages and shortcomings are evaluated. At this point, the proposed resource management mechanisms can be exploited to complement existing commercially deployed tools in the same categories. For instance, the proposed *monitoring* model can depict the server resource utilization state from a new perspective, in relative to commonly deployed utilities in the IT industry, for instance *NIMBUS*, *Oracle OEM*, *Resource Monitor* by Windows and the multiple command-line interface (CLI) tools available in Unix and Linux. The *optimization* model can serve as the pre-requisite for comprehensive hardware health check, where the VM state is probed for abnormality, and the detection of fault is reported for subsequent extensive hardware scanning to take place. The *affirmation* model which relates to resource threshold discovery is able to provide a ballpark figure on the resource limit in the VM, before the thorough conventional stress testing is conducted using commercialized tools in case the computing capability of the VM is in doubt, for instance the HP *LoadRunner*. In the same *affirmation* scheme, the rapid creation of stress-testing scenario proposal provides premeditated stress testing, before the decision to engage the comprehensive conventional, yet cumbersome load testing is made.

The analysis on the data employed in the experiments is included. As the target of the research is to avoid access to real data in the database, the rationale on the choice of preferential data is circled around this security concern. In this case the choice of employing TPC-H data and queries, together with metadata extraction from real workload are discussed.

In parallel with the elaboration on the analysis for the 3 schemes, the potential future enhancements from this development point are also envisaged. These reviews strive to reveal the opportunities to improve the resource utilization in the virtualized environment, for IT administration as well as monetary benefits.

5.2 Data sets

5.2.1 Metadata from real workload

2 types of data sets are employed in the experiments conducted for this research on resource management. In the proposed *monitoring* scheme, the metadata in the real workload is scrutinized. The data collector runs periodically to gather the needed information, and stores them into a custom repository. Here, the data collector is a build-in mechanism in the database. The collection of input data in this experiment is carried out by the *Oracle Workload Repository*, where periodic snapshots can be taken to depict the database and VM states. The high level architecture of this utility is illustrated in figure 5.1. In this figure, it is shown that the database performance statistic is gathered in the shared memory, and can be displayed online via the *v$session* view. The rolling update of the statistic is also published in *v$active_session_history* view so that a more comprehensive interpretation of the database state can be obtained. Such information is only stored temporary in the memory, as and when the 2 views are updated with new performance statistic, the old data is no longer available. In order to capture the data for analysis, the MMON and MMNL background processes are responsible to capture these statistics in snapshots, and store them onto the disk for permanent storage. These snapshotting operations to collect and extract metadata did not incur significant I/O in the test bed when they were run in 1-minute interval; hence it is not a cause of concern for the experiment. For a week of collected data, the incurred storage size in the custom table is less than 10MB in size. This figure is relatively small

and does not significantly affect the overhead from the cost and administrative perspective. Other similar tools available in commercial RDBMS products are *SQL Optimizer* that associates with Sybase database, and *SQL Profiler* which can provide the needed information in SQL Server.

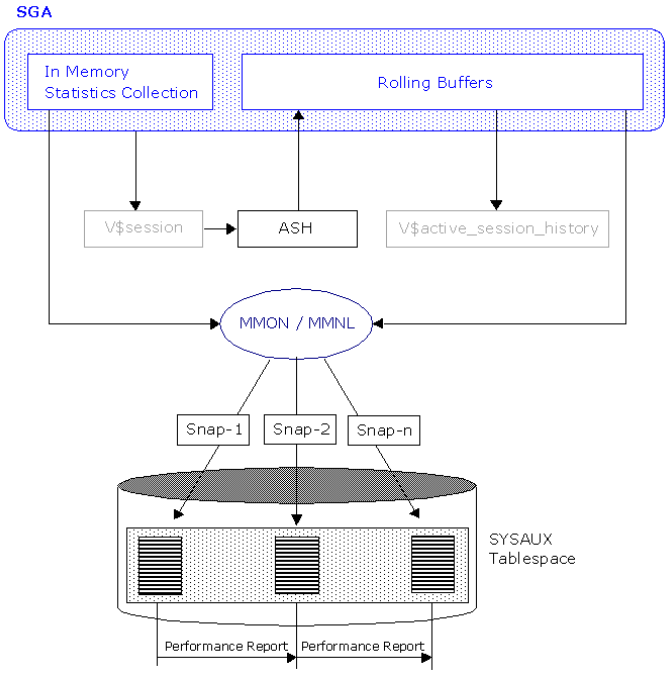


Figure 5.1: High level depiction of the Oracle *Workload Repository* engine. The in-memory statistic delineates the database state, and the information can be stored by enabling snapshotting as illustrated at the bottom half of the diagram. Adapted from (AWR, 2013).

From the database end, the collected metadata consists of DB CPU Time (DCT) and SQL Elapsed Time (SET). DCT denotes the execution time needed to process the particular SQL by the database engine. DCT is a subset of SET. In the experiments carried out in this thesis, SET signifies the total time needed by the VM and database to return query results to the terminals that initiate the queries. In other words, it is the Round-trip time (RTT) to process a SQL from the end users to database; and from the database back to the end users. The components of SET have been detailed in chapter 4. As elaborated before in chapter 3 and 4, the needed pre-requisite to this *monitoring* model, is to ensure that the processor component is the point of constraint. With this assumption and assurance, DCT and SET can comfortably be deployed in the proposed

monitoring model. In this case, the waiting time to process the SQL in the VM will accumulate in the SET region; however this waiting time is not included into DCT. With such scenario, the queuing model can be easily depicted, as the magnitude of difference between DCT and SET at particular CPU run queue denotes the number of database processes queued in the VM. To safeguard the legitimacy of the collected metadata, during the snapshotting phase of data collection, the noises in the VM must not be dominantly affect the number of queuing processes. However in real situation, even in the most scrutinizing condition, it is often unavoidably that ad-hoc and unexpected operating system (OS) processes are triggered when the database snapshots are taken, either intentionally or unintentionally. For instance, there could be request to perform ad-hoc OS backup as the pre-requisite for a system change. Or in another instance, the overrun daemon processes might incur unnecessary overhead to the VM. To cater for such anomalies, the Fourth-Spread *(fs)* method is employed to remove the outliers in the *monitoring* model prior to the data analysis phase. This method segregates the graphical data points into 4 quadrants. Only the data points in the nearest 2 quadrants are deemed valid to be manipulated to serve the modeling purpose. The implementation details of the method are illustrated in section 4.3.3. Nevertheless, the accuracy enhancement of this monitoring model can be significantly boosted when the noises at the OS level are reduced. Hence the careful selection of snapshotting timeframes should be emphasized to avoid duration when OS maintenance jobs are taking place.

5.2.2 Synthetic workload – TPC-H benchmark

The second category of the data sets is utilizing TPC-H benchmark as input into the *optimization* and *affirmation* models. As this benchmark is not specifically fabricated for particular RDBMS, it can be easily deported to other database platforms in case needed. However, the main criteria for choosing this established benchmark in this

research is its wide adoption by multiple RDBMS products. Because it can be easily setup and configured in a great variety of databases, the experiments carried out in this thesis, together with the deduced analysis are applicable generically to other database flavors. The same design methodologies elaborated in chapter 4 can be quickly and effortlessly adapted to other non-Oracle database platforms. This benchmark comprises of 8 tables and 22 queries. The relationship between the tables is illustrated in figure 5.2. The information on the queries can be obtained from Appendix F.

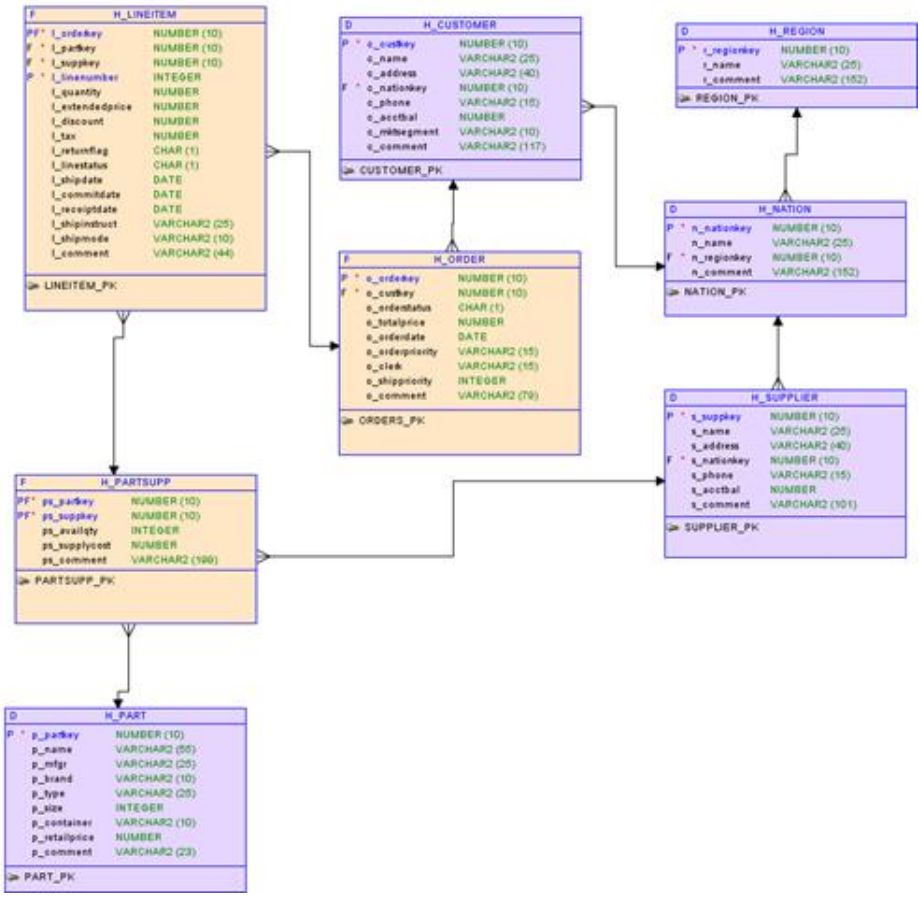


Figure 5.2: TPC-H data model. Adapted from (Kocakahin, 2010).

Furthermore, as the data and queries are standardized across all databases, relative comparison in performance and throughput capability can be performed. Such facility is not significant in this research. However it can provide a magnified insight to the consumers, regarding the expectation of performance and throughput on similar workload that is running in different RDBMS platform. For instance, if the TPC-H

queries are needing *x* amount of longer response time to be processed by database running on RDBMS type A during the experiments as compared to RDBMS type B, the same magnitude of slowness should be expected when the workload is running in user database that is serviced by RDBMS type A, on the same VM configuration.

The choice of TPC-H benchmark in the experiments is also due to the fact that the queries can adequately load the VM to the resource threshold level required by the tests. There are other choices to synthesize this hypothetical workload, for instance, by employing TPC-W or TPC-C benchmarks. However it is discovered that these 2 benchmarks are more suitable to be utilized for studies on measuring transactions' throughput. They are not as appropriate relatively to be utilized to construct the *optimization* and *affirmation* models as their queries are relatively lighter to be used to load the VM. The same observation is revealed in the TPC website. TPC-H benchmark is fabricated to mimic data warehouse environment where heavier queries are frequent. On the other hand, TPC-C and TPC-W benchmarks are of OLTP type, which typify the general nature of web-based OLTP transactions. TPC-H is also a choice here in this research as it contains a variety of queries that produce the execution response time, ranging from 1s to 150s. Hence, depending on the size of the VM, different queries can be deployed according for the experiments.

The TPC-H data occupies 5GB of SAN space, which is relatively small in comparing to the size of common commercially deployed user databases in today's storage frame. As one of the issues in deploying database transactions to the Public Cloud is the distance of I/O between the data and computing node, 5GB of data can be maneuvered easily in case the study on the effect of data-to-computing distance is to be conducted. The future works illustrated at the tail end of chapter 4 elaborates on taking the physical I/O parameter into the fabrication of stress-testing scenario, using *linear programming* and *simplex method*. In this case, the influence of disk I/O will be significant. With TPC-H

data, the distance effect can be conveniently measured in case Public Cloud VM is to be deployed as test bed.

In most of the TPC-H queries, execution parallelism has been built into them, which depicts the common VM configuration of 4-8 virtual processors. Such parallelism effect is not studied in details in this research. However it can be potentially applicable in future work to refine the stress-testing scenario creation in the *affirmation* scheme. At this point, extrapolation on the experimental results is not workable, in order to produce the benchmarking array. However this parallelism effect is suspected to be the cause of the irregularity in predicting the memory reads/s of the iterative execution of the queries in the *affirmation* model. The default parallelism on the queries is not modified in this research; nevertheless they should be altered in subsequent research to match with the virtual processors' configuration, to predict the memory reads/s parameter.

The benefit of using TPC-H benchmark as compared to customized workload is the easy reference of this benchmark across different databases. During the initial stage of the research, it was envisaged that potentially the real data can be scrambled in order to achieve the objective of not allowing visibility to the real sensitive data to safeguard the data security. However, such option has a few shortcomings as follows:

1) In order to scramble the real data, manual intervention is needed to execute the scrambling scripts. Such effort is not beneficial as depicted in following points.
2) The scrambled data does not necessarily guarantee the same performance characteristic when the same set of transactions is exerted against it. This can be explained by considering the following SQL:

```
select department_name, sum(salary) from departments
where num_employee>100;
```

If this statement yields total 300000 memory reads for a duration from the real data in the memory, after the data is scrambled, the logical reads could be of any value but 300000 as different execution path is taken by the RDBMS engine due to the change in data values that changes the predicate clause.

3) Most user databases are relatively larger than the TPC-H benchmark. Hence it is more cumbersome in deployment and management. TPC-H benchmark data is generally smaller and portable; hence in case the workload in the VM needs to be migrated to another VM, the migration of the TPC-H data can be easily performed without much constraint.

4) Customized workload cannot be easily published as benchmark to the wide industry. However, TPC-H data and queries are already widely understood and serving various IT functions. Hence the proposed models in this thesis can be adopted easily, without prior understanding on the testing data.

The following sections illustrate the application of the discussed data sets into the proposed models.

5.3 Resource utilization *monitoring*

5.3.1 Environment for the experiments

To outline the details of the *monitoring* scheme, a considerably large workload is needed. During the development stage of this model, there was not a mission-critical application made available with enough SQL transactions that is hosted on Cloud environment for illustration purpose. However there was one such ideal workload available on a single Sun Solaris server. The experiment was conducted against this workload, where the outcome can be proliferated to database operations on Cloud VM without much variation. The data for the experiment is gathered from a Sun Solaris server, powered by 4 Sun Solaris SPARC64-VII CPU with 4-core architecture, 64GB

RAM and external SAN running on ZFS File system. The application runs on SAP ERP software, on ECC6 HRM Module. The application is OLTP in nature, servicing Human Resource Management System for a large organization. The backend is running on a single instance Oracle 11g database. In this scenario, a week of data on actual transactions is collected, with Server Load taken as average in 1-minute interval. SET and DCT are collected in tandem with the 1-minute interval timeframe. In this case the explanation of the proposal is simplified by having only 1 database running in the host.

The experimental result is exhibited in figure 5.3. In this graphical illustration, all 3 variables – CPU run queue (C), SET ($\tilde{S}$) and DCT ($\tilde{S}'$) are quantitative. When the data points of these 3 values are mapped, the scatter graph is generated as in Figure 5.3. As the correlation between C and $\tilde{S}$, as well as C and $\tilde{S}'$ are linear as explained in section 2.4.3.1, regression lines are drawn mathematically. They are extrapolated on the cluster of scatter plot data, to statistically describe the trend of the SET and DCT.

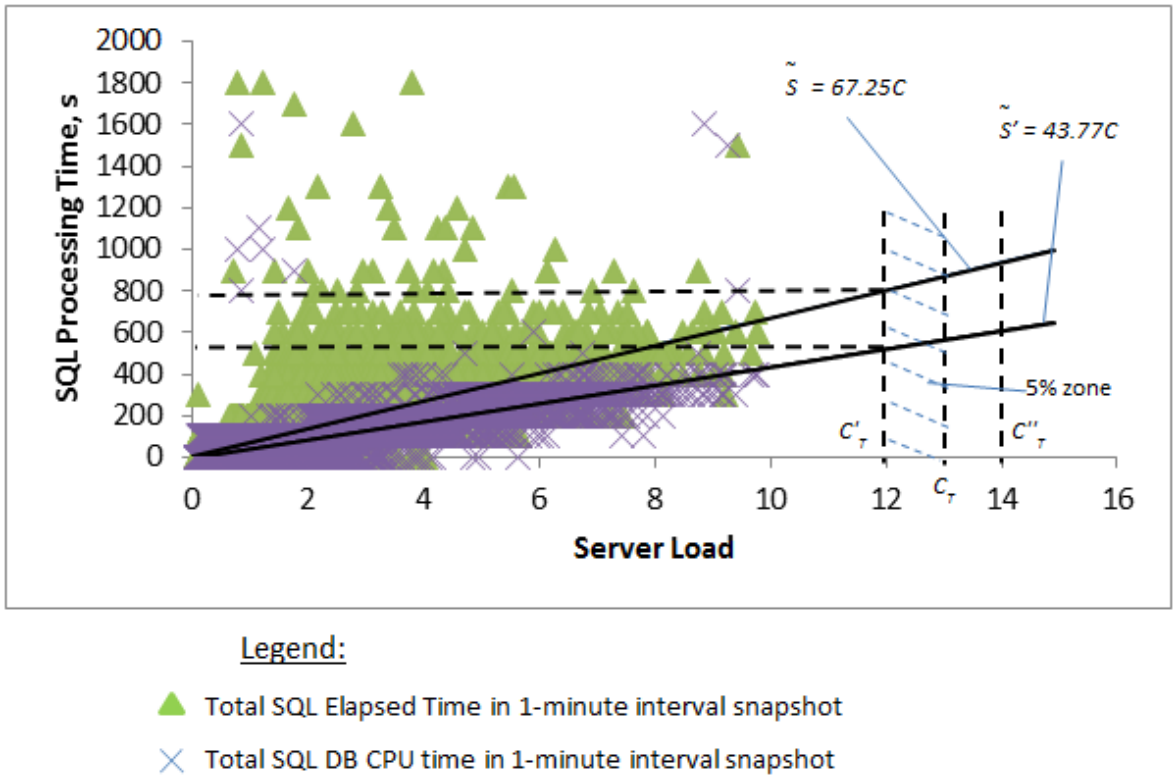


Figure 5.3: Experimental results that show relationship between $\tilde{S}_i$(SET), $\tilde{S}'_i$ (DCT) *and* C_i (CPU run queue size).

5.3.2 Derivative parameters from the experiments

To arrive at the 2 lines mathematically, the *Linear Regression* methodology is employed. Take SET as example, it is defined as $S_i = xC_i + b + \varepsilon_i$. However in the case of this experiment, value b is assumed 0 as observed from the actual host itself, that even with multiple background daemons running, the server load is close to 0 and hence this variable is negligible. This value can be safely assumed as in this case, the server resources are abundant. However if the hardware resources in the server or *VM* are restricted, for instance if there is only a single CPU and 1 GB of physical memory, value of b could be 1 or 2 as a result of the system overhead, and it needs to be acknowledged in the formula. In this case the CPU run queue will be sensitive to the constantly-running overhead processes in the server or VM. Nevertheless, such scenario will only complicate the formulation in the algorithm, without achieving significant desired result. The explained model will work properly in the environment with larger

resources, to curtail the system overhead; however if the overhead is large and unavoidable, a workload pre-check mechanism can be incorporated, to ensure a robust workload control plan is in place. The precaution in the *monitoring* model is to avoid and disregard timeframe when the system is running non-database related overheads before inputting the workload data into the model. In defining the legitimate workloads for input to the model, it is assumed that IT organization has a well-designed maintenance window to cater for unavoidable system overhead, especially the backup operations, where business transactions during this timeframe are kept to minimum.

To explain the case for the experiment in figure 5.3, the SET is having the equation $S_i = xC_i + \varepsilon_i$. The linearly fitted value, $\tilde{S}_i$ is the value fitted exactly on the regression line, and is denoted as $\tilde{S}_i = xC_i$. Hence, the residuals, $\varepsilon_i = S_i - \tilde{S}_i$, are the differences between the actual and fitted values of SET. This variable is not elaborated for the discussion in this thesis, but will serve as a critical component in subsequent work in developing an adaptive system to reduce the noises in the system. The requirement here is only to calculate the value of x to fulfill the requirement here. Using *Least Squares Derivation* method as explained in section 2.4.3.3, with N number of data points, the value of x is obtained as

$$x = \frac{\sum_{i=1}^{N} C_i \tilde{S}_i - \frac{\sum_{i=1}^{N} C_i \sum_{i=1}^{N} \tilde{S}_i}{N}}{\sum_{i=1}^{N} C_i^2 - \frac{(\sum_{i=1}^{N} C_i)^2}{N}} \quad . \qquad (1)$$

With this, the regression line is plotted using $\tilde{S} = xC$, and similarly for DB time, $\tilde{S}' = yC$. To measure the representability of the regression lines to the data points, we use the correlation coefficient (r), defined as

$$r = \frac{\frac{\sum_i^N (C_i - \bar{C})(\tilde{S}_i - \bar{\tilde{S}})}{N}}{\sqrt{\frac{\sum_i^N (\tilde{S}_i - \bar{\tilde{S}})^2}{N}} \sqrt{\frac{\sum_i^N (C_i - \bar{C})^2}{N}}} \quad . \qquad (2)$$

With, $\bar{C} = \frac{1}{N}\sum_i^N C_i$ and $\bar{\tilde{S}} = \frac{1}{N}\sum_i^N S_i$.

r is confined to value between 0 and 1 in our case. 1 denotes that there is a perfect linear correlation between *C* and $\tilde{S}$, while 0 shows no correlation. Intermediate values show partial correlations. This value of *r* will be utilized later to gauge the accuracy of the graph in figure 5.3. Another parameter, $\Delta S = (\tilde{S} - \tilde{S}') / \tilde{S}' * 100$ is also needed in the model. The details explanation for this parameter is shown in section 4.3.2. Δ*S* corresponds to C'_T. This delta of $\tilde{S}$ and $\tilde{S}'$ can be used to gauge if the host condition is still viable for optimal database transactions. During steady state database operations, if $\tilde{S} = xC$ becomes steeper, Δ*S* is then reached for Server Load < C'_T. The reason could due to the fact that the physical reads or memory reads in the database are not efficient. This indirectly indicates that either the I/O subsystem is not functioning optimally or the database cache is not sufficient after prolonged database operations where change in the data volume has occurred. The noises from the operating system can also contribute to this, for instance new auditing daemon could be running in the host, additional host monitoring utility is running etc. The noises ideally are undesired, for a mission critical application running stead-state operations. However in real live environment, system overhead is inevitable. For instance, the VM and database backups will cause significant overhead and these cannot be ignored. In this case a maintenance window is defined, and the workload input will avoid this timeframe when feeding into the algorithm, to preserve the model accuracy.

In case the ΔS is reached when Server Load < C'_T, appropriate measures need to be taken, i.e. fixing the host environment or increase database cache. If all has been done but value of x is still steeper than before, a new C'_T will need to be defined. In this case the new C'_T is the Server Load value corresponds to where ΔS is. It is to note that as values for C_T & C''_T stay as constant here, hence the C'_T & C_T gap is enlarged. When this happens, the probability of transactions to fall into the 5% zone increases. The 5% zone is a hypothesis figure, and it should be adjusted appropriately based on particular application's SLA.

When block of new hardware is added to existing *VM*, the new resource threshold point is rediscovered via the *affirmation* model explained in section 4.5.2.2. Subsequently the *monitoring* model is constructed again to monitor and determine for subsequent need of hardware provisioning.

5.3.3 Experimental data

There are a total of 10621 samples (total collection of the data points as in Figure 5.3) gathered in the 1 week period for each set of SET, $\tilde{S}$ and DCT, $\tilde{S}'$. As mentioned their relationships are $\tilde{S} = xC$ and $\tilde{S}' = yC$ respectively, and by calculation using the data points' values, the gradients are x=67.25 and y=43.77. Hence as seen in Figure 5.3, 2 strong positive regression lines are drawn. There are outliers in the graph, and they are understandably to be caused by noises in the server outside the control of the RDBMS. These can possibly cause by the auditing processes in the OS which spike occasionally while the application transactions are running, File System backup that incurs I/O contention and monitoring daemon to name a few. To gauge the accuracy of the regression lines, correlation coefficient, r, is calculated. Using equation 2, the value r = 0.72 for the regression line on SET, and r = 0.78 for the regression line on DCT. These 2 values show that the fit of the 2 linear models is fairly acceptable. In other words, it

can be assumed that these noises are not affecting the correlations too much. For more accurate plots, these noises will need to be investigated and fixed at OS level, or if ever desired, the outliers in Figure 5.3 can be excluded to increase the accuracy of the regression lines. With these equations, the limit when the server is hitting resource constraint can be further derived.

2 values from the output of *affirmation* model during pre-cutover or right after change of hardware configuration are to be noted, before the database goes into steady-state production mode. They are C'_T & ΔS. The initial C_T is set at 13 and ΔS = 55% respectively. After about a year running into steady-state production mode, the C_T value reduced to 12 with ΔS stays at 55%, as depicted in Figure 5.3.

To represent these 2 properties properly, Fuzzy Logic is employed, as illustrated below:

Step 1: Determine when to examine the host environment and adjust C_T, using *Fuzzy Computing with Words* (Zadeh, 1996):

If SQL elapsed time is very much higher than SQL DB time, the host environment is near suboptimal condition.

With *Fuzzy Implication method* (Alavala, 2008):

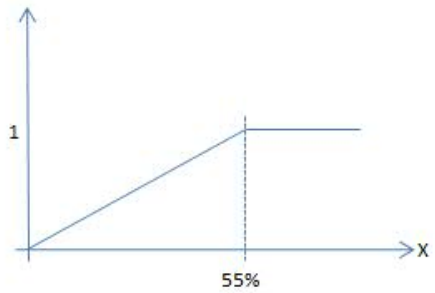


Figure 5.4: Membership Function for ΔS, $A(u)$.

Figure 5.4 shows membership function for ΔS, *A(u). A(u)* is

$$A(u)=\begin{cases} 1 & if\ u \geq 55, \\ \frac{u}{55}\ if\ 0 \leq u \leq 55, \\ 0 & otherwise. \end{cases} \quad (3)$$

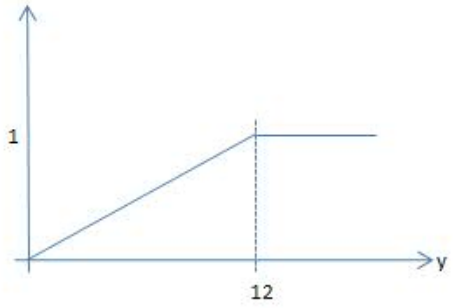


Figure: 5.5. Membership Function for *C, B(v).*

With *u=55*, corresponding Server Load limit, C'_T is obtained, as in Figure 5.3.

Figure 5.5 shows membership function for Server Load, *B(v)* in the server. *B(v)* is

$$B(v)=\begin{cases} 1 & if\ v \geq 12, \\ \frac{v}{12}\ if\ 0 \leq v \leq 12, \\ 0 & otherwise. \end{cases} \quad (4)$$

Subsequently, the constraining relation, *R = A(u) ⇒B(v).*

Step 2: Determine if database transactions need additional hardware:

If more data points fall between C'_T & C_T, trigger point for hardware planning and provisioning is near.

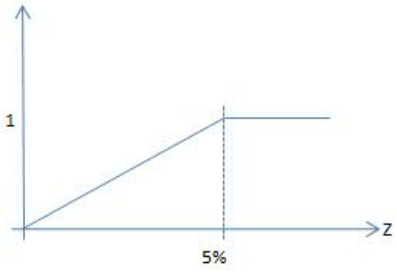


Figure. 5.6: Membership Function for *ρ, C(w).*

Figure 5.6 shows membership function for density of data points, ρ, between C'_T & C_T.

$$C(w)=\begin{cases} 1 & if\ w \geq 5, \\ \frac{w}{5} & if\ 0 \leq w \leq 5, \\ 0 & otherwise. \end{cases} \quad (5)$$

The membership function *A(u)* is obtained from the output from the *affirmation* model, or initial Load Testing as when ΔS is defined. Subsequently from the limit of *A(u), B(v)* is determined during steady-state operations. Using limit of *B(v), C(w)* is derived.

5.3.4 *Monitoring* model's accuracy and system overhead

The snapshot interval to obtain the data points is set to 1 minute in the experiment. It is worthy to note that the smaller the interval, the more accurate the data is. Caution needs to be taken here on the workload to collect $\tilde{S}$, $\tilde{S}'$ and *C* values, as these data computation in the database should not incur too much overhead. In system with not as powerful hardware, 1-minute interval could incur high overhead to the host environment. In contrast when there is more resource available in the host, sampling interval can be small. This can be represented by Fuzzy rule in the form of:

R: *If <x is P>, then <y is Q>*. This is translated to *If <Server load is low>, then <sampling granularity can be small>*.

Then with Fuzzy predicates *P and Q* as Fuzzy sets on *U = domain of x, V = domain of y*, define,

P(x) for '*x is P*' and *Q(y)* for '*y is Q*', and define,

$T(R) = T[P(x) \Rightarrow Q(y)]$ for every *x* in *U* and every *y* in *V*.

Using *Mamdani implication* (Ganesh, 2008) which is appropriate in this case,

$$T[P(x) \Rightarrow Q(y)] = min[P(x), Q(y)].$$

With this visibility, the appropriate overhead values of *P* (Server Load) and *Q* (sampling granularity) can be brought into equilibrium.

5.3.5 Workload characteristic

5.3.5.1 SQL tracking

During the tenure of the database life cycle, it is imperative to keep the DCT of all transactions as close as possible to the initial deployment of the application. In other words, the line $\tilde{S}' = yC$ as in Figure 5.3 should not change ideally. To do this, the SQL must be tuned and run as optimally as possible during the development phase, before production deployment. To explain the SQL verification mechanism, define an array, *B*, which has 30-minute interval in each of its element. Depending on how long the data capture operation is going to run to properly represent all potential SQL in the databases, there is *q* samples in *B*, $B=[\ u_1, u_2, \ldots\ u_{i,\ldots,}\ u_q]$. Then assume there are *n* numbers of databases running in the host, $DB=[db_1, db_2, .. db_j \ldots, db_n]$. Take 1 30-minute snapshot, u_1 to represent activity in other snapshots, and define

s_1 = collection of SET of top *m* number of SQL in all *n* databases, running in u_1. The top *m* SQLs are ranked in descending order by total elapsed time. s_1 is a collection of SQL ID.

Hence, s_1 is $s_1 = [ss_1, ss_2, ss_3 .. , ss_k \ldots, ss_m]$,

where,

ss_k = SQL elapsed time on a SQL *k* that runs in database db_j with *y* iterations in the 30-minute interval, defined as $\sum_{i=1}^{y}\binom{y}{l}\mu_k$, where μ_k = mean elapsed time of the SQL.

Top *m* SQLs are defined as SQLs that exceed *x* duration of runtime including all their iteration in u_1. Top *m* SQLs are dominant resource consumer in the host.

The accurate way to gauge the effectiveness of a SQL is to compare its actual μ with the benchmarked value, in this case the minimum of μ found in all the legitimate gathered data points in figure 5.3. So for ss_k, the minimum of μ is labeled as $\mu_{k\text{-}min}$. For each SQL ID in s_i, the minimum μ is stored in an array, $U = [\mu_{1\text{-}min}, \mu_{2\text{-}min}, \mu_{3\text{-}min} .. , \mu_{k\text{-}min} \ldots, \mu_{m\text{-}min}]$. Hence, data in *U* is to be benchmarked when SQL tuning is taking place.

5.3.5.2 SQL optimization

In real situation, there are un-optimized transactions that disguise the actual need of computational power. SQL must not be allowed to run wildly. The un-optimized condition can be attributed to following few problems and potential solutions:

1) The change in data volume which requires update on the tables' statistic.
2) The change in data structure which requires reevaluation on the SQL context.
3) Accidental deletion of indexes which changes the SQL execution path entirely.
4) Introduction of new codes into the environment which was not tuned properly in prior.

Following section further explains the situations on the behavior of these SQL, particularly on the change on data volume as this is potentially the highest possible occurrence in the database.

For the same SQL which runs multiple iterations, either via bind variables or literal values, its μ, which is the average execution elapsed time, may change but the execution plan stays the same. Few scenarios could lead to this, for instance if the data involved in the SQL increased significantly and statistic has not been gathered in time, or if there is skewed histogram in the data resulted from data change. Another situation that can lead

to this is when there is high resource contention in the VM going beyond CPU run queue threshold. These are represented in Figure 5.7.

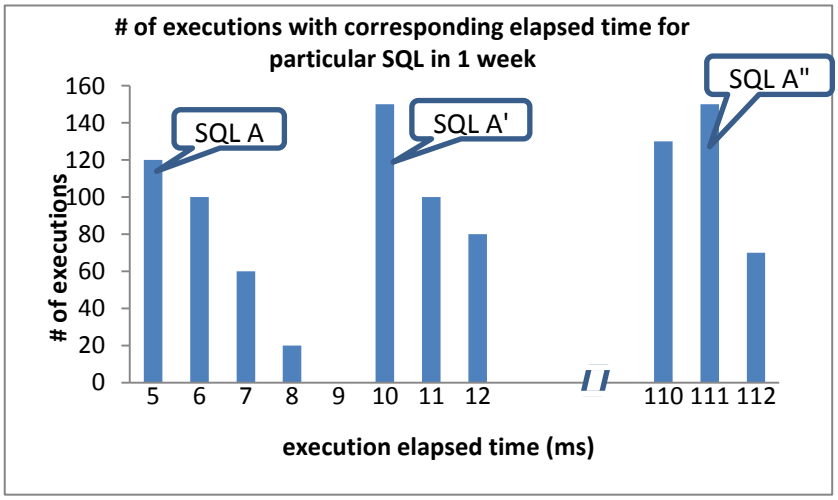


Figure 5.7: Runtime Variation of Particular SQL in 1 Week.

Figure 5.7 shows SET of a SQL executed in each 30-minute segment in array *B*. There are 10 occurrences of the SQL execution in 1 week. *SQL A* was the original statement, optimally tuned, and there is no hardware contention in the server. To explain this further, there are few key aspects to define optimally tuned statement in this case. During end users acceptance test, the buy-off transaction response time could be set as benchmark. Then when the database is running in steady-state production mode, the RDBMS engine can self-tune the SQL. There is also situation where particular SQL is intentionally forced to run on particular execution path to maintain desired response time.

A' is a result of data being added to the tables involved, and it goes undetected by the RDBMS. *A''* illustrates the scenario when data is added significantly to the tables used by the queries before tables' statistic is gathered, or necessary indexes have not been considered to accommodate the new data. In another scenario there can be resource contention in the VM that results in *A''*. Another scenario which is not shown in figure 5.7, is that *SQL A* changes its execution plan, as a result of accidental drop of an index

in a table or sudden surge of cache memory consumption due to sudden increment of table data. These adversely result in excessive physical reads and the μ diverges significantly.

Above are a few situations that affect the accuracy of the model. These SQLs need to be tuned before resource capacity tracking model can report the resource utilization state accurately. As defined in section 5.3.4.1, $U = [\mu_{1\text{-}min}, \mu_{2\text{-}min}, \mu_{3\text{-}min} .. , \mu_{k\text{-}min} \ldots, \mu_{m\text{-}min}]$. The data in this array is used for the purpose of baselining and tuning the involved SQLs.

5.4 Resource utilization *optimization*

5.4.1 Environment for the experiments

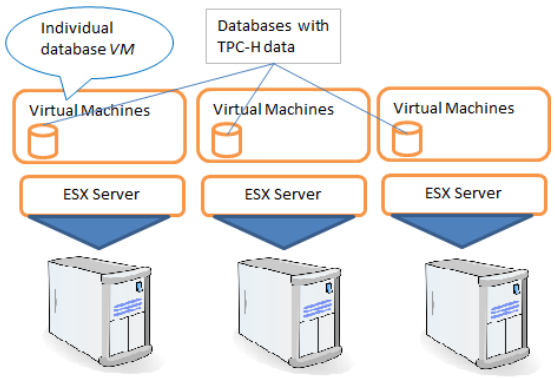


Figure 5.8: *Parallel Database* hosting using *VMWare* Cloud Virtualization Infrastructure. The *optimization* and *affirmation* models are built on the TPC-H database depicted in the diagram.

The VM utilized for the experiments in this research is constituted of the components as illustrated in figure 5.8. The proof-of-concept repository is an Oracle database. Oracle is the RDBMS of choice here, as it offers full-fledged SQL optimizing feature, via the matured *optimizer* technology. It provides the database transactions with many complete SQL optimization technologies; hence the need for SQL tuning effort in all the testing scenarios can be reduced. The *VM* is running on Suse Linux operating system. The

captivated CPU run queue length, or server load values obtained from *'uptime'* command in the OS are the core input for the *optimization* and *affirmation* models.

5.4.2 Experimental results

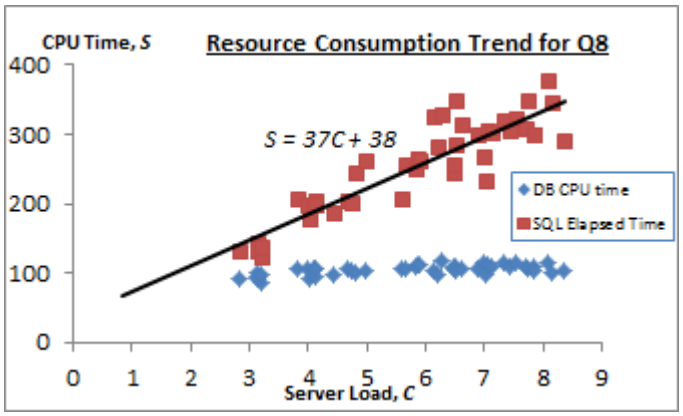


Figure 5.9: The expected output from the *optimization* model. The obtained gradient and y-intercept are the parameters to be compared between the baselining and production phase, as explained in section 4.4.2 and 4.4.3.

The baselining experiments' output is as illustrated in figure 5.9. Appendix E contains the other similar baseline data, produced by different TPC-H queries. The core value of these experiments lies in the linear relationship between SET, *S* in the database and the CPU run queue size, *C* within the resource threshold point. As the interest is to discover the gradients, M_B and y-intercept, Y_B of the linear plots, these regression lines computed during the baselining phase of VM provide the baselines for subsequent hardware performance analysis.

Each set of test which comprises of different combination of TPC-H queries has different values of M_B *and* Y_B. This is because the SQL processing has dependency not only on the number of CPU, but also the logical and physical I/O reads. In real production phase, only 1 to 2 baselines are adequate for subsequent comparison. These regression lines are equitably formulated by following equations:

The gradient of the regression line:

$$M=\frac{N\sum_{i=1}^{N}C_iS_i-(\sum_{i=1}^{N}S_i)(\sum_{i=1}^{N}C_i)}{N\sum_{i=1}^{N}S_i^2-(\sum_{i=1}^{N}S_i)^2}.$$

The y-intercept:

$$Y=\frac{(\sum_{i=1}^{N}C_i)(\sum_{i=1}^{N}S_i^2)-(\sum_{i=1}^{N}S_i)(\sum_{i=1}^{N}S_iC_i)}{N\sum_{i=1}^{N}S_i^2-(\sum_{i=1}^{N}S_i)^2}.$$

The correlation coefficient:

$$r=\frac{\sum_{i=1}^{N}S_iC_i-\frac{1}{N}(\sum_{i=1}^{N}S_i)(\sum_{i=1}^{N}C_i)}{\sqrt{\left[\sum_{i=1}^{N}S_i^2-\frac{1}{N}(\sum_{i=1}^{N}S_i)^2\right]\left[\sum_{i=1}^{N}C_i^2-\frac{1}{N}(\sum_{i=1}^{N}C_i)^2\right]}}.$$

The correlation coefficient, *r* is used for performance evaluation purpose. Its details have been elaborated in section 5.3.2.

5.4.3 Potential deviation from the testing data sets

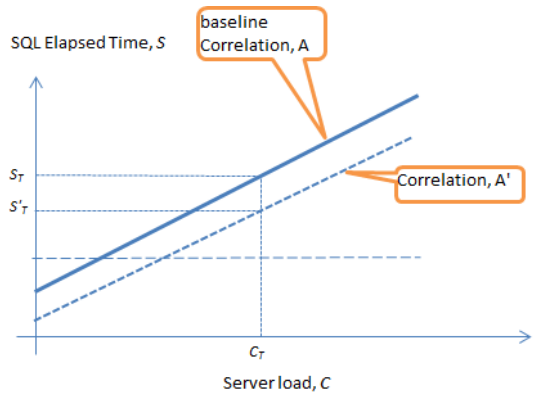


Figure 5.10: Potential change in linear correlation between *S* and *C*.

Figure 5.10 shows linear regression lines of the resource state in a situation in the VM. Correlation *A* is the Fitted Regression Line obtained from the training data sets of particular set of *TPC-H* query, from the control system, in other word, the baseline. If the *VM* shows *condition A'* from the testing data sets, which are the data obtained from the tests during the production phase, where y-intercept differs from baseline after running operations for a while, it signals that the capability of the VM has deterioated.

This could be due to the persistent noises in the OS or partial hardware malfunction. For this, $\Delta s = \frac{S_T - S_T'}{S_T} \times 100\%$ is defined.

Using *Fuzzy Computing with Words* (Zadeh, 1996) concept,

- *If* Δs *is large, the OS and hardware condition needs to be examined.*

Correlation *A"* in Figure 5.11 shows another deviating condition. The fluctuation in the gradient can signify hardware or OS issue. For instance, there are significant irregular noises in the OS, the CPU is not able to access the second core in a dual-core machine, memory shortage due to failure in DIMM or increased I/O time resulted from breakdown in any of the SAN components. The gradient of the testing data set, m_j is derived from same *TPC-H* query sets, and it has less positive value than M_B, which is the benchmarked gradient obtained from training phase. Again, using *Fuzzy Computing with Words* concept,

- *If gradient and y-intercept deviate much from baseline values, the OS and hardware need to be examined.*

The above explained theories assume strong linear correlation between the SQL Processing Time, *S* and server load, *C*. However this might not be the case in actual production system. In this case, the correlation coefficient, *r,(D. G. Kleinbaum, L. L. Kupper, K. E. Muller, & A. Nizam, 1998)* is employed. It is a barometric measurement of the linear association between the data points and the Fitted Regression Line from the baseline data. In this case, *r* will vary between 0 and 1, with value nearer to 1 denotes stronger linear correlation.

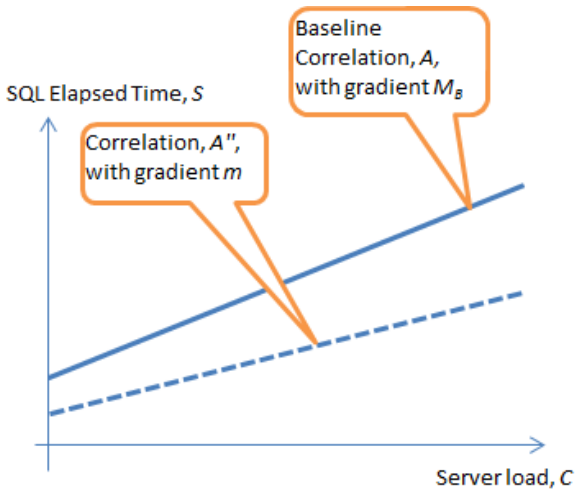


Figure 5.11: Potential change in linear correlation between *S* and *C*.

Using *fuzzy* definition,

- *If correlation coefficient, r_j is less positive, the OS and hardware need to be examined.*

Figure 5.12 shows a condition where the underlying storage is going through a backup process. In this timeframe, the host's environment is not conducive for any transaction, as uncharacteristic performance results are expected due to inconsistent I/O subsystem performance during the backup snapshotting. This behavior is shown in this graph, where erratic data points are collected from the test. Hence, the consistency criterion is voided in this case.

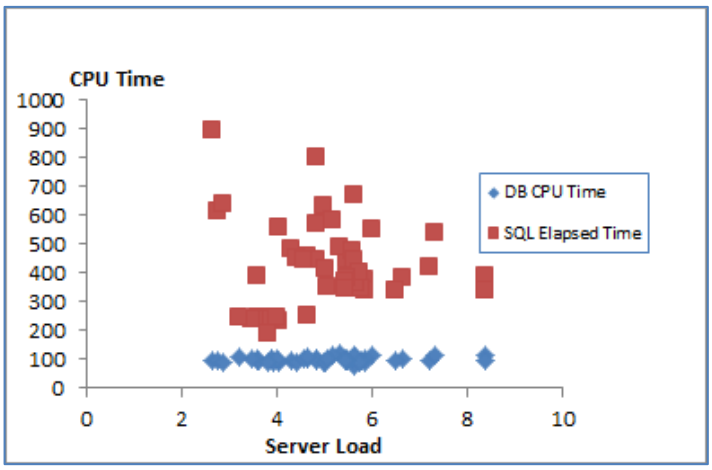


Figure 5.12: Erratic behavior of hardware performance during backup process.

5.5 Resource utilization *affirmation*

The significance of the affirmation scheme lies in its ability to rapidly create the stress testing scenario in the VM. Figure 5.13 depicts the result in the benchmarking stage from TPC-H query #2. The complete result of the proposal in the *affirmation* scheme is illustrated in Appendix H.

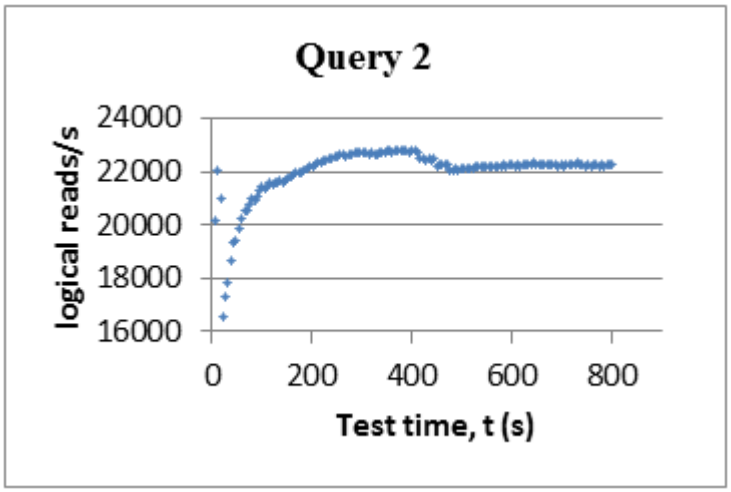


Figure 5.13: Testing result of the *affirmation* model. The output from this test is included into the benchmark array. Sunsequently the benchmark is employed as input to the stress testing scenario.

In the experiment conducted in figure 5.13, the maximum parallel execution frequency of the query, f_T was maintained at 8, hence producing CPU run queue length of 4 in average. These experiments are conducted in a SUSE Linux VM, provisioned with a quad-core Intel processor, with 1GB of memory allocated to the Oracle database cache. In ideal case, the VM should be provisioned with more shared memory, so that the shared memory in the database can contain all the TPC-H tables. With these tables pinned into the cache, the incurred physical reads from the queries become insignificant; hence the proposal here becomes more accurate. This can be done if more than 5GB of cache is available, where practically all the tables in the TPC-H benchmark are pinned. Nevertheless, the experiments carried out in this proposal strive to illustrate the needed components and the relationship between each. It is noteworthy that

regardless of the database cache size, logical reads values for the individual query remain the same, with the difference lies in the runtime of the queries.

The experiments were conducted for all the 22 queries, producing the outcomes that are recorded in a 2 dimensional array, *A*, where the memory reads/s (MR) values are arranged in ascending order.

A[Query][logical reads/s] = {Q16, Q9, Q13, Q21, Q3, Q7, Q5, Q11, Q1, Q22, Q4, Q10, Q14, Q6, Q18, Q19, Q20, Q12, Q15, Q17, Q2, Q8}, {7000, 11000, 10000, 10000, 11000, 11500, 12500, 12500, 13000, 14000, 14500, 14500, 15000, 16000, 16000, 16500, 16500, 17000, 17000, 19000, 22500, 16000};

Some results exhibited in Appendix G depict the testing carried out from combined queries. The maximum run frequency, f_T is maintained at 8, hence each iterative run of queries is maintained at 4. This produces run queue length of 4 in average.

It is observed that when queries are combined in the test, MR is a function of the query runtime. For example, Q8 was taking 63s to complete each run cycle, and Q9 took 76s to complete. Individual result for Q8 yielded MR value of 16500, while Q9 produces 11000 of MR. So, the MR value for this combination is calculated as:

$$\left(\frac{t_i}{\sum_i^n t_i}\right) MR_i + \left(\frac{t_{i+1}}{\sum_i^n t_i}\right) MR_{i+1} + \cdots + \left(\frac{t_n}{\sum_i^n t_i}\right) MR_n \approx MR_T,$$

where,

t_i = runtime of query *i*.

MR_i = Individual query's memory reads/s value.

MR_T = memory reads/s value from baseline load testing OR actual production scenario.

n = number of involved queries.

Hence,

$$\left(\frac{63}{63+76}\right) 16500 + \left(\frac{76}{63+76}\right) 11000 \approx 13500.$$

MR value of 13500 matches the experimental value in Figure 5.14.

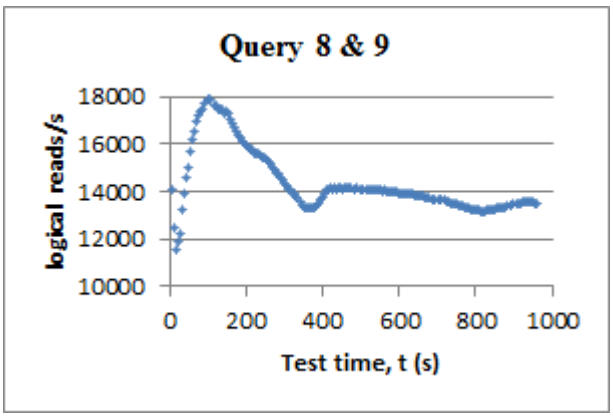


Figure 5.14: Testing result with the combination of TPC-H queries. Such output can also be predicted from the output of individual run from each query.

So, another array, *A'* can be defined as:

A'[Queries][logical reads/s] = {Q2+Q6, Q8+Q9, ...}, {16500, 13500, ...};

and,

$A \cup A' = A''$ which produces a larger repository of MR. The value pairs are to be chosen for VM loading.

To validate the practicality of the MR values in *A''*, we benchmark a real production Human Resource application, serving *Employee Self-Service* and *Manager Self-Service* functions. The database for the application is also running on a VM with a quad-core processor, with 3GB of database cache allocated. The average run queue length is about 4, which is taken as the C_T in the test. For this database, the total logical reads in 1 hour amounted to 27,912,822, which results in 11630 in MR value. Hence, it falls within the boundary of MR values in *A''*. The performance of the triggered transactions is satisfactory at this level as according to the end users. Hence in order to simulate the high load scenario in this *VM*, *TPC-H* query #7 can be employed, with maximum run frequency, f_T set to 8 that yields an average run queue of 4.

5.5.1 Future work

As the benchmarking stage requires quite substantial amount of time in order to produce the benchmarking array, it will be of great value if the experimental result obtained from a particular hardware configuration, can be extrapolated in tandem with the resource allocation changes in the VM. From another perspective, it is also interesting to find out if it is ever possible to derive the value of MR from 1 query to another.

Another point to note is that the server load value, C_T is taken loosely as the limit when the VM hits its resource limit, but it can be assigned any value which can be changed variably with ease in the tests depending on the application choice, as some transactions' response time is relaxed and queuing on CPU resource is allowed. Nevertheless the maximum limit on run queue size is ideal only when there is no process queues on the CPU. Hence, as the CPU run queue value is taken as constant during the experiments, it will add great value to this model if extrapolation can be performed on the benchmarking stage to discover the MR values, from 1 CPU run queue size to another.

In this thesis, only the memory reads criterion is experimented. For physical reads, there is no benchmarked DML SQL to be used. Nevertheless these can be fabricated by forcing repetitive reads operations on large tables' values that cannot fit into the cache in total. However it should be noted that in well-tuned applications, the logical reads is a much dominant parameter in the database as compared to physical reads parameter, to the extent that it can be negligible as it does not post as the constraining factor. Nevertheless, it is also of great value to incorporate the option of physical reads parameter into this *affirmation* model, as explained in section 4.6.

5.6 Summary and discussion

In this chapter, the spectrums of the resource management proposals are analyzed in details. The type of data which is utilized for experiments is elaborated. In this case, the extraction of metadata from real workload, together with synthetic workload using TPC-H data are of interest. The choice of these data types is influenced by the security requirement where access to sensitive information in the database needs to be restricted as much as possible. The reasons for the unsuitability of scrambled data from real workload are also explained.

Subsequently, the experimental result to derive the proof-of-concept for the resource *monitoring* model is exhibited. The utilized parameters are clarified and scrutinized. In this case *Fuzzy Computing with Words* method is employed to represent these parameters for ease of interpretation. After that, the overhead and its effect on the model's accuracy are discussed. In order to increase the relevance and applicability of this resource *monitoring* model in real environment, SQL tuning is identified as the critical pre-requisite before the model can efficiently serve its purpose.

In the next section, the prototyping of the *optimization* model is detailed. The likelihood of deviations from the baselines is elaborated. The outcome of the analysis from these deviations will trigger various actions, for instance, a thorough hardware health check, revisiting of the application SQL, additional resource planning or scaling etc.

After that, the output from the experiments for the *affirmation* model is delineated. The extrapolation from outcome in the benchmarking stage is envisaged. In this section, there are some potential future works that could enrich the model, so that it can adequately applied to the real environments.

The analysis of these research works has also been published in the journals (C. H. Tan & Teh, 2013a, 2013b, 2013c).

The next chapter concludes the research works conducted in this thesis. Some future researches are also envisaged in the chapter, which will solidify these proposed resource management models.

6. CONCLUSIONS

6.1 Introduction

This study demonstrates the capability of utilizing statistical modeling to achieve resource management purposes in virtualized cloud environment. During the conceptualization phase of the prototypes, the proposals are aspired to take the data privacy and security concerns into consideration, as the suggested mechanisms are targeted for database operations that have stringent security requirements. In doing so, *linear regression analysis, machine learning, fuzzy computing, linear programming* and *simplex method* are the mathematical techniques employed in the construction of the models. These numerical approaches to build the models are taking the input from metadata in real data, as well as the TPC-H benchmark. At this point of this research, the proposed schemes can be deployed to complement current commercially available tools and utilities. Such refinement in the resource utilization takes the resource administration to a greater level of efficiency. The study is segregated into 3 themes, and each is related to each other sequentially. The *monitoring*, *optimization* and *affirmation* schemes, each of them targets different aspects in the resource utilization in cloud environment. In aggregation, they provide an almost holistic solution to the resource management domain.

6.2 Summary of solutions to the objectives' questions

At this point, the research questions in chapter 1 can be answered conclusively, as follows:

***Question 1*:** What are the appropriate methods to provide barometric indicators to determine the host performance?

Answer:

In the *monitoring* scheme, the collective response time of the SQL in a representative workload are gathered and analyzed. By employing the *linear regression* and *machine learning* methods, a graphical representation of the workload processing condition is presented for resource planning and scaling purposes. A hypothetical 5% zone is envisaged in the linear graphs. If 5% of total SQL transactions fall into this region, the resource constraint in the VM is deemed hitting the threshold level, which warrants subsequent resource provisioning activity.

Subsequently in the *optimization* scheme, the consistency of the hardware performance is measured by mean of comparing the gradient and y-intercept of the baselined data to the testing data. The testing data is collected periodically during the production phase of the application service offerings. Such recurring event can be scheduled daily or weekly, and the gradient and y-intercept must be similar to the baselines in order to ensure optimality and consistency in the computing performance. In addition, *correlation coefficient* parameter is employed to measure the relevance of the linear plots.

After that, the *affirmation* scheme envisages the creation of stress testing scenario, in order to provide a fitting environment for transactions' response time verification. In this case the host performance is measured based on the collection of response time from SLA-bound transactions. The parameter of SQL response time can be conveniently utilized to gauge the adequacy of computing resource in the VM, as the expected values are already stipulated in the established Quality of Service requirement in the Service Level Agreement.

From another perspective in this *affirmation* scheme, TPC-H queries are utilized to load the VM to the hypothetical resource threshold point. Consequently, the response time

from these queries are engaged to measure and verify the threshold point. Such constraining level in the resource may not be the same for the real workload as there could be relaxed or stringent SLA requirements; nevertheless this threshold obtained from the execution of TPC-H queries provides a ballpark indication on the capability of the VM.

Question 2: How can users' experience be matched to these indicators discovered in (1)?

Answer:

In the *monitoring* model, the workloads are to be obtained from the real environments. The effort to select the workload that is delineative for the general computing requirement in the VM must be careful so that it depicts the real transactions from the end users. In this case, undesired noises are filtered so that only real users' processes are channel into the *monitoring* mechanism for analysis purpose. Hence, the produced outcome from the analysis on the *monitoring* model will characterize the real users' computing requirement in the VM.

In the *optimization* scheme, the iterative runs of the TPC-H queries in the VM to diagnose the hardware performance consistency and optimality produce a relatively closer depiction of the VM performance, when it is compared to hardware health check activities that depend solely on OS parameters' values for interpretation. This is because the execution of TPC-H queries is running the database SQLs that mimics real database operations. Hence, resultants from SQL executions are nearer to the real users' experience.

The same condition is applied to the *affirmation* model. The determination of resource adequacy in the VM is characterized by verification of real SLA-bound transactions,

when the VM is stressed to its resource limit. The combined OS and database parameters, coupled with real execution of SQL to create the stress-testing scenario in the VM, are a more strategic approach to verify the critical transactions as well as to probe the VM capability.

***Question 3*:** What are the significant and appropriate parameters to be used to measure host performance?

Answer

Many of the current resource management utilities are practicing silo monitoring in the computing hosts. In other words, the parameters employed are either emphasized on operating system variables, or solely from the database end. The most appropriate method to exhibit the resource utilization state in the VM as portrayed in this thesis, is to combine the parameters from the operating system and database, where the condition in the host is matched to the status in the database. Such bilateral verification mechanism provides checks and balances, so that erroneous reporting from 1 end can be discovered by another.

***Question 4*:** How do these parameters interact with each other, in order to provide a more solidified output to measure the host and database performance?

Answer

The CPU run queue and SQL processing time are the 2 main parameters employed in the proposed resource management mechanisms. When the workload processing requirement increases, more CPU cycles are needed. In this case the process queue on the processors becomes longer. From the database perspective, this corresponds to the increment in the SQL processing time. The 2 parameters are directly proportional, before the resources utilization threshold in the VM. The interest in all the proposed

mechanisms is confined within this threshold point, where the linear correlation between these 2 parameters is manipulated to describe the VM and database performance condition.

***Question 5*:** How can the proposed mechanisms deliver the intended objectives, in term of accuracy and consistency?

Answer

Ideally, the models are invaluable in boosting the efficiency of currently deployed resource management utilities. Subsequent refinements in the area of automation and coding of the algorithms in GUI mode are essential to promote the adoption of the models in real environments. Nevertheless, the oscillation of the parameters' values in the data gathering phase for all the 3 models needs to be refined further. The instability in the parameter reporting for CPU run queue is the trickiest component in the proposed mechanisms. Once this fluctuation symptom is steadied, the consistency and accuracy of the models can be significantly improved.

***Question 6*:** How the hardware in the VM performs before and after resource constraining threshold?

Answer

Before the resource threshold, the computing resources are spent to complete the end users' requests. This is the most ideal condition from the perspective of providing application service offering to the consumers. In this mode, the transactions are processed without excessive waits either in the operating system or the database. The proposed mechanisms strive to provide clearer visibility to the resource state in the VM so that database operations can be performed within this limit as much as possible. Beyond the resource threshold, there is no guarantee that the database transactions can

be completed within the stipulated limit of the response time. In the extreme case, the transactions may be aborted due to excessive waiting time. Hence, the resource threshold limit is manipulated by the proposed algorithms to delineate the resource management proposals.

Question 7: Many of the RDBMS products in the industry have not developed the capability to be dynamically scaled, and the migration from 1 RDBMS platform to another is quite unlikely in commercial arena, what type of resource management mechanisms are appropriate for such semi-dynamic scalability requirement by these database systems?

Answer

The proposed mechanisms for resource management envisaged such condition for many parallel databases. In the resource utilization *monitoring* theme, the metadata is analyzed using *linear regression* method, and the outcome is obtained after a week of data collection and analysis. Such prolonged data aggregation is more accurate as compared to workload analysis obtained from short historical duration. In the resource utilization *optimization* arena, the algorithms computed by *machine learning* and *linear regression* are tested periodically, in recommended weekly schedule. The observatory result from the testing is evaluated weekly in this case for fault discovery. In this case, the hardware is not expected to change when comparing between the training and testing data in between the week. Subsequently, the proposed light-weighted stress testing mechanism in the resource utilization *affirmation* topic is carried out with the knowledge that the historical I/O condition can be referred to in order to mimic the real workload condition. Hence, the hardware configuration is expected to remain constant.

***Question 8*:** As fault analysis is a continuous effort, how capable the proposed mechanisms in accomplishing this goal?

Answer

The testing phases in the proposed mechanism in the resource utilization *optimization* theme which is computed using the *machine learning* technique can be executed daily or weekly. With such schedule, the fault in the hardware can be adequately detected, where the output is subsequently fed for failure prediction.

***Question 9*:** How to address the shortcoming of the current available benchmark, where one-size-fit-all scenario is almost nonexistent?

Answer

With the dynamicity nature of the various real workloads, it is almost impossible to produce benchmarks that can decently represent them. The proposed mechanisms in this thesis are utilizing TPC-H benchmark, with the view that these synthetic data and queries can be uniformly standardized across all platforms. Hence, the result of experiments carried out from 1 platform could potentially be applied for other platforms.

***Question 10*:** How security aspect is addressed in details, by the proposed mechanisms?

Answer

The idea of the proposed mechanisms is to prohibit IT administrators from accessing real users' data, but still preserving the capability to perform resource management tasks, with probably more superior methods. With these proposals, the real data can comfortably be masked without the concern that the data shielding activity from IT personnel may affect the normal administrative jobs.

6.3 Limitations of current study

The exhibited prototypes in this thesis can be considered as pioneer in the area of resource management. As the knowledge is new, there are a few short-comings which are to be further refined in subsequent research:

6.3.1 The *monitoring* scheme

1) The *monitoring* outcome is invalidated whenever the underlying hardware configuration is changed in the VM. Each time the resource state is changed, the resource threshold discovery mechanism is employed to identify the new CPU run queue threshold value. This value can only serve as the ballpark figure for the particular set of workload. This is due to the fact that each application has different SLA requirement, where some have more stringent response time specification, whereas some are more relaxed. Hence, a method to learn the threshold value which is tailored to particular workload will increase the accuracy of the model.

2) The discovery of the 5% zone is the core target in this scheme. In order to identify this segment, the metadata of the representative workload has to be sufficiently gathered. For some applications that become active only during particular timeframe of the year, the identification of this 5% zone location from real data will have to wait for such timeframe to arrive. Load testing to simulate the real transactions in the VM could be the viable solution. However it can only be done on the VM where the hosted databases allow such outages.

3) The metadata collector illustrated in this thesis assumes the service is adequately provided by the RDBMS vendors. Nevertheless, such facility is not guaranteed to be available in all the RDBMS. For those databases that do not have such option on data collection, the *monitoring* model cannot be deployed directly. A custom development is required to code the data collector's algorithms.

6.3.2 The *optimization* scheme

1) The fault analysis in this scheme is capable to predict relative long term hardware issue in the VM. It cannot be utilized to predict the potential failure which might happen in the short term, for instance within 1 hour. Hence conventional fault analysis and failure prediction to discover short term problem in the hardware cannot be complemented by this mechanism.

2) The collection of the testing data sets during the production phase is unwieldy due to the oscillative nature of the CPU run queue parameter. However this does not necessary signify faulty underlying hardware. Hence the tests may need to be carried out multiple times to arrive at the state where stabilized data is obtained.

3) Manual intervention is needed to run the test and harvest the required data. Further automation is required to hasten the tests and reduce inaccuracy due to human error.

6.3.3 The *affirmation* scheme

1) The benchmarking phase in producing the array of reference is taking much time. Furthermore, the whole cycle of acquiring the array needs to be repeated each time the hardware configuration changes. Such long outage requirement is not feasible in mission-critical applications that do not allow long downtime on the database VM. Hence a method needs to arrive to address the issue of long outage window, potentially using extrapolation technique.

2) In creating the stress testing scenario, only memory reads/s parameter is employed. It could be beneficial if combination of physical reads, memory and physical writes parameters are taken into account in constructing the stress-testing scenario.

3) Similar to the *optimization* model, the execution of the mechanism is dependent on IT administrators. The proposal can benefit from more automation steps built into the model.

6.4 Recommendations and future directions

In order for the proposed mechanisms to be deployed for real production use, they must be packaged into *graphical user interface* (GUI) product, in order to aid the deployment and usage. In today's IT industry, in order to promote wide adoption of certain products, GUI-based functionalities are the most important pre-requisite. For instance, all the recent development on cloud virtualization management software (SolarWinds, 2013; Splunk, 2013; VMTurbo, 2013) are porting all the available functionalities onto GUI-interfaces.

Apart from the above major improvement, following section discusses some important refinements which will take the proposals to greater acceptance level by industrial users:

1) The *monitoring* model

- A mechanism can be developed to detect and filter unnecessary noises in the VM, in order to preserve the purity of the metadata from the representative workload. With such facility, the effort to remove the outliers during the conversion of the collected data to the linear graphical representation can be alleviated.
- A method to learn the exact threshold value which is tailored to particular workload will increase the accuracy of the model.
- As the pre-requisite to the model is to ensure that the underlying SQL in the workload are performing to the most optimal condition, a mechanism can be developed to observe the run time of the repetitive transactions, so that overrun scenarios due to missing indexes, invalidated statistic or inefficiency in indexing operation can be detected and rectified.

2) The *optimization* model

- An automated mechanism to execute the tests during production phase, to obtain the stabilized outcome can greatly increase the ease of usage of the model. Furthermore, automation will also reduce administrative cost and boost the efficiency of the model.
- One of the challenges in this proposal is to determine the right duration in order for the CPU run queue parameter to achieve the stabilized condition. This is an area that worth further scrutiny.
- Future work is also worth carried out to analyze the CPU run queue parameter to discover the underlying factors that influence the oscillative reporting of the parameter value.

3) The *affirmation* model

- As the current proposal focuses on memory reads parameter, it can potentially be of great value to conduct more researches to include the physical reads, memory and disk writes parameters into the stress testing scenario creation. Such potential is envisaged in section 4.6. In that section, the methods to discover the optimize values for query executions are *linear programming* and *simplex method*. Potentially, the *duality* can be more advantageous as compared to *simplex method*; hence it will worth the research effort to scrutinize on the *duality* method in case the optimization algorithm is to be explored further.
- Throughout the thesis, the processor is assumed to be the component that is the dominant resource utilization factor in the VM. However, the underlying I/O subsystem could potentially become the constraining point, if the SAN storage is not rightly configured. Hence, a mechanism to detect the situation

where the I/O subsystem becomes the leading constraint in the VM is worth the research effort.

- The benchmarking stage in this model is consuming quite a substantial amount of time. Future work to include extrapolation on existing references to discover the new set of benchmark will be significant, as this will greatly reduce the benchmarking effort.

References

Ahuja, S., Mani, S., & Zambrano, J. (2012). A Survey of the State of Cloud Computing in Healthcare. *Network & Communication Technologies, 1*(2).

Alavala, C. R. (2008). *Fuzzy Logic and Neural Networks*. New Delhi, India: New Age International (P) Ltd.

Allen, A. O. (1990). *Probability, Statistics, and Queuing theory with Computer Science Applications – 2nd Edition*. San Diego, CA, USA: Academic Press, Inc.

Amazon. (2012). Amazon Web Services: Risk and Compliance. Amazon.com Inc.

Amos, C. K. (Producer). (2013, Feb 13). Soft sensors win hard jobs. *Chemical Processing*. Retrieved from http://www.chemicalprocessing.com/articles/2005/606/

An, B., Lesser, V., Irwin, D., & Zink, M. (2010). *Automated Negotiation with Decommitment for Dynamic Resource Allocation in Cloud Computing.* Paper presented at the Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems.

Anderson, D. (2004). *Boinc: A system for public-resource computing and storage.* Paper presented at the 5th IEEE/ACM International Workshop on Grid Computing.

Andrzejak, A., & Silva, L. (2007). *Deterministic models of software aging and optimal rejuvenation schedules.* Paper presented at the IEEE International Symposium on Integrated Network Management.

Apache. (August 2011). PigMix. Retrieved May 2013, 2013, from https://cwiki.apache.org/confluence/display/PIG/PigMix

Aria. (2013). An Enterprise-Grade Platform for Cloud Billing. Retrieved July, 2013, from http://www.ariasystems.com/our-platform/leverage-cloud-freedom

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., . . . Zaharia, M. (2009). Above the Clouds - A Berkeley View of Cloud Computing. UC Berkeley Reliable Adaptive Distributed Systems Laboratory.

Avril, P., & Hardie, W. (2013). Plug into the Cloud with Oracle Database 12c. Redwood Shores, CA: Oracle Corporation.

AWR. (2013). Oracle Automatic Workload Repository Survival Guide. Retrieved Aug, 2013, from http://www.akadia.com/services/ora_awr_survival_guide.html

Azure. (2010). Windows Azure: How Four Groups at Microsoft Developed Scalable, Efficient Applications that Unleashed Competitive Differentiators and Delivered High Value to Customers: Microsoft Corporation.

Babar, M. A., & Chauhan, M. A. (2011). *A tale of migration to cloud computing for sharing experiences and observations.* Paper presented at the Proceeding of the 2nd Internatonal Workshop on Software Engineering for Cloud Computing.

Bach, F. R., & Jordan, M., I. (2002). Kernel independent component analysis. *Journal of Machine Learning Research, 3*, 1-48.

Ban, L. B., Cocchiara, R., Lovejoy, K., Telford, R., & Ernest, M. (2010). The evolving role of IT managers and CIOs - Findings from the 2010 IBM Global IT Risk Study: IBM Corporation.

Banga, G., & Druschel, P. (1997). *Measuring the Capacity of a Web Server*. Paper presented at the Proceeding of the USENIX Symposium on Internet Technologies & Systems.

Barna, C., Litoiu, M., & Ghanbari, H. (2011). *Autonomic Load-Testing Framework.* Paper presented at the Proc. of the 8th ACM Intl. Conf. on Autonomic Comp.

Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems, 28*(5), 755–768.

Beloglazov, A., & Buyya, R. (2010). *Energy Efficient Resource Management in Virtualized Cloud Data Centers.* Paper presented at the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing.

Beloglazov, A., & Buyya, R. (2012). Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Transactions on Parallel and Distributed Systems*.

Bercovici, S. (2010). Enabling On-Demand Dynamic Resource Allocation and Elasticity in the Cloud with Radware's Solutions: Radware, Ltd.

Blagodurov, S., Gmach, D., Arlitt, M., Chen, Y., Hyser, C., & Fedorova, A. (2013). *Maximizing Server Utilization while Meeting Critical SLAs via Weight-Based Collocation Management.* Paper presented at the IFIP/IEEE International Symposium on Integrated Network Management.

Bolch, G., Greiner, S., Meer, H. D., & Trivedi, K. S. (2006). Markov Chain Model *in Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Hoboken, NJ: Wiley.com.

Botelho, B. (2007). Gartner predicts data center power and cooling crisis. Retrieved July, 2013, from http://searchdatacenter.techtarget.com/news/1260874/Gartner-predicts-data-center-power-and-cooling-crisis

Brett. (2008). Setting Up TPC-H. Retrieved Aug, 2013, from http://brettschroeder.blogspot.com/2008/09/setting-up-tpc-h-part-2.html

Buyya, R., Beloglazov, A., & Abawajy, J. (2010). *Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges.* Paper presented at the International Conference on Parallel and Distributed Processing Techniques and Applications.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Journal of Future Generation Computer Systems, 25*(6), 599-616.

C12G. (2010). OpenNebula extends support to Deltacloud APIs: virtualization.info.

Cardeñosa, G., Díez, I., Coronado, M., & Rodrigues, J. (2012). Analysis of Cloud-Based Solutions on EHRs Systems in Different Scenarios. *Journal of Medical Systems, 2012*(1).

Casale, G., Kalbasi, A., Krishnamurthy, D., & Rolia, J. (2009). *Automatic stress testing of multi-tier systems by dynamic bottleneck switch generation.* Paper presented at the Proc. of the 10th ACM/IFIP/USENIX Intl. Conf. on Middleware

Cengage. (2013). The Simplex method: minimization. Retrieved Aug, 2013, from http://college.cengage.com/mathematics/larson/elementary_linear/4e/shared/downloads/c09s4.pdf

Chen, T. S., Liu, C. H., Chen, T. L., Chen, C. S., Bau, J. G., & Lin, T. C. (2012). Secure Dynamic Access Control Scheme of PHR in Cloud Computing. *Journal of Medical Systems, 36*(6), 4005-4020.

Chen, Y. P. (May 2012). We don't know enough to make a Big Data benchmark suite an academia-industry view: UC Berkeley/Cloudera.

Chen, Y. P., Ganapathi, A., Griffith, R., & Katz, R. (March 2011). *The case for evaluating MapReduce performance using workload suites.* Paper presented at the Proc. of the 2011 IEEE 19th Annual Intl. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Syst.

Chen, Y. P., Ganapathi, A., & Katz, R. (2011). Challenges and opportunities for managing data systems using statistical models. *IEEE Data Eng. Bull., 34*, 53-60.

Chieu, T. C., Watson, T. J., Mohindra, A., Karve, A. A., & Segal, A. (2009). *Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment* Paper presented at the IEEE International Conference on e-Business Engineering.

Cisco. (2013). Intrusion Prevention System (IPS). Retrieved June, 2013, from http://www.cisco.com/en/US/products/ps5729/Products_Sub_Category_Home.html

Cloud-Security-Alliance. (2010). Top Threats to Cloud Computing V1.0: Cloud Security Alliance.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), 377-387.

Cole, R., Funke, F., Giakoumakis, L., Guy, W., Kemper, A., Krompass, S., . . . Waas, F. (2011). *The mixed workload CH-benCHmark.* Paper presented at the Proceedings of the Fourth International Workshop on Testing Database Systems.

Convirture. (2013). ConVirt Enterprise Cloud. Retrieved May 2013, 2013, from http://www.convirture.com/products_cloud.php

Cooney, M. (2011). Gartner: 10 key IT trends for 2012. Retrieved 19 May, 2013, from http://www.networkworld.com/community/blog/gartner-10-key-it-trends-2012

Costanzo, A. D., Assunção, M. D. D., & Buyya, R. (2009). Harnessing cloud technologies for a virtualized distributed computing infrastructure. *IEEE Internet Computing, 13*(5), 24-33.

Curino, C., Jones, E., Popa, R. A., Malviya, N., Wu, E., Madden, S., . . . Zeldovich, N. (2011). *Relational Cloud: A Database Service for the Cloud.* Paper presented at the 5th Biennial Conference on Innovative Data Systems Research.

Curino, C., Jones, E., Zhang, Y., & Madden, S. (2010). Schism: a Workload Driven Approach to Database Replication and Partitioning *Journal of VLDB Endowment, 3*(1-2), 48-57.

Das, S., Nishimura, S., Agrawal, D., & Abbadi, A. E. (2010). Live Database Migration for Elasticity in a Multitenant Database for Cloud Platforms: UCSB Computer Science Technical Report.

Dean, J., & Ghemawat, S. (January 2008). MapReduce: simplified data Processing on large clusters. *Communication of the ACM 50th Anniversary Issue, 51*(1), 107-113.

Devore, J. L. (2008). *Probability and Statistics for Engineering and the Sciences*. Belmont, CA, USA: Thomson Learning, Inc.

Donahue, S. (2010). Can Cloud Computing Help Fix Health Care. *Cloudbook Journal*, *1*(6).

Dongarra, J., & Sullivan, F. (2000). Guest Editors' Introduction: The Top 10 Algorithms. *Computing in Science and Engineering*, *2*(1).

Dutreilh, X., Rivierre, N., Moreau, A., & Malenfant, J. (2010). *From Data Center Resource Allocation to Control Theory and Back.* Paper presented at the IEEE 3rd International Conference on Cloud Computing (CLOUD).

Elastras. (2013). An Elastic Transactional Data Store for the Cloud. Retrieved June, 2013, from http://code.google.com/p/elastras/

Elkan, C. (2003). *Using the triangle inequality to accelerate k-means.* Paper presented at the Proceedings of the 20th International Conference on Machine Learning.

Eucalyptus. (2013a). The Eucalyptus Cloud. Retrieved June, 2013, from http://www.eucalyptus.com/eucalyptus-cloud/iaas

Eucalyptus. (2013b). The Eucalyptus Cloud. Retrieved July, 2013, from http://www.eucalyptus.com/eucalyptus-cloud/iaas

Eucalyptus. (2013c). Why Eucalyptus. Retrieved July, 2013, from http://www.eucalyptus.com/why-eucalyptus

Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., & Wong, P. (1997). *Theory and practice in parallel job scheduling.* Paper presented at the Proceedings of the Job Scheduling Strategies for Parallel Processing.

FGAC. (2003). Implementing Application Context and Fine-Grained Access Control. Retrieved Aug, 2013, from http://docs.oracle.com/cd/B13789_01/network.101/b10773/apdvcntx.htm

Fito, J. O., Goiri, I., & Guitart, J. (2010). *SLA-driven Elastic Cloud Hosting Provider.* Paper presented at the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP).

Fortinet. (2013). Intrusion Prevention System (IPS). Retrieved June, 2013, from http://www.fortinet.com/solutions/ips.html

Foster, I., Zhao, Y., Raicu, I., & Lu, S. Y. (2008). *Cloud Computing and Grid Computing 360-Degree Compared.* Paper presented at the Grid Computing Environments Workshop.

Fu, S., & Xu, C. Z. (2010). Quantifying event correlations for proactive failure management in networked computing systems. *Journal of Parallel and Distributed Computing*, *70*(11), 1100-1109.

Gainaru, A., Cappello, F., Snir, M., & Kramer, W. (2012). *Fault prediction under the microscope: A closer look into HPC systems.* Paper presented at the Proceedings of the Intl. Conf. on High Perf. Comp., Networking, Storage and Analysis

Gallet, M., Yigitbasi, N., Javadi, B., Kondo, D., Iosup, A., & Epema, D. (2010). *A model for space-correlated failures in large-scale distributed systems.* Paper presented at the In Proceedings of the 16th international Euro-Par conference on Parallel processing: Part I

Ganapathi, A., Chen, Y. P., Fox, A., Katz, R., & Patterson, D. (2010). *Statistics-driven workload modeling for the Cloud* Paper presented at the IEEE 26th International Conference on Data Engineering Workshops (ICDEW).

Ganapathi, A., Kuno, H., Dayal, U., Wiener, J. L., Fox, A., Jordan, M., & Patterson, D. (2009). *Predicting Multiple Performance Metrics for Queries: Better Decisions Enabled by Machine Learning.* Paper presented at the Proc. of the IEEE Intl. Conf. on Data Engineering.

Ganesh, M. (2008). *Introduction to Fuzzy Sets and Fuzzy logic*. Upper Saddle River, NJ: Prentice Hall Inc.

Garg, S. K., Gopalaiyengar, S. K., & Buyya, R. (2011). *SLA-Based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter.* Paper presented at the Proceedings of the 11th international conference on Algorithms and architectures for parallel processing.

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google File System. *ACM SIGOPS Operating Systems Review, 37*(5), 29-43.

Ghoshal, D., Canon, R., & Ramakrishnan, L. (2011). *I/O Performance of Virtualized Cloud Environments.* Paper presented at the In Proceeding of the 2nd International Workshop on Data Intensive Computing in the Clouds.

Glas, M., & Andres, P. (2011). Achieving the Cloud Computing Vision. Oracle Corporation.

Gmach, D., Rolia, J., Cherkasova, L., Belrose, G., Turicchi, T., & Kemper, A. (2008). *An integrated approach to resource pool management: policies, efficiency and quality metrics.* Paper presented at the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC.

Google. (2010). Security Whitepaper: Google Apps Messaging and Collaboration Products. Google Inc.

Gray, D., Los, R., Shackleford, D., & Sullivan, B. (2012). Top Threats to Cloud Computing - Survey Results Update 2012: Cloud Security Alliance.

Grimme, C., Lepping, J., & Papaspyrou, A. (2008). *Prospects of collaboration between compute providers by means of job interchange.* Paper presented at the Proceedings of the 13th international conference on Job scheduling strategies for parallel processing, Berlin, Heidelberg.

Gulati, A., Kumar, C., & Ahmad, I. (2009). Modeling Workloads and Devices for IO Load Balancing in Virtualized Environments. *ACM SIGMETRICS Performance Evaluation Review, 37*(3), 61-66.

Gulati, A., Shanmuganathan, G., Holler, A., & Ahmad, I. (2011). *Cloud-scale resource management: challenges and techniques.* Paper presented at the Proc. of the 3rd USENIX conf. on Hot topics in cloud computing

Hamlen, K., Kantarcioglu, K., Khan, L., & Thuraisingham, B. (2010). Security Issues for Cloud Computing. *International Journal of Information Security and Privacy, 4*(2), 39-51.

Hanmer, R. (2010). Software Rejuvenation. Alcatel-Lucent.

Harms, R., & Yamartino, M. (2010). The Economics of the Cloud. Microsoft Corporation.

Hashemi, S. M., & Bardsiri, A. K. (2012). Cloud Computing Vs. Grid Computing. *ARPN Journal of Systems and Software, 2*(5).

HP. (2007). Application performance testing in VMware environments - Identify and control performance and capacity risks. Hewlett-Packard Development Company: Hewlett-Packard Development Company.

HP. (2012). HP SiteScope software. Hewlett-Packard Development Company.

HP. (2013a). HP LoadRunner - Unmatched power and flexibility software testing tool for application performance. Retrieved July, 2013, from http://www8.hp.com/my/en/software-solutions/software.html?compURI=1175451

HP. (2013b). HP TippingPoint Next Generation Intrusion Prevention System (NGIPS). Retrieved June, 2013, from http://www8.hp.com/us/en/software-solutions/software.html?compURI=1343617

Hsu, C. H., Chen, T. L., & Park, J. H. (2008). On improving resource utilization and system throughput of master slave job scheduling in heterogeneous systems. *The Journal of Supercomputing, 45*(1), 129-150.

httperf. (2013). Welcome to the httperf homepage. Retrieved June, 2013, from http://www.hpl.hp.com/research/linux/httperf/

Huang, C. J., Guan, C. T., Chen, H. M., Wang, Y. W., Chang, S. C., Li, C. Y., & Weng, C. H. (2013). An adaptive resource management scheme in cloud computing. *Engineering Applications of Artificial Intelligence, 26*(1), 382–389.

Huang, S., Huang, J., Dai, J., Xie, T., & Huang, B. (March 2010). *The HiBench benchmark suite: Characterization of the MapReduce-based data analysis.* Paper presented at the IEEE 26th Intl. Conf. on Data Eng. Workshops.

Huber, A. (2013). Enterprise Manager 12c Cloud Control Application Performance Management: Oracle Corporation.

Hueeber, S. (2011). Examples for Simplex Algorithm. Retrieved July, 2013, from http://m2matlabdb.ma.tum.de/download.jsp?MC_ID=8&SC_ID=11&MP_ID=79

IBM. (2011). Platform-as-a-Service: What ISVs Need for World-Class Cloud Offerings. IBM.

IBM. (2013a). IBM PCIe Cryptographic Coprocessor. Retrieved June, 2013, from http://www-03.ibm.com/security/cryptocards/pciecc/overview.shtml

IBM. (2013b). Installing a Red Hat Enterprise Linux version 5 gold image. Retrieved June, 2013, from http://pic.dhe.ibm.com/infocenter/ibmfsb/v2r1/index.jsp?topic=%2Fcom.ibm.sbsolutions.doc%2FugRHELImageInstall.htm

IBM. (2013c). Rational Performance Tester Retrieved July, 2013, from http://www-03.ibm.com/software/products/us/en/performance/

Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., & Epema, D. H. J. (2008). The grid workloads archive. *Future Generation Computer Systems, 24*(7), 672–686.

Iqbal, W., Dailey, M. N., & Carrera, D. (2010). *SLA-Driven Dynamic Resource Management for Multi-tier Web Applications in a Cloud.* Paper presented at the

10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid).

Jangra, A., & Saini, T. (2013). Scheduling Optimization in Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(4).

Javadi, B., Abawajy, J., & Buyya, R. (2012). Failure-aware resource provisioning for hybrid cloud infrastructure. *Journal of Parallel and Distributed Computing, 72*(10), 1318-1331.

Javadi, B., Kondo, D., Vincent, J. M., & Anderson, D. P. (2011). Discovering statistical models of availability in large distributed systems: An empirical study of SETI@home. *IEEE Transactions on Parallel and Distributed Systems, 22*(11), 1896-1903.

Javadi, B., Thulasiraman, P., & Buyya, R. (2012). *Enhancing performance of failure-prone clusters by adaptive provisioning of cloud resources.* Paper presented at the The Journal of Supercomputing.

Jia, Y. T., & Shao, Z. (July 2009). A Benchmark for Hive, PIG and Hadoop: Apache.org.

Kaehler, S. D. (2005). Fuzzy logic - An Introduction (Part 3): The Seattle Robotics Society.

Karthikeyan, N., & Sukanesh, R. (2012). Cloud Based Emergency Health Care Information Service in India. *Journal of Medical Systems, 32*(6), 4031-4036.

Khatua, S., Ghosh, A., & Mukherjee, N. (2010). *Optimizing the utilization of virtual resources in Cloud environment.* Paper presented at the 2010 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems.

King, R. (2008). How Cloud Computing Is Changing the World. from http://www.businessweek.com/technology/content/aug2008/tc2008082_445669.htm

Kleinbaum, D., Kupper, L., Muller, K., & Nizam, A. (1998). *Applied Regression Analysis and Other Multivariable Methods.* Pacific Grove, CA, USA: Duxbury Press.

Kleinbaum, D. G., Kupper, L. L., Muller, K. E., & Nizam, A. (1998). *Applied Regression Analysis and Other Multivariable Methods*. Pacific Grove, CA, USA: Duxbury Press.

Kleinrock, L., & Korfhage, W. (1993). Collection unused processing capacity: An analysis of transient distributed systems. *IEEE transactions on Parallel and Distributed Systems, 4*(5).

Kocakahin. (2010). Create Your Own Oracle TPC-H Playground on Linux. Retrieved Aug, 2013, from http://husnusensoy.wordpress.com/2010/10/22/create-your-own-oracle-tpc-h-playground-on-linux/

Kondo, D., Javadi, B., Iosup, A., & Epema, D. (2010). *The failure trace archive: enabling comparative analysis of failures in diverse distributed systems.* Paper presented at the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing.

Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. Oakland, CA: Analytics Press.

Krishnamurthy, D., Rolia, J. A., & Majumdar, S. (2006). A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems. *IEEE Transactions on Software Engineering, 32*(11), 868-882.

Kumar, M., Roberts, S., & Kawalek, C. (2011). Oracle VM 3: Application-Driven Virtualization: Oracle Corp.

Kumar, V., & Garg, K. K. (2012). Migration of Services to the Cloud Environment: Challenges and Best Practices. *International Journal of Computer Applications, 1*(6), 1-6.

Kumar, V., Swetha, M., & Muneshwara, M. P., S. (2012). Cloud Computing: Towards case study of data security mechanism. . *International Journal of Advanced Technology and Engineering Research, 2*(4).

KVM. (2013). KVM Hypervisor. Retrieved July, 2013, from http://www.vservercenter.com/kvm-hypervisor

Laio, F. (2004). Cramer-von Mises and Anderson-Darling goodness of fit tests for extreme value distributions with unknown parameters. *Water Resources Research, 40*.

Li, M., Yu, S. C., Zheng, Y., Ren, K., & Lou, W. J. (2012). Scalable and Secure Sharing of Personal Health Records in Cloud Computing using Attribute-based Encryption. *IEEE Transactions on Parallel and Distributed Systems*.

Li, Q., & Guo, Y. (2010). *Optimization of Resource Scheduling in Cloud Computing.* Paper presented at the 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.

Lin, W. W., Liang, C., Wang, J. Z., & Buyya, R. (2012). *Bandwidth-aware divisible task scheduling for cloud computing.* Paper presented at the Software: Practice and Experience - Wiley Online Library.

Little, J. D. C. (1961). A Proof for the Queuing Formula: L=λW. *Operations Research, 9*(3), 383-387.

Lupse, O., Vida, M., & Stoicu-Tivadar, L. (2012). *Cloud Computing and Interoperability in Healthcare Information Systems.* Paper presented at the The First International Conference on Intelligent Systems and Applications.

Manning, C. D., Raghavan, P., & Schütze, H. (2009). Hierarchical clustering *An Introduction to Information Retrieval* (pp. 377-400). Cambridge, England: Cambridge University Press.

Mateen, A., Raza, B., Sher, M., Awais, M. M., & Mustapha, N. (2011). Workload management: A technology perspective with respect to self characteristics. *International Journal of Physical Sciences, 7*(9), 1482-1492.

MATLAB. (2013). Correlation coefficients. Retrieved Aug, 2013, from http://www.mathworks.com/help/matlab/ref/corrcoef.html

Microsoft. (2007). Windows Server 2008 Hyper-V Product Overview – An Early look: Microsoft Corp.

Microsoft. (2011). SQL Azure - Moving Business Intelligence To The Cloud. Wipro Technologies.

Microsoft. (2013a). How to Clone a Virtual Machine. Retrieved June, 2013, from http://technet.microsoft.com/en-us/library/bb740905.aspx

Microsoft. (2013b). Microsoft health Vault. Retrieved June, 2013, from https://www.healthvault.com/my/en

Microsoft. (2013c). Testing Performance and Stress Using Visual Studio Web Performance and Load Tests. Retrieved July, 2013, from http://msdn.microsoft.com/en-us/library/vstudio/dd293540.aspx

Microsoft. (2013d). Virtual Machine Manager. Retrieved May 2013, 2013, from http://technet.microsoft.com/en-us/library/gg610610.aspx

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). Foundations of Machine Learning. MIT: MIT Press.

Mosberger, D., & Jin, T. (1998). httperfmA Tool for Measuring Web Server Performance: HP Research Labs.

Mozafari, B., Curino, C., & Madden, S. (2013). *DBSeer: Resource and Performance Prediction for Building a Next Generation Database Cloud.* Paper presented at the 6th Biennial Conference on Innovative Data Systems Research.

Myers, J. E. (2012). Data Modeling for Healthcare Systems Integration: Use of the MetaModel: The Metadata Company.

Myung, J. (2003). Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology, 47*, 90-100.

Narsky, I. (2003). *Goodness of Fit: What Do We Really Want to Know?* Paper presented at the California Institute of Technology.

NERSC. (2013). National Energy Research Scientific Computing Center. Retrieved July, 2013, from http://www.nersc.gov/

Neter, J., Kutner, M., Nachtsheim, C., & Wasserman, W. (1996). *Applied Linear Regression Models.* USA: The McGraw-Hill Companies, Inc.

Nimbus. (2013). Nimbus Monitoring System. Retrieved 4 May 2013, 2013, from http://www.nimbusproject.org/

OEM. (2013). Oracle Enterprise Manager 12c. Retrieved Aug, 2013, from http://www.oracle.com/technetwork/oem/enterprise-manager/overview/index.html

OpenNebula. (2013). About the OpenNebula.org Project. Retrieved July, 2013, from http://opennebula.org/about:about

Oracle. (2009). Introduction to Automatic Workload Management. Retrieved Aug, 2013, from http://docs.oracle.com/cd/B28359_01/rac.111/b28254/hafeats.htm#BABBBGJB

Oracle. (2011). Oracle Exalogic Elastic Cloud: A Brief Introduction. Oracle Corporation.

Oracle. (2012). Oracle Engineered Systems Price List. Oracle Corporation.

Oracle. (2013). Oracle Cloud Management Pack for Oracle Database: Oracle Corp.

Oracle_RAC. (2013). Introduction to Oracle Real Application Clusters. Retrieved Julu, 2013, from http://docs.oracle.com/cd/B28359_01/rac.111/b28254/admcon.htm

Pavlo, A., Curino, C., & Zdonik, S. (2012). *Skew-Aware Automatic Database Partitioning in Shared-Nothing, Parallel OLTP Systems.* Paper presented at the Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data.

Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., & Stonebraker, M. (2009). *A Comparison of Approaches to Large-Scale Data Analysis.* Paper presented at the Proceedings of the 2009 ACM SIGMOD International Conference on Management of data.

Pettey, C. (2007). Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emissions. Retrieved 19 May, 2013, from http://www.gartner.com/newsroom/id/503867

Principe, J. C., Euliano, N. R., & Lefebvre, W. C. (2000). *Neural and Adaptive Systems*. New York, NY: John Wiley & Sons, Inc.

Raicu, I., Zhao, Y., Foster, I. T., & Szalay, A. (2008). *Accelerating Large-Scale Data Exploration through Data Diffusion.* Paper presented at the Proceedings of the 2008 international workshop on Data-aware distributed computing.

Raluca Ada Popa, R. A., Redfield, C. M. S., Zeldovich, N., & Balakrishnan, H. (2011). *CryptDB: Protecting Confidentiality with Encrypted Query Processing.* Paper presented at the Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles

Roy, S., & Mukherjee, N. (2011). Efficient resource management for running multiple concurrent jobs in a computational grid environment. *Future Generation Computer Systems, 27*(8), 1070–1082.

Sakr, S., & Liu, A. (2012). *SLA-Based and Consumer-Centric Dynamic Provisioning for Cloud Databases.* Paper presented at the IEEE 5th International Conference on Cloud Computing.

Salfner, F., Lenk, M., & Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys, 42*(3).

Samplify. (2013). Big Science (HPC). Retrieved June, 2013, from http://www.samplify.com/applications/hpc/

Saugatuck. (2011). Platform-as-a-Service: What ISVs Need for World-Class Cloud Offerings: Saugatuck Technology Inc.

Sinha, S. M. (2006). *Mathematical Programming - Theory and Methods*. New Delhi, India: Elsevier Science.

Sitescope. (2013). HP SiteScope software. Retrieved Aug, 2013, from http://www8.hp.com/us/en/software-solutions/software.html?compURI=1174244

SolarWinds. (2013). Virtualization Manager. Retrieved Aug, 2013, from http://www.solarwinds.com/virtualization-manager.aspx

Splunk. (2013). Splunk App for VMware. Retrieved Aug, 2013, from http://www.splunk.com/view/splunk-app-for-vmware/SP-CAAAHKV

Tan, C. H., & Teh, Y. W. (2013a). Harnessing Cloud Computing for Dynamic Resource Requirement by Database Workloads. *Journal of Information Science and Engineering, 29*(5).

Tan, C. H., & Teh, Y. W. (2013b). Secure Hardware Performance Analysis in Virtualized Cloud Environment. *Mathematical Problems in Engineering, 2003*.

Tan, C. H., & Teh, Y. W. (2013c). Synthetic Hardware Performance Analysis in Virtualized Cloud Environment for Healthcare Organization. *Journal of Medical Systems, 37*(4).

Tan, P. N., Steinbach, M., & Kumar, V. (2006). Introduction to Data Mining. Boston, US: Pearson Addison Wesley.

Tbeileh, K. (2009). Oracle Database Vault with Oracle Database 11g Release 2: Oracle Corporation.

Teh, Y. W. (2006). *Introduction to data mining*. Kuala Lumpur: University Malaya Press.

Thakkar, D., Hassan, A. E., Hamann, G., & Flora, P. (2008). *A Framework for Measurement Based Performance Modeling.* Paper presented at the Proc. of the 7th Intl. workshop on Software and performance.

TPC-H. (2013). TPC-H. Retrieved Aug, 2013, from http://www.tpc.org/tpch/

TPC. (2012). TPC Benchmark, revision 2.14.4: TPC.

TPC. (2013a). Overview of the TPC Benchmark C: The Order-Entry Benchmark: TPC.

TPC. (2013b). TPC-DS. TPC: TPC.

Upson, S. (2011). Cloud Computing - It's always Sunny in the Cloud. Retrieved May 2011, from http://spectrum.ieee.org/static/special-report-top-11-technologies-of-the-decade

Vaidyanathan, K., & Trivedi, K. S. (2005). A Comprehensive Model for Software Rejuvenation. *IEEE Transactions on Dependable and Secure Computing 2*(2).

VAM. (2013). Vogel's Approximation Method (VAM). Retrieved July, 2013, from http://www.springerreference.com/docs/html/chapterdbid/6314.html

Vandenberghe, L., Boyd, S., & Nouralishahi, M. (2002). *Robust Linear Programming and Optimal Control.* Paper presented at the In Proceedings of the 15th Ifac World Congress on the International Federation of Automatic Control.

Vishwanath, K., & Nagappan, N. (2010). *Characterizing Cloud Computing Hardware Reliability.* . Paper presented at the Proceedings of the 1st ACM symposium on Cloud computing.

VMTurbo. (2013). VMTurbo. Retrieved Aug, 2013, from http://www.vmturbo.com/

VMWare. (2006). VMware Infrastructure Architecture Overview. VMWare Inc.

VMWare. (2013a). VMware vCenter Server. Retrieved July, 2013, from http://www.vmware.com/products/vcenter-server/overview.html

VMWare. (2013b). VMware vSphere Hypervisor. Retrieved July, 2013, from http://www.vmware.com/products/vsphere-hypervisor/overview.html

Wan, D. D., Greenway, A., Harris, J. G., & Alter, A. E. (2010). Six questions every health industry executive should ask about cloud computing: Accenture Inc.

Wang, X. W., Sun, J. J., Li, H. X., Wu, C., & Huang, M. (2013). A Reverse Auction Based Allocation Mechanism in the Cloud Computing Environment. *Applied Mathematics & Information Sciences, 7*(1), 75-84.

Ward, M. (2011). Relational Database Management Systems in the Cloud: Microsoft SQL Server 2008 R2: Amazon Web Services.

Weisstein, E. W. (2013). Distance. Retrieved July, 2013, from http://mathworld.wolfram.com/Distance.html

Wiesel, A., Eldar, Y. C., & Yeredor, A. (2008). Linear Regression With Gaussian Model Uncertainty: Algorithms and Bounds. *IEEE Transactions on Signal Processing, 56*(6), 2194 - 2205.

Williams, J. (2013). A technical Overview of New Features for Automatic Storage Management in Oracle Database 12c: Oracle Corporation.

Williams, S. (2011). IBM Cloud Services - Balancing compute options: How IBM SmartCloud can be a catalyst for IT transformation: Technology Business Research Inc.

Wolloch, U. (2013). Introduction to EBS volumes and snapshots. Retrieved July, 2013, from http://www.n2ws.com/blog/introduction-to-ebs-volumes-and-snapshots.html

Xen. (2013). The Hypervisor. Retrieved July, 2013, from http://www.xenproject.org/developers/teams/hypervisor.html

Xhafa, F., & Abraham, A. (2010). Computational models and heuristic methods for Grid scheduling problems. *Future Generation Computer Systems, 26*(4), 608-621.

Yagoub, K., & Gongloor, P. (2007). SQL Performance Analyzer: Oracle Corporation.

Yigitbasi, N., Gallet, M., Kondo, D., Iosup, A., & Epema, D. (2010). *Analysis and modeling of time-correlated failures in large-scale distributed systems.* Paper presented at the 11th IEEE/ACM International Conference on Grid Computing (GRID).

Younge, A. J., Laszewski, G. V., Wang, L. Z., Lopez, A. S., & Carithers, W. (2010). *Efficient Resource Management for Cloud Computing Environments.* Paper presented at the International Green Computing Conference.

Yuan, Y., & Liu, W. C. (2011). *Efficient resource management for cloud computing.* Paper presented at the International Conference on System Science, Engineering Design and Manufacturing Informatization (ICSEM).

Zadeh, L. (1996). Fuzzy Logic = Computing with Words. . *IEEE Transactions on Fuzzy Systems, 4*(2).

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications, 1*, 7-18.

Zhang, R., & Liu, L. (2010). *Security Models and Requirements for Healthcare Application Clouds.* Paper presented at the IEEE 3rd International Conference on Cloud Computing.

Zhang, S., Zhang, S. F., Chen, X. B., & Huo, X. Z. (2010). *The Comparison Between Cloud Computing and Grid Computing.* Paper presented at the International Conference on Computer Application and System Modeling.

Zhu, X. J. (2007). *Semi-Supervised Learning Tutorial.* Paper presented at the International Conference on Machine Learning.

Zhu, X. J. (2008). Semi-Supervised Learning Literature Survey. USA: University of Wisconsin, Madison.

Appendix A

Optimization by Vogel's Approximation Method.

	0	0	5	5		
	Consumer 1	**Consumer 2**	**Consumer 3**	**Consumer 4**	**supply**	
Provider 1	10	30	25	15	14	5
Provider 2	x	10	x	x	0	
Provider 3	10	30	20	20	15	10
Provider 4	30	40	35	45	12	5
dummy	0	0	0	0	1	0
requirement	10	5	12	15		

	Consumer 1	**Consumer 2**	**Consumer 3**	**Consumer 4**	**supply**
Provider 1	x	30	25	15	14
Provider 2	x	10	x	x	0
Provider 3	10	30	20	20	15-10=5
Provider 4	x	40	35	45	12
dummy	x	0	0	0	1
requirement	10=10=0	5	12	15	

		0	5	5		
	Consumer 1	**Consumer 2**	**Consumer 3**	**Consumer 4**	**supply**	
Provider 1	x	30	25	15	14	10
Provider 2	x	10	x	x	0	
Provider 3	10	30	20	20	5	0
Provider 4	x	40	35	45	12	10
dummy	x	0	0	0	1	0
requirement	0	5	12	15		

	Consumer 1	**Consumer 2**	**Consumer 3**	**Consumer 4**	**supply**
Provider 1	x	x	x	14	14-14=0
Provider 2	x	10	x	x	0
Provider 3	10	30	20	20	5
Provider 4	x	40	35	45	12
dummy	x	0	0	0	1
requirement	0	5	12	15-14=1	

		10	15	25		
	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply	
Provider 1	x	x	x	14	0	
Provider 2	x	10	x	x	0	
Provider 3	10	30	20	20	5	0
Provider 4	x	40	35	45	12	5
dummy	x	0	0	0	1	0
requirement	0	5	12	1		

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	30	20	1	5-1=4
Provider 4	x	40	35	x	12
dummy	x	0	0	x	1
requirement	0	5	12	1-1=0	

		10	15			
	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply	
Provider 1	x	x	x	14	0	
Provider 2	x	10	x	x	0	
Provider 3	10	30	20	1	4	10
Provider 4	x	40	35	x	12	5
dummy	x	0	0	x	1	0
requirement	0	5	12	0		

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	x	4	1	4-4=0
Provider 4	x	40	35	x	12
dummy	x	0	0	x	1
requirement	0	5	12-4=8	0	

		40	35			
	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply	
Provider 1	x	x	x	14	0	
Provider 2	x	10	x	x	0	
Provider 3	10	x	4	1	0	
Provider 4	x	40	35	x	12	5
dummy	x	0	0	x	1	0
requirement	0	5	8	0		

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	x	4	1	0
Provider 4	x	5	35	x	12-5=7
dummy	x	x	0	x	1
requirement	0	5-5=0	8	0	

			35			
	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply	
Provider 1	x	x	x	14	0	
Provider 2	x	10	x	x	0	
Provider 3	10	x	4	1	0	
Provider 4	x	5	35	x	7	35
dummy	x	x	0	x	1	0
requirement	0	0	8	0		

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	x	4	1	0
Provider 4	x	5	7	x	7-7=0
dummy	x	x	0	x	1
requirement	0	0	8-7=1	0	

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	x	4	1	0
Provider 4	x	5	7	x	0
dummy	x	x	0	x	1
requirement	0	0	1	0	

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	x	4	1	0
Provider 4	x	5	7	x	0
dummy	x	x	1	x	1-1=0
requirement	0	0	1-1=0	0	

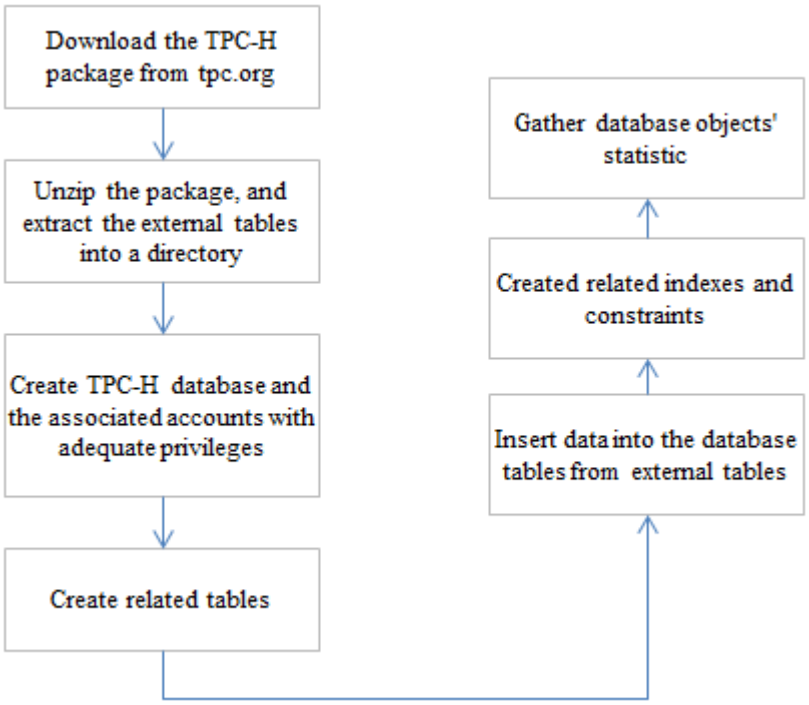
	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1	x	x	x	14	0
Provider 2	x	10	x	x	0
Provider 3	10	x	4	1	0
Provider 4	x	5	7	x	0
dummy	x	x	1	x	0
requirement	0	0	0	0	

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	supply
Provider 1				14(15)	0
Provider 2		10(15)			0
Provider 3	10(10)		4(20)	1(20)	0
Provider 4		5(40)	7(35)		0
dummy			1(0)		0
requirement	0	0	0	0	

Appendix B

The following TPC-H database setup steps are modified from (Kocakahin, 2010) and (Brett, 2008).

1. The high level pseudo-code to setup the TPC-H database is illustrated as follows:



2. The actual setup steps in Unix environment are as follows:

** *"$>" denotes commands executed in the command prompt*

```
$> mkdir tpch
$> mv tpch_2_12_0_b5.zip ./tpch
$> cd tpch/
$> unzip tpch_2_12_0_b5.zip

$> cp makefile.suite makefile
$> vi makefile (modify following parameters)

CC      = gcc
DATABASE= ORACLE
MACHINE = LINUX
WORKLOAD = TPCH

$> make

$> ./dbgen -s 4 -S 1 -C 8 -v
$> ./dbgen -s 4 -S 2 -C 8 -v
$> ./dbgen -s 4 -S 3 -C 8 -v
$> ./dbgen -s 4 -S 4 -C 8 -v
$> ./dbgen -s 4 -S 5 -C 8 -v
$> ./dbgen -s 4 -S 6 -C 8 -v
$> ./dbgen -s 4 -S 7 -C 8 -v
$> ./dbgen -s 4 -S 8 -C 8 -v
```

```
$> du -ch *.tbl* | tail -1

$> gzip -4 -v *.tbl*
```

Setup steps in the blank Oracle database

***** "SQL>" denotes commands executed in the sqlplus prompt***

```
SQL> create tablespace TPCH
        datafile '+DATA' size 100M autoextend on next 10M
        maxsize 10G segment space management auto
        encryption using 'AES256' default storage (encrypt);

SQL> create user tpch
        identified by tpch123
        default tablespace TPCH;

SQL> grant dba to tpch;

SQL> create directory tpch_dir as '/oracle/backups/TEST/tpch';
SQL> create directory zcat_dir as '/bin';

SQL> drop table region_ext;
SQL> drop table nation_ext;
SQL> drop table supplier_ext;
SQL> drop table customer_ext;
SQL> drop table order_ext;
SQL> drop table part_ext;
SQL> drop table partsupp_ext;
SQL> drop table lineitem_ext;

SQL> CREATE TABLE region_ext (r_regionkey NUMBER(10),
        r_name varchar2(25),
        r_comment varchar(152))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('region.tbl.gz'))
        NOPARALLEL
        REJECT LIMIT 0
        NOMONITORING;

SQL> CREATE TABLE nation_ext (n_nationkey NUMBER(10),
        n_name varchar2(25),
        n_regionkey number(10),
        n_comment varchar(152))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('nation.tbl.gz'))
```

```
        NOPARALLEL
        REJECT LIMIT 0
        NOMONITORING;

SQL> CREATE TABLE supplier_ext (S_SUPPKEY NUMBER(10),
        S_NAME VARCHAR2(25),
        S_ADDRESS VARCHAR2(40),
        S_NATIONKEY NUMBER(10),
        S_PHONE VARCHAR2(15),
        S_ACCTBAL NUMBER,
        S_COMMENT VARCHAR2(101))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('supplier.tbl.1.gz','supplier.tbl.2.gz','supplier.tbl.3.gz','supplier.tbl.4.gz',
        'supplier.tbl.5.gz','supplier.tbl.6.gz','supplier.tbl.7.gz','supplier.tbl.8.gz') )
        PARALLEL 2
        REJECT LIMIT 0
        NOMONITORING;

SQL> CREATE TABLE customer_ext (C_CUSTKEY NUMBER(10),
        C_NAME VARCHAR2(25),
        C_ADDRESS VARCHAR2(40),
        C_NATIONKEY NUMBER(10),
        C_PHONE VARCHAR2(15),
        C_ACCTBAL NUMBER,
        C_MKTSEGMENT VARCHAR2(10),
        C_COMMENT VARCHAR2(117))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('customer.tbl.1.gz','customer.tbl.2.gz','customer.tbl.3.gz','customer.tbl.4.gz',
        'customer.tbl.5.gz','customer.tbl.6.gz','customer.tbl.7.gz','customer.tbl.8.gz') )
        PARALLEL 2
        REJECT LIMIT 0
        NOMONITORING;

SQL> CREATE TABLE order_ext (O_ORDERKEY NUMBER(10),
        O_CUSTKEY NUMBER(10),
        O_ORDERSTATUS CHAR(1),
        O_TOTALPRICE NUMBER,
        O_ORDERDATE VARCHAR2(10),
        O_ORDERPRIORITY VARCHAR2(15),
        O_CLERK VARCHAR2(15),
        O_SHIPPRIORITY NUMBER(38),
        O_COMMENT VARCHAR2(79))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
```

```
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('orders.tbl.1.gz','orders.tbl.2.gz','orders.tbl.3.gz','orders.tbl.4.gz',
        'orders.tbl.5.gz','orders.tbl.6.gz','orders.tbl.7.gz','orders.tbl.8.gz'))
        PARALLEL 2
        REJECT LIMIT 0
        NOMONITORING;

SQL> CREATE TABLE part_ext (P_PARTKEY NUMBER(10),
        P_NAME VARCHAR2(55),
        P_MFGR VARCHAR2(25),
        P_BRAND VARCHAR2(10),
        P_TYPE VARCHAR2(25),
        P_SIZE NUMBER(38),
        P_CONTAINER VARCHAR2(10),
        P_RETAILPRICE NUMBER,
        P_COMMENT VARCHAR2(23))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('part.tbl.1.gz','part.tbl.2.gz','part.tbl.3.gz','part.tbl.4.gz',
        'part.tbl.5.gz','part.tbl.6.gz','part.tbl.7.gz','part.tbl.8.gz') )
        PARALLEL 2
        REJECT LIMIT 0
        NOMONITORING;

SQL> CREATE TABLE partsupp_ext (PS_PARTKEY NUMBER(10),
        PS_SUPPKEY NUMBER(10),
        PS_AVAILQTY NUMBER(38),
        PS_SUPPLYCOST NUMBER,
        PS_COMMENT VARCHAR2(199))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('partsupp.tbl.1.gz','partsupp.tbl.2.gz','partsupp.tbl.3.gz','partsupp.tbl.4.gz',
        'partsupp.tbl.5.gz','partsupp.tbl.6.gz','partsupp.tbl.7.gz','partsupp.tbl.8.gz'))
        PARALLEL 2
        REJECT LIMIT 0
        NOMONITORING;

SQL> CREATE TABLE lineitem_ext (L_ORDERKEY NUMBER(10),
        L_PARTKEY NUMBER(10),
        L_SUPPKEY NUMBER(10),
        L_LINENUMBER NUMBER(38),
```

```
        L_QUANTITY NUMBER,
        L_EXTENDEDPRICE NUMBER,
        L_DISCOUNT NUMBER,
        L_TAX NUMBER,
        L_RETURNFLAG CHAR(1),
        L_LINESTATUS CHAR(1),
        L_SHIPDATE VARCHAR2(10),
        L_COMMITDATE VARCHAR2(10),
        L_RECEIPTDATE VARCHAR2(10),
        L_SHIPINSTRUCT VARCHAR2(25),
        L_SHIPMODE VARCHAR2(10),
        L_COMMENT VARCHAR2(44))
        ORGANIZATION EXTERNAL (
        TYPE oracle_loader
        DEFAULT DIRECTORY tpch_dir
        ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        PREPROCESSOR zcat_dir:'zcat'
        BADFILE 'bad_%a_%p.bad'
        LOGFILE 'log_%a_%p.log'
        FIELDS TERMINATED BY '|'
        MISSING FIELD VALUES ARE NULL)
        LOCATION ('lineitem.tbl.1.gz','lineitem.tbl.2.gz','lineitem.tbl.3.gz','lineitem.tbl.4.gz',
        'lineitem.tbl.5.gz','lineitem.tbl.6.gz','lineitem.tbl.7.gz','lineitem.tbl.8.gz'))
        PARALLEL 2
        REJECT LIMIT 0
        NOMONITORING;

SQL> DROP TABLE H_CUSTOMER CASCADE CONSTRAINTS ;
SQL> DROP TABLE H_LINEITEM CASCADE CONSTRAINTS ;
SQL> DROP TABLE H_NATION CASCADE CONSTRAINTS ;
SQL> DROP TABLE H_ORDER CASCADE CONSTRAINTS ;
SQL> DROP TABLE H_PART CASCADE CONSTRAINTS ;
SQL> DROP TABLE H_PARTSUPP CASCADE CONSTRAINTS ;
SQL> DROP TABLE H_REGION CASCADE CONSTRAINTS ;
SQL> DROP TABLE H_SUPPLIER CASCADE CONSTRAINTS ;

SQL> CREATE TABLE H_CUSTOMER (c_custkey NUMBER(10) NOT NULL,
        c_name VARCHAR2(25) NOT NULL,
        c_address VARCHAR2(40) NOT NULL,
        c_nationkey NUMBER(10) NOT NULL ,
        c_phone VARCHAR2(15) NOT NULL,
        c_acctbal NUMBER NOT NULL,
        c_mktsegment VARCHAR2(10) NOT NULL,
        c_comment VARCHAR2(117) NOT NULL)
        PARALLEL 2;

SQL> CREATE TABLE H_LINEITEM (l_orderkey NUMBER(10) NOT NULL,
        l_partkey NUMBER(10) NOT NULL,
        l_suppkey NUMBER(10) NOT NULL ,
        l_linenumber INTEGER  NOT NULL ,
        l_quantity NUMBER NOT NULL,
        l_extendedprice NUMBER NOT NULL,
        l_discount NUMBER NOT NULL,
        l_tax NUMBER NOT NULL,
        l_returnflag CHAR(1) NOT NULL ,
        l_linestatus CHAR(1) NOT NULL,
        l_shipdate DATE NOT NULL,
        l_commitdate DATE NOT NULL,
        l_receiptdate DATE NOT NULL,
        l_shipinstruct VARCHAR2(25) NOT NULL,
        l_shipmode VARCHAR2(10) NOT NULL,
        l_comment VARCHAR2(44) NOT NULL)
```

```
        PARALLEL 2;

SQL> CREATE TABLE H_NATION (n_nationkey NUMBER(10) NOT NULL,
        n_name VARCHAR2(25) NOT NULL,
        n_regionkey NUMBER (10) NOT NULL,
        n_comment VARCHAR2 (152) NOT NULL)
        NOPARALLEL;

SQL> CREATE TABLE H_ORDER (o_orderkey NUMBER (10)  NOT NULL,
        o_custkey NUMBER(10)  NOT NULL,
        o_orderstatus CHAR(1) NOT NULL,
        o_totalprice NUMBER NOT NULL,
        o_orderdate DATE NOT NULL,
        o_orderpriority VARCHAR2(15) NOT NULL,
        o_clerk VARCHAR2(15) NOT NULL,
        o_shippriority INTEGER NOT NULL,
        o_comment VARCHAR2(79) NOT NULL)
        PARALLEL 2;

SQL> CREATE TABLE H_PART (p_partkey NUMBER(10)  NOT NULL,
        p_name VARCHAR2(55) NOT NULL,
        p_mfgr VARCHAR2(25) NOT NULL,
        p_brand VARCHAR2(10) NOT NULL,
        p_type VARCHAR2(25) NOT NULL,
        p_size INTEGER NOT NULL,
        p_container VARCHAR2(10) NOT NULL,
        p_retailprice NUMBER NOT NULL,
        p_comment VARCHAR2(23) NOT NULL)
        PARALLEL 2;

SQL> CREATE TABLE H_PARTSUPP (ps_partkey NUMBER (10)  NOT NULL ,
        ps_suppkey NUMBER (10)  NOT NULL ,
        ps_availqty INTEGER NOT NULL,
        ps_supplycost NUMBER NOT NULL,
        ps_comment VARCHAR2 (199) NOT NULL)
        PARALLEL 2;

SQL> CREATE TABLE H_REGION (r_regionkey NUMBER (10)  NOT NULL ,
        r_name VARCHAR2 (25) NOT NULL,
        r_comment VARCHAR2 (152) NOT NULL)
        NOPARALLEL;

SQL> CREATE TABLE H_SUPPLIER (s_suppkey NUMBER (10)  NOT NULL ,
        s_name VARCHAR2 (25) NOT NULL,
        s_address VARCHAR2 (40) NOT NULL,
        s_nationkey NUMBER (10)  NOT NULL ,
        s_phone VARCHAR2 (15) NOT NULL,
        s_acctbal NUMBER NOT NULL,
        s_comment VARCHAR2 (101) NOT NULL)
        PARALLEL 2;

SQL> truncate table h_lineitem;
SQL> truncate table h_order;
SQL> truncate table h_part;
SQL> truncate table h_customer;
SQL> truncate table h_nation;
SQL> truncate table h_region;
SQL> truncate table h_partsupp;
SQL> truncate table h_supplier;
SQL> alter session enable parallel dml;

SQL> insert /*+append*/into h_lineitem
        select L_ORDERKEY,
```

```
            L_PARTKEY,
            L_SUPPKEY,
            L_LINENUMBER,
            L_QUANTITY,
            L_EXTENDEDPRICE,
            L_DISCOUNT,
            L_TAX,
            L_RETURNFLAG,
            L_LINESTATUS,
            to_date(L_SHIPDATE, 'YYYY-MM-DD'),
            to_date(L_COMMITDATE, 'YYYY-MM-DD'),
            to_date(L_RECEIPTDATE, 'YYYY-MM-DD'),
            L_SHIPINSTRUCT,
            L_SHIPMODE,
            L_COMMENT
        from lineitem_ext;

SQL> insert /*+append*/ into h_partsupp  select * from partsupp_ext;
SQL> insert /*+append*/ into h_part  select * from part_ext;
SQL> insert /*+append*/ into h_order
        select o_orderkey,
            o_custkey,
            o_orderstatus,
            o_totalprice,
            to_date(o_orderdate, 'YYYY-MM-DD'),
            O_ORDERPRIORITY,
            o_clerk,
            O_SHIPPRIORITY,
            o_comment
        from order_ext;

SQL> insert /*+append*/ into h_customer  select * from customer_ext;
SQL> insert /*+append*/ into h_supplier  select * from supplier_ext;
SQL> insert  /*+append*/ into h_nation  select * from nation_ext;
SQL> insert /*+append*/ into h_region  select * from region_ext;
SQL> commit;

SQL> ALTER TABLE H_REGION ADD CONSTRAINT REGION_PK PRIMARY KEY
(r_regionkey);
SQL> ALTER TABLE H_NATION ADD CONSTRAINT NATION_PK PRIMARY KEY
(n_nationkey);
SQL> ALTER TABLE H_SUPPLIER ADD CONSTRAINT SUPPLIER_PK PRIMARY KEY
(s_suppkey);

SQL> create unique index partsupp_pk on h_partsupp(ps_partkey,ps_suppkey) parallel 2;

SQL> ALTER TABLE H_PARTSUPP ADD CONSTRAINT PARTSUPP_PK
        PRIMARY KEY(ps_partkey,ps_suppkey) using index PARTSUPP_PK;

SQL> create unique index PART_PK on H_PART(p_partkey) parallel 2;

SQL> ALTER TABLE H_PART ADD CONSTRAINT PART_PK
        PRIMARY KEY (p_partkey) using index PART_PK;

SQL> create unique index ORDERS_PK on H_ORDER(o_orderkey) parallel 2;

SQL> ALTER TABLE H_ORDER ADD CONSTRAINT ORDERS_PK
        PRIMARY KEY (o_orderkey) using index ORDERS_PK;

SQL> create unique index LINEITEM_PK on H_LINEITEM(l_linenumber, l_orderkey) parallel 2;

SQL> ALTER TABLE H_LINEITEM ADD CONSTRAINT LINEITEM_PK
        PRIMARY KEY (l_linenumber, l_orderkey)  using index LINEITEM_PK;
```

```
SQL> create unique index CUSTOMER_PK on H_CUSTOMER(c_custkey) parallel 2;

SQL> ALTER TABLE H_CUSTOMER ADD CONSTRAINT CUSTOMER_PK
        PRIMARY KEY (c_custkey) using index CUSTOMER_PK;

-- FK Constraints
SQL> ALTER TABLE H_LINEITEM
        ADD CONSTRAINT LINEITEM_PARTSUPP_FK FOREIGN KEY (l_partkey, l_suppkey)
        REFERENCES H_PARTSUPP(ps_partkey, ps_suppkey) NOT DEFERRABLE;

SQL> ALTER TABLE H_ORDER
        ADD CONSTRAINT ORDER_CUSTOMER_FK FOREIGN KEY (o_custkey)
        REFERENCES H_CUSTOMER (c_custkey) NOT DEFERRABLE;

SQL> ALTER TABLE H_PARTSUPP
        ADD CONSTRAINT PARTSUPP_PART_FK FOREIGN KEY (ps_partkey)
        REFERENCES H_PART (p_partkey) NOT DEFERRABLE;

SQL> ALTER TABLE H_PARTSUPP
        ADD CONSTRAINT PARTSUPP_SUPPLIER_FK FOREIGN KEY (ps_suppkey)
        REFERENCES H_SUPPLIER (s_suppkey) NOT DEFERRABLE;

SQL> ALTER TABLE H_SUPPLIER
        ADD CONSTRAINT SUPPLIER_NATION_FK FOREIGN KEY (s_nationkey)
        REFERENCES H_NATION (n_nationkey) NOT DEFERRABLE;

SQL> ALTER TABLE H_CUSTOMER
        ADD CONSTRAINT CUSTOMER_NATION_FK FOREIGN KEY (c_nationkey)
        REFERENCES H_NATION (n_nationkey) NOT DEFERRABLE;

SQL> ALTER TABLE H_NATION
        ADD CONSTRAINT NATION_REGION_FK FOREIGN KEY (n_regionkey)
        REFERENCES H_REGION (r_regionkey) NOT DEFERRABLE;

SQL> ALTER TABLE H_LINEITEM
        ADD CONSTRAINT LINEITEM_ORDER_FK FOREIGN KEY (l_orderkey)
        REFERENCES H_ORDER (o_orderkey) NOT DEFERRABLE;


SQL> exec dbms_stats.gather_schema_stats(ownname => 'TPCH',degree => 2,cascade => true);
```

Appendix C

Shell program that stresses the VM. This program is used in the *optimization* scheme.

```
#!/bin/ksh

Q1() {
q=`sqlplus -s tpch/tpch123 << END

SELECT L_RETURNFLAG,
L_LINESTATUS,
SUM(L_QUANTITY) AS SUM_QTY,
 SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-
L_DISCOUNT)) AS SUM_DISC_PRICE,
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
AVG(L_QUANTITY) AS AVG_QTY,
 AVG(L_EXTENDEDPRICE) AS AVG_PRICE,
 AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM H_LINEITEM
WHERE L_SHIPDATE <= to_date('1998-12-01','YYYY-MM-DD') - 90
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS;

EXIT
END`
}

Q2() {
q=`sqlplus -s tpch/tpch123 << END

SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY,
P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM
H_PART, H_SUPPLIER, H_PARTSUPP, H_NATION, H_REGION
WHERE P_PARTKEY = PS_PARTKEY
AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15
AND P_TYPE LIKE '%%BRASS'
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'EUROPE'
AND PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST)
        FROM H_PARTSUPP, H_SUPPLIER, H_NATION, H_REGION
        WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
        AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY
        AND R_NAME = 'EUROPE')
AND rownum<101
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY;

EXIT
END`
}

Q3() {
q=`sqlplus -s tpch/tpch123 << END

SELECT L_ORDERKEY,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
O_ORDERDATE, O_SHIPPRIORITY
FROM H_CUSTOMER, H_ORDER, H_LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING'
AND C_CUSTKEY = O_CUSTKEY
```

```
AND L_ORDERKEY = O_ORDERKEY
AND O_ORDERDATE < to_date('1995-03-15','YYYY-MM-DD')
AND L_SHIPDATE > to_date('1995-03-15','YYYY-MM-DD')
and rownum<10
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE ;

EXIT
END`
}

Q4() {
q=`sqlplus -s tpch/tpch123 << END

SELECT O_ORDERPRIORITY,
COUNT(*) AS ORDER_COUNT
FROM H_ORDER
WHERE O_ORDERDATE >= to_date('1993-07-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1993-07-01','YYYY-MM-DD'),3)
AND
EXISTS (SELECT * FROM H_LINEITEM
        WHERE L_ORDERKEY = O_ORDERKEY
        AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY;

EXIT
END`
}

Q5() {
q=`sqlplus -s tpch/tpch123 << END

SELECT N_NAME,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM H_CUSTOMER, H_ORDER, H_LINEITEM, H_SUPPLIER, H_NATION, H_REGION
WHERE C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND L_SUPPKEY = S_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA'
AND O_ORDERDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)
GROUP BY N_NAME
ORDER BY REVENUE DESC;

EXIT
END`
}

Q6() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM H_LINEITEM
WHERE L_SHIPDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND L_SHIPDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)
AND L_DISCOUNT BETWEEN .06 - 0.01 AND .06 + 0.01
AND L_QUANTITY < 24;

EXIT
```

```
END`
}

Q7() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUPP_NATION, CUST_NATION,
L_YEAR, SUM(VOLUME) AS REVENUE
FROM
        (SELECT N1.N_NAME AS SUPP_NATION,
        N2.N_NAME AS CUST_NATION,
        extract(year from L_SHIPDATE) AS L_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
        FROM H_SUPPLIER, H_LINEITEM, H_ORDER,
        H_CUSTOMER, H_NATION N1, H_NATION N2
        WHERE S_SUPPKEY = L_SUPPKEY
        AND O_ORDERKEY = L_ORDERKEY
        AND C_CUSTKEY = O_CUSTKEY
        AND S_NATIONKEY = N1.N_NATIONKEY
        AND C_NATIONKEY = N2.N_NATIONKEY
        AND ((N1.N_NAME = 'FRANCE' AND N2.N_NAME = 'GERMANY') OR
                (N1.N_NAME = 'GERMANY' AND N2.N_NAME = 'FRANCE'))
        AND L_SHIPDATE BETWEEN to_date('1995-01-01','YYYY-MM-DD') AND to_date('1996-12-
31','YYYY-MM-DD')) SHIPPING
GROUP BY SUPP_NATION, CUST_NATION, L_YEAR
ORDER BY SUPP_NATION, CUST_NATION, L_YEAR;

EXIT
END`
}

Q8() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
O_YEAR,
SUM(CASE WHEN NATION = 'BRAZIL' THEN VOLUME ELSE 0 END)/SUM(VOLUME) AS
MKT_SHARE
FROM
        (SELECT
        extract(year from O_ORDERDATE) AS O_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME,
        N2.N_NAME AS NATION
        FROM
        H_PART, H_SUPPLIER, H_LINEITEM, H_ORDER,
        H_CUSTOMER, H_NATION N1, H_NATION N2, H_REGION
        WHERE
        P_PARTKEY = L_PARTKEY
        AND S_SUPPKEY = L_SUPPKEY
        AND L_ORDERKEY = O_ORDERKEY
        AND O_CUSTKEY = C_CUSTKEY
        AND C_NATIONKEY = N1.N_NATIONKEY
        AND N1.N_REGIONKEY = R_REGIONKEY
        AND R_NAME = 'AMERICA'
        AND S_NATIONKEY = N2.N_NATIONKEY
        AND O_ORDERDATE BETWEEN to_date('1995-01-01','YYYY-MM-DD')
                AND to_date('1996-12-31','YYYY-MM-DD')
        AND P_TYPE= 'ECONOMY ANODIZED STEEL') ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR;

EXIT
END`
```

```
}

Q9() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
NATION, O_YEAR,
SUM(AMOUNT) AS SUM_PROFIT
FROM
        (SELECT N_NAME AS NATION, extract(year from O_ORDERDATE) AS O_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
        FROM
        H_PART, H_SUPPLIER, H_LINEITEM, H_PARTSUPP, H_ORDER, H_NATION
        WHERE S_SUPPKEY = L_SUPPKEY
        AND PS_SUPPKEY= L_SUPPKEY
        AND PS_PARTKEY = L_PARTKEY
        AND P_PARTKEY= L_PARTKEY
        AND O_ORDERKEY = L_ORDERKEY
        AND S_NATIONKEY = N_NATIONKEY
        AND P_NAME LIKE '%%green%%') PROFIT
GROUP BY NATION, O_YEAR
ORDER BY NATION, O_YEAR DESC;

EXIT
END`
}

Q10() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
C_CUSTKEY, C_NAME,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
C_ACCTBAL, N_NAME,
C_ADDRESS, C_PHONE, C_COMMENT
FROM H_CUSTOMER, H_ORDER, H_LINEITEM, H_NATION
WHERE C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND O_ORDERDATE>= to_date('1993-10-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1993-10-01','YYYY-MM-DD'),3)
AND L_RETURNFLAG = 'R' AND C_NATIONKEY = N_NATIONKEY
and rownum<21
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS,
C_COMMENT
ORDER BY REVENUE DESC;

EXIT
END`
}

Q11() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
PS_PARTKEY,
SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM
H_PARTSUPP, H_SUPPLIER, H_NATION
WHERE
PS_SUPPKEY = S_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_NAME = 'GERMANY'
GROUP BY PS_PARTKEY
```

```
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) > (SELECT
SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000
        FROM H_PARTSUPP, H_SUPPLIER, H_NATION
        WHERE PS_SUPPKEY = S_SUPPKEY
        AND S_NATIONKEY = N_NATIONKEY
        AND N_NAME = 'GERMANY')
ORDER BY VALUE DESC;

EXIT
END`
}

Q12() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
L_SHIPMODE,
SUM(CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR O_ORDERPRIORITY = '2-HIGH'
THEN 1 ELSE 0 END) AS HIGH_LINE_COUNT,
SUM(CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND O_ORDERPRIORITY <> '2-HIGH'
THEN 1 ELSE 0 END ) AS LOW_LINE_COUNT
FROM H_ORDER, H_LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND L_SHIPMODE IN ('MAIL','SHIP')
AND L_COMMITDATE < L_RECEIPTDATE
AND L_SHIPDATE < L_COMMITDATE
AND L_RECEIPTDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND L_RECEIPTDATE < add_months(to_date('1995-09-01','YYYY-MM-DD'),1)
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE;

EXIT
END`
}

Q13() {
q=`sqlplus -s tpch/tpch123 << END

SELECT C_COUNT, COUNT(*) AS CUSTDIST
FROM (SELECT C_CUSTKEY, COUNT(O_ORDERKEY) as C_COUNT
        FROM H_CUSTOMER left outer join H_ORDER on C_CUSTKEY = O_CUSTKEY
        AND O_COMMENT not like '%%special%%requests%%'
        GROUP BY C_CUSTKEY) C_ORDERS
GROUP BY C_COUNT
ORDER BY CUSTDIST DESC, C_COUNT DESC;

EXIT
END`
}

Q14() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
100.00*SUM(CASE WHEN P_TYPE LIKE 'PROMO%%' THEN L_EXTENDEDPRICE*(1-
L_DISCOUNT)
ELSE 0 END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM
H_LINEITEM, H_PART
WHERE
L_PARTKEY = P_PARTKEY
AND L_SHIPDATE >= to_date('1995-09-01','YYYY-MM-DD')
AND L_SHIPDATE < add_months(to_date('1995-09-01','YYYY-MM-DD'),1);
```

```
EXIT
END`
}

Q15() {
q=`sqlplus -s tpch/tpch123 << END

CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE) AS
SELECT L_SUPPKEY,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
FROM H_LINEITEM
WHERE L_SHIPDATE >= to_date('1996-01-01','YYYY-MM-DD')
AND L_SHIPDATE < add_months(to_date('1996-01-01','YYYY-MM-DD'),3)
GROUP BY L_SUPPKEY;

SELECT
S_SUPPKEY, S_NAME, S_ADDRESS, S_PHONE, TOTAL_REVENUE
FROM H_SUPPLIER, REVENUE0
WHERE
S_SUPPKEY = SUPPLIER_NO
AND TOTAL_REVENUE = (SELECT MAX(TOTAL_REVENUE) FROM REVENUE0)
ORDER BY S_SUPPKEY;

DROP VIEW REVENUE0;

EXIT
END`
}

Q16() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
P_BRAND, P_TYPE, P_SIZE,
COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM
H_PARTSUPP, H_PART
WHERE
P_PARTKEY = PS_PARTKEY
AND P_BRAND <> 'Brand#45'
AND P_TYPE NOT LIKE 'MEDIUM POLISHED%%'
AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9)
AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY
        FROM H_SUPPLIER
        WHERE S_COMMENT LIKE '%%Customer%%Complaints%%')
GROUP BY P_BRAND, P_TYPE, P_SIZE
ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE;

EXIT
END`
}

Q17() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY
FROM
H_LINEITEM, H_PART
WHERE
P_PARTKEY = L_PARTKEY
AND P_BRAND = 'Brand#23'
```

```
AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY < (SELECT 0.2*AVG(L_QUANTITY)
FROM H_LINEITEM WHERE L_PARTKEY = P_PARTKEY);

EXIT
END`
}

Q18() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE,
SUM(L_QUANTITY)
FROM
H_CUSTOMER, H_ORDER, H_LINEITEM
WHERE O_ORDERKEY IN (SELECT
        L_ORDERKEY
        FROM H_LINEITEM
        GROUP BY L_ORDERKEY
        HAVING SUM(L_QUANTITY) > 300)
AND C_CUSTKEY = O_CUSTKEY
AND O_ORDERKEY = L_ORDERKEY
AND rownum<101
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE;

EXIT
END`
}

Q19() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM H_LINEITEM, H_PART
WHERE
(P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12'
        AND P_CONTAINER IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10
        AND P_SIZE BETWEEN 1 AND 5
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND ='Brand#23'
        AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        AND L_QUANTITY >=10 AND L_QUANTITY <=10 + 10
        AND P_SIZE BETWEEN 1 AND 10
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34'
        AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND L_QUANTITY >=20 AND L_QUANTITY <= 20 + 10
        AND P_SIZE BETWEEN 1 AND 15
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON');

EXIT
END`
}

Q20() {
```

```
q=`sqlplus -s tpch/tpch123 << END

SELECT
S_NAME, S_ADDRESS
FROM
H_SUPPLIER, H_NATION
WHERE
S_SUPPKEY IN (SELECT
        PS_SUPPKEY FROM H_PARTSUPP
        WHERE PS_PARTKEY in (SELECT
                P_PARTKEY FROM H_PART
                WHERE P_NAME like 'forest%%')
        AND PS_AVAILQTY > (SELECT 0.5*sum(L_QUANTITY)
                FROM H_LINEITEM
                WHERE L_PARTKEY = PS_PARTKEY
                AND L_SUPPKEY = PS_SUPPKEY
                AND L_SHIPDATE >= to_date('1994-01-01','YYYY-MM-DD')
                AND L_SHIPDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)))
AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'CANADA'
ORDER BY S_NAME;

EXIT
END`
}

Q21() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
S_NAME, COUNT(*) AS NUMWAIT
FROM
H_SUPPLIER, H_LINEITEM L1, H_ORDER, H_NATION
WHERE
S_SUPPKEY = L1.L_SUPPKEY
AND O_ORDERKEY = L1.L_ORDERKEY
AND O_ORDERSTATUS = 'F'
AND L1.L_RECEIPTDATE> L1.L_COMMITDATE
AND EXISTS (SELECT * FROM
        H_LINEITEM L2
        WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
        AND L2.L_SUPPKEY <> L1.L_SUPPKEY)
AND NOT EXISTS (SELECT *
        FROM H_LINEITEM L3
        WHERE L3.L_ORDERKEY = L1.L_ORDERKEY
        AND L3.L_SUPPKEY <> L1.L_SUPPKEY
        AND L3.L_RECEIPTDATE > L3.L_COMMITDATE)
AND S_NATIONKEY = N_NATIONKEY
AND N_NAME = 'SAUDI ARABIA'
and rownum<101
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME;

EXIT
END`
}

Q22() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
CNTRYCODE,
COUNT(*) AS NUMCUST,
SUM(C_ACCTBAL) AS TOTACCTBAL
```

```
FROM (SELECT SUBSTR(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL
        FROM H_CUSTOMER
        WHERE
        SUBSTR(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17')
        AND C_ACCTBAL > (SELECT AVG(C_ACCTBAL)
                FROM H_CUSTOMER WHERE C_ACCTBAL > 0.00
                AND SUBSTR(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17'))
                AND NOT EXISTS (SELECT *
                        FROM H_ORDER WHERE O_CUSTKEY = C_CUSTKEY)) CUSTSALE
GROUP BY CNTRYCODE
ORDER BY CNTRYCODE;

EXIT
END`
}


SQL_ID=" \
Q3-4y7ucx9354fxy
Q22-5bks84w8ut3dy
"

# Q1-b383b8ptd6m38
# Q3-4y7ucx9354fxy
# Q4-5rpbt92d2w4ks
# Q5-8y0yasa5zjyr1
# Q6-1zn3xrx01mtck
# Q7-2tryzag0xbu4m
# Q8-c8bp67faftkh2
# Q9-bccvz740py3dv
# Q10-2bkjqzpz3ubsc
# Q11-9fw9rgatw0h2b
# Q12-94tpbact4tt8c
# Q13-9f2czfz2pm9zr
# Q14-0c2bha5xd99js
# Q16-9f16buakax45p
# Q17-33fsxr05jhazw
# Q18-ctakajmsjp98s
# Q19-14yf8frfjbcry
# Q20-302hwrypt1g02
# Q21-3z61g4q8uhvac
# Q22-5bks84w8ut3dy

a=2520
b=0

while [ $a -ge 0 ];do
        for sql in $SQL_ID;do
                R=`echo "$sql" | awk -F'-' '{print $1}'`
                S=`echo "$sql" | awk -F'-' '{print $2}'`

                q=`sqlplus -s "tpch/tpch123" <<END
                set heading off
                set feedback off
                select status, sid from v\\$session where sql_id='${S}'
                and program like '%(TNS%'
                and schemaname='TPCH'
                ;
END`
                p=`echo "$q" | grep . | wc -l`

                if [ "$a" -eq 2520 ];then
                        r=6
```

```
                elif [ "$a" -eq 2000 ];then
                        r=5
                elif [ "$a" -eq 1600 ];then
                        r=4
                elif [ "$a" -eq 1200 ];then
                        r=3
                elif [ "$a" -eq 800 ];then
                        r=2
                elif [ "$a" -eq 400 ];then
                        r=1
                fi

                if [ "$p" -lt "$r" ];then
                        $R &
                fi

        done
        sleep 1
        a=`expr $a - 1`
        b=`expr $b + 1`
        echo "$a"

        if [ "$b" -eq 120 ];then
                ./awr &
        fi
done

a=`ps -ef  |grep awr | awk '{print $2}' | sed "s/^/kill -9 /g"`
echo "$a" > kill_awr
./kill_awr
> kill_awr
```

Appendix D

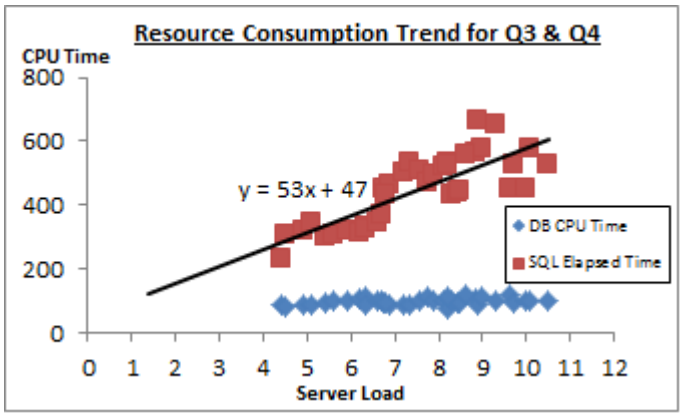
Shell program that filters out the unreliable data and triggers the data extractor. This program is used in the *optimization* scheme.

```
#!/bin/ksh


q=`sqlplus -s tpch/tpch123 << END
        set feedback off
        set heading off
        select max(snap_id) from dba_hist_osstat;
EXIT
END`

a=9876
b=`expr $a + 1`
c=`echo $q`

echo $c

while [ "$a" -le "$c" ];do
        p=`sqlplus -s tpch/tpch123 <<END
        set feedback off
        set heading off
        select nvl((e1.value - b1.value),-1)/1000000
                from dba_hist_sys_time_model  e1
                        , dba_hist_sys_time_model  b1
                where b1.instance_number        = e1.instance_number
                and b1.stat_name            = 'sql execute elapsed time'
                and b1.stat_id              = e1.stat_id
                 and b1.snap_id               = ${a}
          and e1.snap_id              = ${b};
END`

        q=`sqlplus -s tpch/tpch123 <<END
        set feedback off
        set heading off
        select nvl((e1.value - b1.value),-1)/1000000
                from dba_hist_sys_time_model  e1
                        , dba_hist_sys_time_model  b1
                where b1.instance_number        = e1.instance_number
                and b1.stat_name            = 'DB CPU'
                and b1.stat_id              = e1.stat_id
                 and b1.snap_id               = ${a}
          and e1.snap_id              = ${b};
END`

        r=`sqlplus -s tpch/tpch123 <<END
        set feedback off
        set heading off
        select value from dba_hist_osstat where stat_name='LOAD' and snap_id=${a};
END`

        s=`sqlplus -s tpch/tpch123 <<END
        set feedback off
        set heading off
        select value from dba_hist_osstat where stat_name='LOAD' and snap_id=${b};
END`

        H=1.10
```

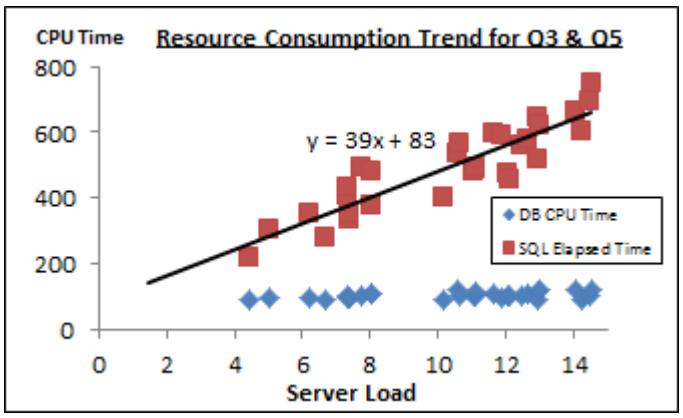
```
        L=0.90

        r=`echo "$r" | grep . | sed "s/  *//g" | sed "s/          //g"`
        s=`echo "$s" | grep . | sed "s/  *//g" | sed "s/          //g"`

        s1=$( echo "scale=2; (${r} * ${H})" | bc)
        s2=$( echo "scale=2; (${r} * ${L})" | bc)

        if [[ "$s" -le "$s1" ]] && [[ "$s" -ge "$s2" ]];then
                t=$( echo "scale=2; (${r} + ${s}) / 2" | bc)
                echo ${t} ${q} ${p} | sed "s/          / /g" | sed "s/  */ /g"
        fi

        a=`expr $a + 1`
        b=`expr $b + 1`
done
```

Appendix E

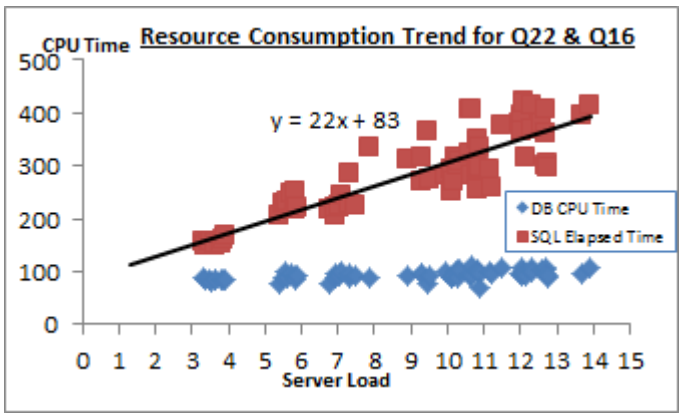
Following figures show the constructed baselines for *optimization* scheme.



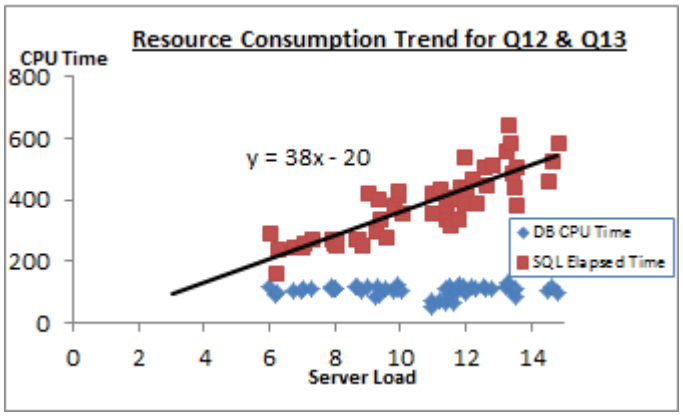
Testing result from iterative run of combined query #3 & 4.



Testing result from iterative run of combined query #3 & 5.



Testing result from iterative run of combined query #22 & 16.



Testing result from iterative run of combined query #12 & 13.

Appendix F

Shell program that produces the benchmark data using single TPC-H query. This program is used in the *affirmation* scheme.

```
#!/bin/ksh

Q1() {
q=`sqlplus -s tpch/tpch123 << END

SELECT L_RETURNFLAG,
L_LINESTATUS,
SUM(L_QUANTITY) AS SUM_QTY,
 SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-
L_DISCOUNT)) AS SUM_DISC_PRICE,
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
AVG(L_QUANTITY) AS AVG_QTY,
 AVG(L_EXTENDEDPRICE) AS AVG_PRICE,
 AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM H_LINEITEM
WHERE L_SHIPDATE <= to_date('1998-12-01','YYYY-MM-DD') - 90
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS;

EXIT
END`
}

Q2() {
q=`sqlplus -s tpch/tpch123 << END

SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY,
P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM
H_PART, H_SUPPLIER, H_PARTSUPP, H_NATION, H_REGION
WHERE P_PARTKEY = PS_PARTKEY
AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15
AND P_TYPE LIKE '%%BRASS'
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'EUROPE'
AND PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST)
        FROM H_PARTSUPP, H_SUPPLIER, H_NATION, H_REGION
        WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
        AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY
        AND R_NAME = 'EUROPE')
AND rownum<101
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY;

EXIT
END`
}

Q3() {
q=`sqlplus -s tpch/tpch123 << END

SELECT L_ORDERKEY,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
O_ORDERDATE, O_SHIPPRIORITY
FROM H_CUSTOMER, H_ORDER, H_LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING'
```

```
AND C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND O_ORDERDATE < to_date('1995-03-15','YYYY-MM-DD')
AND L_SHIPDATE > to_date('1995-03-15','YYYY-MM-DD')
and rownum<10
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE ;

EXIT
END`
}

Q4() {
q=`sqlplus -s tpch/tpch123 << END

SELECT O_ORDERPRIORITY,
COUNT(*) AS ORDER_COUNT
FROM H_ORDER
WHERE O_ORDERDATE >= to_date('1993-07-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1993-07-01','YYYY-MM-DD'),3)
AND
EXISTS (SELECT * FROM H_LINEITEM
        WHERE L_ORDERKEY = O_ORDERKEY
        AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY;

EXIT
END`
}

Q5() {
q=`sqlplus -s tpch/tpch123 << END

SELECT N_NAME,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM H_CUSTOMER, H_ORDER, H_LINEITEM, H_SUPPLIER, H_NATION, H_REGION
WHERE C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND L_SUPPKEY = S_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA'
AND O_ORDERDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)
GROUP BY N_NAME
ORDER BY REVENUE DESC;

EXIT
END`
}

Q6() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM H_LINEITEM
WHERE L_SHIPDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND L_SHIPDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)
AND L_DISCOUNT BETWEEN .06 - 0.01 AND .06 + 0.01
AND L_QUANTITY < 24;
```

```
EXIT
END`
}

Q7() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUPP_NATION, CUST_NATION,
L_YEAR, SUM(VOLUME) AS REVENUE
FROM
        (SELECT N1.N_NAME AS SUPP_NATION,
        N2.N_NAME AS CUST_NATION,
        extract(year from L_SHIPDATE) AS L_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
        FROM H_SUPPLIER, H_LINEITEM, H_ORDER,
        H_CUSTOMER, H_NATION N1, H_NATION N2
        WHERE S_SUPPKEY = L_SUPPKEY
        AND O_ORDERKEY = L_ORDERKEY
        AND C_CUSTKEY = O_CUSTKEY
        AND S_NATIONKEY = N1.N_NATIONKEY
        AND C_NATIONKEY = N2.N_NATIONKEY
        AND ((N1.N_NAME = 'FRANCE' AND N2.N_NAME = 'GERMANY') OR
                (N1.N_NAME = 'GERMANY' AND N2.N_NAME = 'FRANCE'))
        AND L_SHIPDATE BETWEEN to_date('1995-01-01','YYYY-MM-DD') AND to_date('1996-12-
31','YYYY-MM-DD')) SHIPPING
GROUP BY SUPP_NATION, CUST_NATION, L_YEAR
ORDER BY SUPP_NATION, CUST_NATION, L_YEAR;

EXIT
END`
}

Q8() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
O_YEAR,
SUM(CASE WHEN NATION = 'BRAZIL' THEN VOLUME ELSE 0 END)/SUM(VOLUME) AS
MKT_SHARE
FROM
        (SELECT
        extract(year from O_ORDERDATE) AS O_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME,
        N2.N_NAME AS NATION
        FROM
        H_PART, H_SUPPLIER, H_LINEITEM, H_ORDER,
        H_CUSTOMER, H_NATION N1, H_NATION N2, H_REGION
        WHERE
        P_PARTKEY = L_PARTKEY
        AND S_SUPPKEY = L_SUPPKEY
        AND L_ORDERKEY = O_ORDERKEY
        AND O_CUSTKEY = C_CUSTKEY
        AND C_NATIONKEY = N1.N_NATIONKEY
        AND N1.N_REGIONKEY = R_REGIONKEY
        AND R_NAME = 'AMERICA'
        AND S_NATIONKEY = N2.N_NATIONKEY
        AND O_ORDERDATE BETWEEN to_date('1995-01-01','YYYY-MM-DD')
                AND to_date('1996-12-31','YYYY-MM-DD')
        AND P_TYPE= 'ECONOMY ANODIZED STEEL') ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR;

EXIT
```

```
END`
}

Q9() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
NATION, O_YEAR,
SUM(AMOUNT) AS SUM_PROFIT
FROM
        (SELECT N_NAME AS NATION, extract(year from O_ORDERDATE) AS O_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
        FROM
        H_PART, H_SUPPLIER, H_LINEITEM, H_PARTSUPP, H_ORDER, H_NATION
        WHERE S_SUPPKEY = L_SUPPKEY
        AND PS_SUPPKEY= L_SUPPKEY
        AND PS_PARTKEY = L_PARTKEY
        AND P_PARTKEY= L_PARTKEY
        AND O_ORDERKEY = L_ORDERKEY
        AND S_NATIONKEY = N_NATIONKEY
        AND P_NAME LIKE '%%green%%') PROFIT
GROUP BY NATION, O_YEAR
ORDER BY NATION, O_YEAR DESC;

EXIT
END`
}

Q10() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
C_CUSTKEY, C_NAME,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
C_ACCTBAL, N_NAME,
C_ADDRESS, C_PHONE, C_COMMENT
FROM H_CUSTOMER, H_ORDER, H_LINEITEM, H_NATION
WHERE C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND O_ORDERDATE>= to_date('1993-10-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1993-10-01','YYYY-MM-DD'),3)
AND L_RETURNFLAG = 'R' AND C_NATIONKEY = N_NATIONKEY
and rownum<21
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS,
C_COMMENT
ORDER BY REVENUE DESC;

EXIT
END`
}

Q11() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
PS_PARTKEY,
SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM
H_PARTSUPP, H_SUPPLIER, H_NATION
WHERE
PS_SUPPKEY = S_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_NAME = 'GERMANY'
```

```
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) > (SELECT
SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000
        FROM H_PARTSUPP, H_SUPPLIER, H_NATION
        WHERE PS_SUPPKEY = S_SUPPKEY
        AND S_NATIONKEY = N_NATIONKEY
        AND N_NAME = 'GERMANY')
ORDER BY VALUE DESC;

EXIT
END`
}

Q12() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
L_SHIPMODE,
SUM(CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR O_ORDERPRIORITY = '2-HIGH'
THEN 1 ELSE 0 END) AS HIGH_LINE_COUNT,
SUM(CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND O_ORDERPRIORITY <> '2-HIGH'
THEN 1 ELSE 0 END ) AS LOW_LINE_COUNT
FROM H_ORDER, H_LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND L_SHIPMODE IN ('MAIL','SHIP')
AND L_COMMITDATE < L_RECEIPTDATE
AND L_SHIPDATE < L_COMMITDATE
AND L_RECEIPTDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND L_RECEIPTDATE < add_months(to_date('1995-09-01','YYYY-MM-DD'),1)
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE;

EXIT
END`
}

Q13() {
q=`sqlplus -s tpch/tpch123 << END

SELECT C_COUNT, COUNT(*) AS CUSTDIST
FROM (SELECT C_CUSTKEY, COUNT(O_ORDERKEY) as C_COUNT
        FROM H_CUSTOMER left outer join H_ORDER on C_CUSTKEY = O_CUSTKEY
        AND O_COMMENT not like '%%special%%requests%%'
        GROUP BY C_CUSTKEY) C_ORDERS
GROUP BY C_COUNT
ORDER BY CUSTDIST DESC, C_COUNT DESC;

EXIT
END`
}

Q14() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
100.00*SUM(CASE WHEN P_TYPE LIKE 'PROMO%%' THEN L_EXTENDEDPRICE*(1-
L_DISCOUNT)
ELSE 0 END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM
H_LINEITEM, H_PART
WHERE
L_PARTKEY = P_PARTKEY
AND L_SHIPDATE >= to_date('1995-09-01','YYYY-MM-DD')
```

```
AND L_SHIPDATE < add_months(to_date('1995-09-01','YYYY-MM-DD'),1);

EXIT
END`
}

Q15() {
q=`sqlplus -s tpch/tpch123 << END

-- CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE) AS
-- SELECT L_SUPPKEY,
-- SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
-- FROM H_LINEITEM
-- WHERE L_SHIPDATE >= to_date('1996-01-01','YYYY-MM-DD')
-- AND L_SHIPDATE < add_months(to_date('1996-01-01','YYYY-MM-DD'),3)
-- GROUP BY L_SUPPKEY;

SELECT
S_SUPPKEY, S_NAME, S_ADDRESS, S_PHONE, TOTAL_REVENUE
FROM H_SUPPLIER, REVENUE0
WHERE
S_SUPPKEY = SUPPLIER_NO
AND TOTAL_REVENUE = (SELECT MAX(TOTAL_REVENUE) FROM REVENUE0)
ORDER BY S_SUPPKEY;

-- DROP VIEW REVENUE0;

EXIT
END`
}

Q16() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
P_BRAND, P_TYPE, P_SIZE,
COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM
H_PARTSUPP, H_PART
WHERE
P_PARTKEY = PS_PARTKEY
AND P_BRAND <> 'Brand#45'
AND P_TYPE NOT LIKE 'MEDIUM POLISHED%%'
AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9)
AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY
        FROM H_SUPPLIER
        WHERE S_COMMENT LIKE '%%Customer%%Complaints%%')
GROUP BY P_BRAND, P_TYPE, P_SIZE
ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE;

EXIT
END`
}

Q17() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY
FROM
H_LINEITEM, H_PART
WHERE
P_PARTKEY = L_PARTKEY
```

```
AND P_BRAND = 'Brand#23'
AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY < (SELECT 0.2*AVG(L_QUANTITY)
FROM H_LINEITEM WHERE L_PARTKEY = P_PARTKEY);

EXIT
END`
}

Q18() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE,
SUM(L_QUANTITY)
FROM
H_CUSTOMER, H_ORDER, H_LINEITEM
WHERE O_ORDERKEY IN (SELECT
        L_ORDERKEY
        FROM H_LINEITEM
        GROUP BY L_ORDERKEY
        HAVING SUM(L_QUANTITY) > 300)
AND C_CUSTKEY = O_CUSTKEY
AND O_ORDERKEY = L_ORDERKEY
AND rownum<101
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE;

EXIT
END`
}

Q19() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM H_LINEITEM, H_PART
WHERE
(P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12'
        AND P_CONTAINER IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10
        AND P_SIZE BETWEEN 1 AND 5
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND ='Brand#23'
        AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        AND L_QUANTITY >=10 AND L_QUANTITY <=10 + 10
        AND P_SIZE BETWEEN 1 AND 10
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34'
        AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND L_QUANTITY >=20 AND L_QUANTITY <= 20 + 10
        AND P_SIZE BETWEEN 1 AND 15
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON');

EXIT
END`
}
```

```
Q20() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
S_NAME, S_ADDRESS
FROM
H_SUPPLIER, H_NATION
WHERE
S_SUPPKEY IN (SELECT
        PS_SUPPKEY FROM H_PARTSUPP
        WHERE PS_PARTKEY in (SELECT
                P_PARTKEY FROM H_PART
                WHERE P_NAME like 'forest%%')
        AND PS_AVAILQTY > (SELECT 0.5*sum(L_QUANTITY)
                FROM H_LINEITEM
                WHERE L_PARTKEY = PS_PARTKEY
                AND L_SUPPKEY = PS_SUPPKEY
                AND L_SHIPDATE >= to_date('1994-01-01','YYYY-MM-DD')
                AND L_SHIPDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)))
AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'CANADA'
ORDER BY S_NAME;

EXIT
END`
}

Q21() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
S_NAME, COUNT(*) AS NUMWAIT
FROM
H_SUPPLIER, H_LINEITEM L1, H_ORDER, H_NATION
WHERE
S_SUPPKEY = L1.L_SUPPKEY
AND O_ORDERKEY = L1.L_ORDERKEY
AND O_ORDERSTATUS = 'F'
AND L1.L_RECEIPTDATE> L1.L_COMMITDATE
AND EXISTS (SELECT * FROM
        H_LINEITEM L2
        WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
        AND L2.L_SUPPKEY <> L1.L_SUPPKEY)
AND NOT EXISTS (SELECT *
        FROM H_LINEITEM L3
        WHERE L3.L_ORDERKEY = L1.L_ORDERKEY
        AND L3.L_SUPPKEY <> L1.L_SUPPKEY
        AND L3.L_RECEIPTDATE > L3.L_COMMITDATE)
AND S_NATIONKEY = N_NATIONKEY
AND N_NAME = 'SAUDI ARABIA'
and rownum<101
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME;

EXIT
END`
}

Q22() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
CNTRYCODE,
COUNT(*) AS NUMCUST,
```

```
SUM(C_ACCTBAL) AS TOTACCTBAL
FROM (SELECT SUBSTR(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL
        FROM H_CUSTOMER
        WHERE
        SUBSTR(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17')
        AND C_ACCTBAL > (SELECT AVG(C_ACCTBAL)
                FROM H_CUSTOMER WHERE C_ACCTBAL > 0.00
                AND SUBSTR(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17'))
                AND NOT EXISTS (SELECT *
                        FROM H_ORDER WHERE O_CUSTKEY = C_CUSTKEY)) CUSTSALE
GROUP BY CNTRYCODE
ORDER BY CNTRYCODE;

EXIT
END`
}


SQL_ID=" \
Q9-bccvz740py3dv
"

# Q1-b383b8ptd6m38
# Q2-84u9xq2p56f68
# Q3-4y7ucx9354fxy
# Q4-5rpbt92d2w4ks
# Q5-8y0yasa5zjyr1
# Q6-1zn3xrx01mtck
# Q7-2tryzag0xbu4m
# Q8-c8bp67faftkh2
# Q9-bccvz740py3dv
# Q10-2bkjqzpz3ubsc
# Q11-9fw9rgatw0h2b
# Q12-94tpbact4tt8c
# Q13-9f2czfz2pm9zr
# Q14-0c2bha5xd99js
# Q15-9fj78vapy7uny
# Q16-9f16buakax45p
# Q17-33fsxr05jhazw
# Q18-ctakajmsjp98s
# Q19-14yf8frfjbcry
# Q20-302hwrypt1g02
# Q21-3z61g4q8uhvac
# Q22-5bks84w8ut3dy

snap() {
sqlplus -s tpch/tpch123 << END
        set feedback off
        set heading off
        EXEC dbms_workload_repository.create_snapshot;
        select max(snap_id) from dba_hist_sys_time_model
        where DBID=(select dbid from v\$database);
END
}

CG_CHK() {
sqlplus -s tpch/tpch123 << END
        set heading off
        set feedback off
        select value from v\$sysstat
        where name='consistent gets';
END
}
```

```
b=0
baseline_ratio=40870
load_threshold=4.5
run_freq=5 # from MATLAB
begin_time=`echo $(date +%s)`
sleep 1

begin_CG=`CG_CHK`

while (true);do
	elapsed_clock_time=`expr $(date +%s) - ${begin_time}`


	CURR_CG_TOTAL=`CG_CHK`
	CURR_CG_DIFF=`echo $((${CURR_CG_TOTAL} - ${begin_CG}))`

	CG_ratio=`echo $((${CURR_CG_DIFF} / ${elapsed_clock_time}))`
	# echo "CURR_CG_DIFF=" $CURR_CG_DIFF
	echo "elapsed_clock_time=" $elapsed_clock_time
	echo "CG_ratio=" $CG_ratio
	echo "baseline_ratio=" $baseline_ratio

	R=`echo "$SQL_ID" | awk -F'-' '{print $1}'`
	S=`echo "$SQL_ID" | awk -F'-' '{print $2}'`
	echo $S
	server_load=`uptime | awk '{print $10}' | awk -F',' '{print $1}'`
	echo "server_load=" $server_load

	if [[ ${CG_ratio} -lt ${baseline_ratio} ]] \
		&& [[ ${server_load} -le ${load_threshold} ]];then
		SQL_CNT=`sqlplus -s tpch/tpch123 << END
			set heading off
			set feedback off
			select count(*) from v\\$session
			where sql_id='\${S}';
END`
		echo $SQL_CNT
		echo $run_freq
		x=0
		while [[ ${run_freq} -gt 0 ]] && \
			[[ ${SQL_CNT} -eq 0 ]];do
			$R > /dev/null 2>&1 &
			echo "abc"
			run_freq=`expr $run_freq - 1`
			x=`expr $x + 1`
			echo "SQL_CNT=" $SQL_CNT
			echo "x=" $x
			echo "run_freq=" $run_freq
		done
	fi
	sleep 5
	run_freq=5
done
```

Appendix G

Shell program that produces the benchmark data using multiple TPC-H queries. This program is used in the *affirmation* scheme.

```
#!/bin/ksh

Q1() {
q=`sqlplus -s tpch/tpch123 << END

SELECT L_RETURNFLAG,
L_LINESTATUS,
SUM(L_QUANTITY) AS SUM_QTY,
 SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-
L_DISCOUNT)) AS SUM_DISC_PRICE,
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
AVG(L_QUANTITY) AS AVG_QTY,
 AVG(L_EXTENDEDPRICE) AS AVG_PRICE,
 AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM H_LINEITEM
WHERE L_SHIPDATE <= to_date('1998-12-01','YYYY-MM-DD') - 90
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS;

EXIT
END`
}

Q2() {
q=`sqlplus -s tpch/tpch123 << END

SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY,
P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM
H_PART, H_SUPPLIER, H_PARTSUPP, H_NATION, H_REGION
WHERE P_PARTKEY = PS_PARTKEY
AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15
AND P_TYPE LIKE '%%BRASS'
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'EUROPE'
AND PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST)
        FROM H_PARTSUPP, H_SUPPLIER, H_NATION, H_REGION
        WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
        AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY
        AND R_NAME = 'EUROPE')
AND rownum<101
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY;

EXIT
END`
}

Q3() {
q=`sqlplus -s tpch/tpch123 << END

SELECT L_ORDERKEY,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
O_ORDERDATE, O_SHIPPRIORITY
FROM H_CUSTOMER, H_ORDER, H_LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING'
```

```
AND C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND O_ORDERDATE < to_date('1995-03-15','YYYY-MM-DD')
AND L_SHIPDATE > to_date('1995-03-15','YYYY-MM-DD')
and rownum<10
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE ;

EXIT
END`
}

Q4() {
q=`sqlplus -s tpch/tpch123 << END

SELECT O_ORDERPRIORITY,
COUNT(*) AS ORDER_COUNT
FROM H_ORDER
WHERE O_ORDERDATE >= to_date('1993-07-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1993-07-01','YYYY-MM-DD'),3)
AND
EXISTS (SELECT * FROM H_LINEITEM
        WHERE L_ORDERKEY = O_ORDERKEY
        AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY;

EXIT
END`
}

Q5() {
q=`sqlplus -s tpch/tpch123 << END

SELECT N_NAME,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM H_CUSTOMER, H_ORDER, H_LINEITEM, H_SUPPLIER, H_NATION, H_REGION
WHERE C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND L_SUPPKEY = S_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA'
AND O_ORDERDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)
GROUP BY N_NAME
ORDER BY REVENUE DESC;

EXIT
END`
}

Q6() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM H_LINEITEM
WHERE L_SHIPDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND L_SHIPDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)
AND L_DISCOUNT BETWEEN .06 - 0.01 AND .06 + 0.01
AND L_QUANTITY < 24;
```

```
EXIT
END`
}

Q7() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUPP_NATION, CUST_NATION,
L_YEAR, SUM(VOLUME) AS REVENUE
FROM
        (SELECT N1.N_NAME AS SUPP_NATION,
        N2.N_NAME AS CUST_NATION,
        extract(year from L_SHIPDATE) AS L_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
        FROM H_SUPPLIER, H_LINEITEM, H_ORDER,
        H_CUSTOMER, H_NATION N1, H_NATION N2
        WHERE S_SUPPKEY = L_SUPPKEY
        AND O_ORDERKEY = L_ORDERKEY
        AND C_CUSTKEY = O_CUSTKEY
        AND S_NATIONKEY = N1.N_NATIONKEY
        AND C_NATIONKEY = N2.N_NATIONKEY
        AND ((N1.N_NAME = 'FRANCE' AND N2.N_NAME = 'GERMANY') OR
                (N1.N_NAME = 'GERMANY' AND N2.N_NAME = 'FRANCE'))
        AND L_SHIPDATE BETWEEN to_date('1995-01-01','YYYY-MM-DD') AND to_date('1996-12-
31','YYYY-MM-DD')) SHIPPING
GROUP BY SUPP_NATION, CUST_NATION, L_YEAR
ORDER BY SUPP_NATION, CUST_NATION, L_YEAR;

EXIT
END`
}

Q8() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
O_YEAR,
SUM(CASE WHEN NATION = 'BRAZIL' THEN VOLUME ELSE 0 END)/SUM(VOLUME) AS
MKT_SHARE
FROM
        (SELECT
        extract(year from O_ORDERDATE) AS O_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME,
        N2.N_NAME AS NATION
        FROM
        H_PART, H_SUPPLIER, H_LINEITEM, H_ORDER,
        H_CUSTOMER, H_NATION N1, H_NATION N2, H_REGION
        WHERE
        P_PARTKEY = L_PARTKEY
        AND S_SUPPKEY = L_SUPPKEY
        AND L_ORDERKEY = O_ORDERKEY
        AND O_CUSTKEY = C_CUSTKEY
        AND C_NATIONKEY = N1.N_NATIONKEY
        AND N1.N_REGIONKEY = R_REGIONKEY
        AND R_NAME = 'AMERICA'
        AND S_NATIONKEY = N2.N_NATIONKEY
        AND O_ORDERDATE BETWEEN to_date('1995-01-01','YYYY-MM-DD')
                AND to_date('1996-12-31','YYYY-MM-DD')
        AND P_TYPE= 'ECONOMY ANODIZED STEEL') ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR;

EXIT
```

```
END`
}

Q9() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
NATION, O_YEAR,
SUM(AMOUNT) AS SUM_PROFIT
FROM
        (SELECT N_NAME AS NATION, extract(year from O_ORDERDATE) AS O_YEAR,
        L_EXTENDEDPRICE*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
        FROM
        H_PART, H_SUPPLIER, H_LINEITEM, H_PARTSUPP, H_ORDER, H_NATION
        WHERE S_SUPPKEY = L_SUPPKEY
        AND PS_SUPPKEY= L_SUPPKEY
        AND PS_PARTKEY = L_PARTKEY
        AND P_PARTKEY= L_PARTKEY
        AND O_ORDERKEY = L_ORDERKEY
        AND S_NATIONKEY = N_NATIONKEY
        AND P_NAME LIKE '%%green%%') PROFIT
GROUP BY NATION, O_YEAR
ORDER BY NATION, O_YEAR DESC;

EXIT
END`
}

Q10() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
C_CUSTKEY, C_NAME,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
C_ACCTBAL, N_NAME,
C_ADDRESS, C_PHONE, C_COMMENT
FROM H_CUSTOMER, H_ORDER, H_LINEITEM, H_NATION
WHERE C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY
AND O_ORDERDATE>= to_date('1993-10-01','YYYY-MM-DD')
AND O_ORDERDATE < add_months(to_date('1993-10-01','YYYY-MM-DD'),3)
AND L_RETURNFLAG = 'R' AND C_NATIONKEY = N_NATIONKEY
and rownum<21
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS,
C_COMMENT
ORDER BY REVENUE DESC;

EXIT
END`
}

Q11() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
PS_PARTKEY,
SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM
H_PARTSUPP, H_SUPPLIER, H_NATION
WHERE
PS_SUPPKEY = S_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_NAME = 'GERMANY'
```

```
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) > (SELECT
SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000
        FROM H_PARTSUPP, H_SUPPLIER, H_NATION
        WHERE PS_SUPPKEY = S_SUPPKEY
        AND S_NATIONKEY = N_NATIONKEY
        AND N_NAME = 'GERMANY')
ORDER BY VALUE DESC;

EXIT
END`
}

Q12() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
L_SHIPMODE,
SUM(CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR O_ORDERPRIORITY = '2-HIGH'
THEN 1 ELSE 0 END) AS HIGH_LINE_COUNT,
SUM(CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND O_ORDERPRIORITY <> '2-HIGH'
THEN 1 ELSE 0 END ) AS LOW_LINE_COUNT
FROM H_ORDER, H_LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND L_SHIPMODE IN ('MAIL','SHIP')
AND L_COMMITDATE < L_RECEIPTDATE
AND L_SHIPDATE < L_COMMITDATE
AND L_RECEIPTDATE >= to_date('1994-01-01','YYYY-MM-DD')
AND L_RECEIPTDATE < add_months(to_date('1995-09-01','YYYY-MM-DD'),1)
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE;

EXIT
END`
}

Q13() {
q=`sqlplus -s tpch/tpch123 << END

SELECT C_COUNT, COUNT(*) AS CUSTDIST
FROM (SELECT C_CUSTKEY, COUNT(O_ORDERKEY) as C_COUNT
        FROM H_CUSTOMER left outer join H_ORDER on C_CUSTKEY = O_CUSTKEY
        AND O_COMMENT not like '%%special%%requests%%'
        GROUP BY C_CUSTKEY) C_ORDERS
GROUP BY C_COUNT
ORDER BY CUSTDIST DESC, C_COUNT DESC;

EXIT
END`
}

Q14() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
100.00*SUM(CASE WHEN P_TYPE LIKE 'PROMO%%' THEN L_EXTENDEDPRICE*(1-
L_DISCOUNT)
ELSE 0 END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM
H_LINEITEM, H_PART
WHERE
L_PARTKEY = P_PARTKEY
AND L_SHIPDATE >= to_date('1995-09-01','YYYY-MM-DD')
```

```
AND L_SHIPDATE < add_months(to_date('1995-09-01','YYYY-MM-DD'),1);

EXIT
END`
}

Q15() {
q=`sqlplus -s tpch/tpch123 << END

-- CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE) AS
-- SELECT L_SUPPKEY,
-- SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
-- FROM H_LINEITEM
-- WHERE L_SHIPDATE >= to_date('1996-01-01','YYYY-MM-DD')
-- AND L_SHIPDATE < add_months(to_date('1996-01-01','YYYY-MM-DD'),3)
-- GROUP BY L_SUPPKEY;

SELECT
S_SUPPKEY, S_NAME, S_ADDRESS, S_PHONE, TOTAL_REVENUE
FROM H_SUPPLIER, REVENUE0
WHERE
S_SUPPKEY = SUPPLIER_NO
AND TOTAL_REVENUE = (SELECT MAX(TOTAL_REVENUE) FROM REVENUE0)
ORDER BY S_SUPPKEY;

-- DROP VIEW REVENUE0;

EXIT
END`
}

Q16() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
P_BRAND, P_TYPE, P_SIZE,
COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM
H_PARTSUPP, H_PART
WHERE
P_PARTKEY = PS_PARTKEY
AND P_BRAND <> 'Brand#45'
AND P_TYPE NOT LIKE 'MEDIUM POLISHED%%'
AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9)
AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY
        FROM H_SUPPLIER
        WHERE S_COMMENT LIKE '%%Customer%%Complaints%%')
GROUP BY P_BRAND, P_TYPE, P_SIZE
ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE;

EXIT
END`
}

Q17() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY
FROM
H_LINEITEM, H_PART
WHERE
P_PARTKEY = L_PARTKEY
```

```
AND P_BRAND = 'Brand#23'
AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY < (SELECT 0.2*AVG(L_QUANTITY)
FROM H_LINEITEM WHERE L_PARTKEY = P_PARTKEY);

EXIT
END`
}

Q18() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE,
SUM(L_QUANTITY)
FROM
H_CUSTOMER, H_ORDER, H_LINEITEM
WHERE O_ORDERKEY IN (SELECT
        L_ORDERKEY
        FROM H_LINEITEM
        GROUP BY L_ORDERKEY
        HAVING SUM(L_QUANTITY) > 300)
AND C_CUSTKEY = O_CUSTKEY
AND O_ORDERKEY = L_ORDERKEY
AND rownum<101
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE;

EXIT
END`
}

Q19() {
q=`sqlplus -s tpch/tpch123 << END

SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM H_LINEITEM, H_PART
WHERE
(P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12'
        AND P_CONTAINER IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10
        AND P_SIZE BETWEEN 1 AND 5
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND ='Brand#23'
        AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        AND L_QUANTITY >=10 AND L_QUANTITY <=10 + 10
        AND P_SIZE BETWEEN 1 AND 10
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR
(P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34'
        AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND L_QUANTITY >=20 AND L_QUANTITY <= 20 + 10
        AND P_SIZE BETWEEN 1 AND 15
        AND L_SHIPMODE IN ('AIR', 'AIR REG')
        AND L_SHIPINSTRUCT = 'DELIVER IN PERSON');

EXIT
END`
}
```

```
Q20() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
S_NAME, S_ADDRESS
FROM
H_SUPPLIER, H_NATION
WHERE
S_SUPPKEY IN (SELECT
        PS_SUPPKEY FROM H_PARTSUPP
        WHERE PS_PARTKEY in (SELECT
                P_PARTKEY FROM H_PART
                WHERE P_NAME like 'forest%%')
        AND PS_AVAILQTY > (SELECT 0.5*sum(L_QUANTITY)
                FROM H_LINEITEM
                WHERE L_PARTKEY = PS_PARTKEY
                AND L_SUPPKEY = PS_SUPPKEY
                AND L_SHIPDATE >= to_date('1994-01-01','YYYY-MM-DD')
                AND L_SHIPDATE < add_months(to_date('1994-01-01','YYYY-MM-DD'),1*12)))
AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'CANADA'
ORDER BY S_NAME;

EXIT
END`
}

Q21() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
S_NAME, COUNT(*) AS NUMWAIT
FROM
H_SUPPLIER, H_LINEITEM L1, H_ORDER, H_NATION
WHERE
S_SUPPKEY = L1.L_SUPPKEY
AND O_ORDERKEY = L1.L_ORDERKEY
AND O_ORDERSTATUS = 'F'
AND L1.L_RECEIPTDATE> L1.L_COMMITDATE
AND EXISTS (SELECT * FROM
        H_LINEITEM L2
        WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
        AND L2.L_SUPPKEY <> L1.L_SUPPKEY)
AND NOT EXISTS (SELECT *
        FROM H_LINEITEM L3
        WHERE L3.L_ORDERKEY = L1.L_ORDERKEY
        AND L3.L_SUPPKEY <> L1.L_SUPPKEY
        AND L3.L_RECEIPTDATE > L3.L_COMMITDATE)
AND S_NATIONKEY = N_NATIONKEY
AND N_NAME = 'SAUDI ARABIA'
and rownum<101
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME;

EXIT
END`
}

Q22() {
q=`sqlplus -s tpch/tpch123 << END

SELECT
CNTRYCODE,
COUNT(*) AS NUMCUST,
```

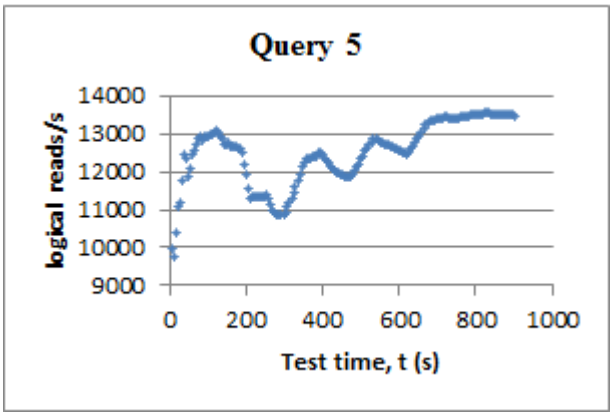
```
SUM(C_ACCTBAL) AS TOTACCTBAL
FROM (SELECT SUBSTR(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL
        FROM H_CUSTOMER
        WHERE
        SUBSTR(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17')
        AND C_ACCTBAL > (SELECT AVG(C_ACCTBAL)
                FROM H_CUSTOMER WHERE C_ACCTBAL > 0.00
                AND SUBSTR(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17'))
                AND NOT EXISTS (SELECT *
                        FROM H_ORDER WHERE O_CUSTKEY = C_CUSTKEY)) CUSTSALE
GROUP BY CNTRYCODE
ORDER BY CNTRYCODE;

EXIT
END`
}


SQL_ID=" \
Q8-c8bp67faftkh2
"
SQL_ID2=" \
Q9-bccvz740py3dv
"

# Q1-b383b8ptd6m38
# Q2-84u9xq2p56f68
# Q3-4y7ucx9354fxy
# Q4-5rpbt92d2w4ks
# Q5-8y0yasa5zjyr1
# Q6-1zn3xrx01mtck
# Q7-2tryzag0xbu4m
# Q8-c8bp67faftkh2
# Q9-bccvz740py3dv
# Q10-2bkjqzpz3ubsc
# Q11-9fw9rgatw0h2b
# Q12-94tpbact4tt8c
# Q13-9f2czfz2pm9zr
# Q14-0c2bha5xd99js
# Q15-9fj78vapy7uny
# Q16-9f16buakax45p
# Q17-33fsxr05jhazw
# Q18-ctakajmsjp98s
# Q19-14yf8frfjbcry
# Q20-302hwrypt1g02
# Q21-3z61g4q8uhvac
# Q22-5bks84w8ut3dy

snap() {
sqlplus -s tpch/tpch123 << END
        set feedback off
        set heading off
        EXEC dbms_workload_repository.create_snapshot;
        select max(snap_id) from dba_hist_sys_time_model
        where DBID=(select dbid from v\$database);
END
}

CG_CHK() {
sqlplus -s tpch/tpch123 << END
        set heading off
        set feedback off
        select value from v\$sysstat
```

```
        where name='consistent gets';
END
}

b=0
baseline_ratio=40870
load_threshold=4.5
run_freq=3
run_freq2=3
begin_time=`echo $(date +%s)`
sleep 1

begin_CG=`CG_CHK`

while (true);do
    elapsed_clock_time=`expr $(date +%s) - ${begin_time}`

    CURR_CG_TOTAL=`CG_CHK`
    CURR_CG_DIFF=`echo $((${CURR_CG_TOTAL} - ${begin_CG}))`

    CG_ratio=`echo $((${CURR_CG_DIFF} / ${elapsed_clock_time}))`
    # echo "CURR_CG_DIFF=" $CURR_CG_DIFF
    echo "elapsed_clock_time=" $elapsed_clock_time
    echo "CG_ratio=" $CG_ratio
    echo "baseline_ratio=" $baseline_ratio

    R=`echo "$SQL_ID" | awk -F'-' '{print $1}'`
    R2=`echo "$SQL_ID2" | awk -F'-' '{print $1}'`
    S=`echo "$SQL_ID" | awk -F'-' '{print $2}'`
    S2=`echo "$SQL_ID2" | awk -F'-' '{print $2}'`

    echo $S
    echo $S2
    server_load=`uptime | awk '{print $10}' | awk -F',' '{print $1}'`
    echo "server_load=" $server_load

    if [[ ${CG_ratio} -lt ${baseline_ratio} ]] \
        && [[ ${server_load} -le ${load_threshold} ]];then
        SQL_CNT=`sqlplus -s tpch/tpch123 << END
            set heading off
            set feedback off
            select count(*) from v\\$session
            where sql_id='\${S}';
END`
        x=0
        while [[ ${run_freq} -gt 0 ]] && \
            [[ ${SQL_CNT} -eq 0 ]];do
            $R > /dev/null 2>&1 &
            echo "abc"
            run_freq=`expr $run_freq - 1`
            x=`expr $x + 1`
            echo "SQL_CNT=" $SQL_CNT
            echo "x=" $x
            echo "run_freq=" $run_freq
        done
    fi

    if [[ ${CG_ratio} -lt ${baseline_ratio} ]] \
        && [[ ${server_load} -le ${load_threshold} ]];then
        SQL_CNT2=`sqlplus -s tpch/tpch123 << END
            set heading off
            set feedback off
            select count(*) from v\\$session
```

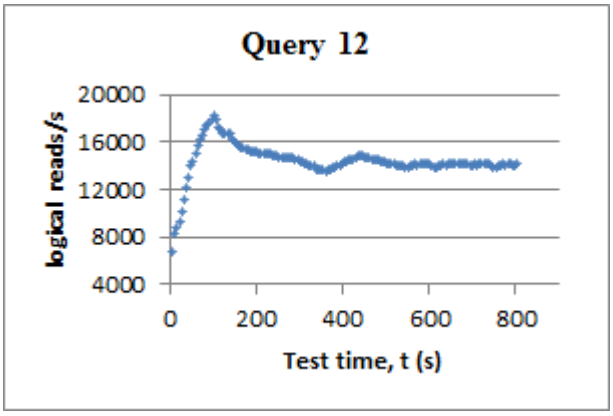
```
                where sql_id='\${S2}';
END`
        y=0
        while [[ ${run_freq2} -gt 0 ]] && \
            [[ ${SQL_CNT2} -eq 0 ]];do
            $R2 > /dev/null 2>&1 &
            echo "xyz"
            run_freq2=`expr $run_freq2 - 1`
            y=`expr $y + 1`
            echo "SQL_CNT2=" $SQL_CNT2
            echo "y=" $y
            echo "run_freq2=" $run_freq2
        done
    fi
    sleep 5
    run_freq=3
    run_freq2=3
done
```

Appendix H

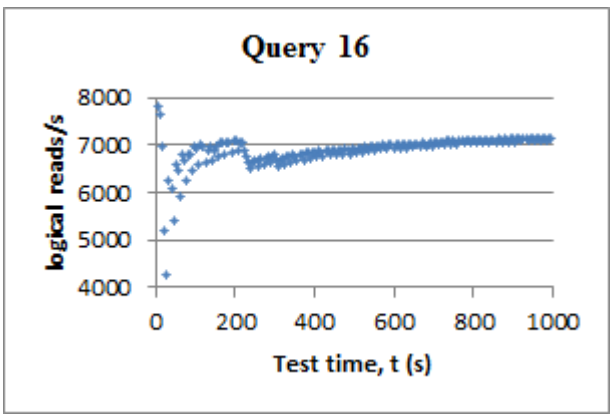
Following figures exhibit the generated benchmark data for *affirmation* scheme. The tests were conducted with CPU run queue size kept at 4, and parallel execution of 8.



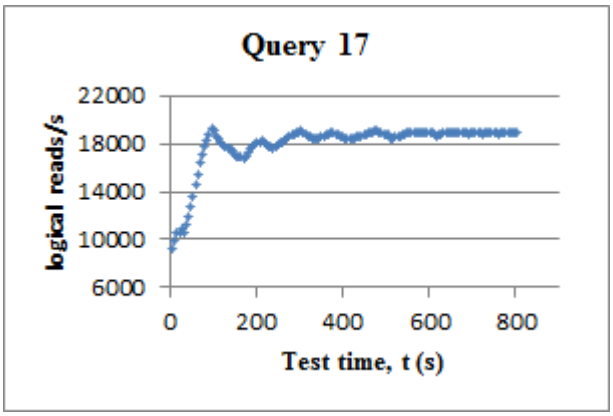
Testing result from iterative and parallel execution of TPC-H query #5.



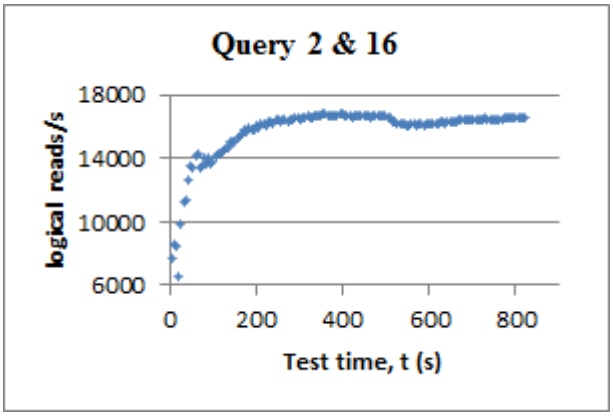
Testing result from iterative and parallel execution of TPC-H query #12.



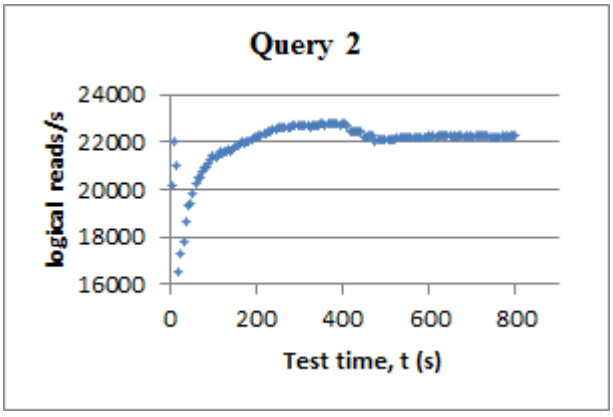
Testing result from iterative and parallel execution of TPC-H query #16.



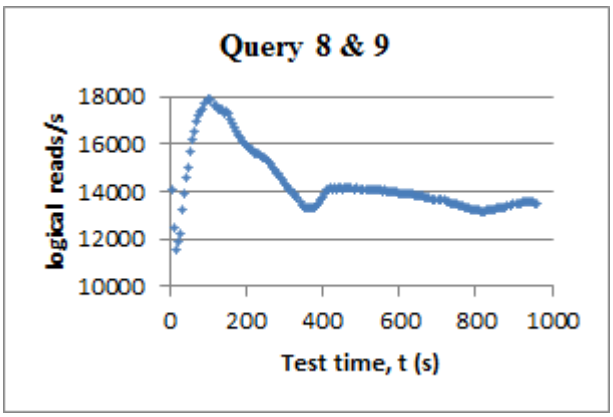
Testing result from iterative and parallel execution of TPC-H query #17.



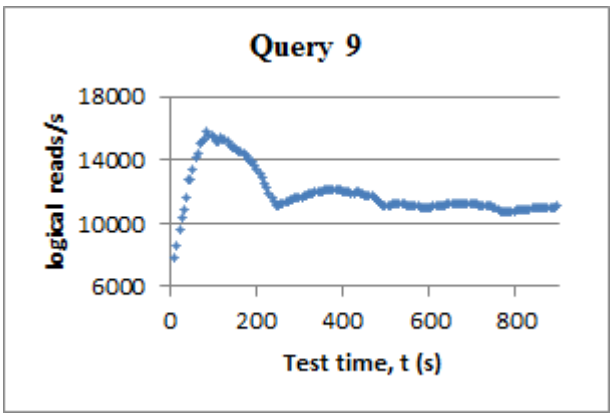
Testing result from iterative and parallel execution of combination of TPC-H query # 2 & 16.



Testing result from iterative and parallel execution of TPC-H query #2.



Testing result from iterative and parallel execution of combination of TPC-H query #8 & 9.



Testing result from iterative and parallel execution of TPC-H query #9.

Appendix I

Published journal papers:

Tan, C. H., & Teh, Y. W. (2013a). Harnessing Cloud Computing for Dynamic Resource Requirement by Database Workloads. *Journal of Information Science and Engineering, 29*(5). (ISI-Cited Publication)

Tan, C. H., & Teh, Y. W. (2013c). Synthetic Hardware Performance Analysis in Virtualized Cloud Environment for Healthcare Organization. *Journal of Medical Systems, 37*(4). (ISI-Cited Publication)

Tan, C. H., & Teh, Y. W. (2013b). Secure Hardware Performance Analysis in Virtualized Cloud Environment. *Mathematical Problems in Engineering*. (ISI-Cited Publication)

Published conference proceedings:

Tan, C. H., & Teh, Y. W. (2013). Hardware Resource Performance Optimization and Affirmation in Virtualized Cloud Environment for Healthcare Organization. *International Conference on Medical Innovation and Computing Services*. *Tainan City, Taiwan.*

Tan, C. H., & Teh, Y. W. (2011). A Fuzzy way to Index Tuning. *2nd International Conference on Management Science and Engineering*. *Chengdou,China.*

Journal citation report (2012):

Journal name	**Publisher**	**Category Name**	**Total Journals in Category**	**Journal Rank in Category**	**Quartile in Category**	**Impact Factor**
Journal of Information Science and Engineering	Institute Information Science	Computer Science, Information Systems	132	121	Q4	0.299
Journal of Medical Systems	Springer	Health Care Sciences & Services	82	36	Q2	1.783
Mathematical Problems in Engineering	Hindawi Publishing Corporation	Mathematics, Interdisciplinary Applications	92	29	Q2	1.383