

PARALLEL PROCESSING FOR DATA RETRIEVAL IN
ODOO ENTERPRISE RESOURCE PLANNING REPORTING
SYSTEM

ROUA ABDELMUNIEM OSMAN ALHAG EISA

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

2021

**PARALLEL PROCESSING FOR DATA RETRIEVAL IN ODOO
ENTERPRISE RESOURCE PLANNING REPORTING SYSTEM**

ROUA ABDELMUNIEM OSMAN ALHAG EISA

**DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF MASTER OF
SOFTWARE ENGINEERING
(SOFTWARE TECHNOLOGY)**

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

2021

UNIVERSITY OF MALAYA
ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Roua Abdelmuniem Osman Alhag Eisa

Matric No: WOC160016

Name of Degree: Master of Software Engineering (Software Technology)

Title of Dissertation ("this Work"): Parallel Processing for Data Retrieval in
Odoo Enterprise Resource Planning Reporting System

Field of Study: Data Mining

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 Roua

Date: 12/04/2021

Subscribed and solemnly declared before,

Witness's Signature

Date:

Name:

Designation:

**PARALLEL PROCESSING FOR DATA RETRIEVAL IN ODOO
ENTERPRISE RESOURCE PLANNING REPORTING SYSTEM**

ABSTRACT

Reporting process in Enterprise Resource Planning (ERP) system plays an important role, as different information from different processes can be merged to generate reports. Management can use these reports for providing key value indicators for progress assessment, as well as the identification of poor business performance and the formulation of strategies to eliminate them. Odoo framework, previously known as OpenERP, is the most commonly installed open source ERP system worldwide. During the ERP system lifetime massive data generated from the daily operations, most implemented open source ERP systems such as the Odoo framework are using Relational Database Management System (RDBMS) as data storage, while the amount of the data increases this traditional data analysis, processing and storage technologies are not capable enough to store and/or process a large amount of data effectively and the performance became an issue as the relational database applies sequential data processing. This performance latency has an implication on overall system performance, concurrent users' sessions, business processing, and report processing which all affect organization processes and decision making to achieve business goals. Report processing time increases while the number of data increases due to data retrieving from the relational database, where the more data are processed; the more time it needs to generate a report. This research aims to solve Odoo's reporting latency problem, where the proposed solution is to import data from the Odoo database and store it in NoSQL data storage to perform parallel data processing to generate the required report faster than the existing approaches to generating the same report. The applied research methodology comprises several steps which include a literature review that discusses the previous ERP system comparisons, existing reporting approaches and

the successful deployment of parallel data processing in various domains. Another step is preliminary experiment conduct to compare the performance of generating sale orders report using the existed approaches, the remain steps discuss the design, development and evaluation of the research proposed solution. The research results find out that the parallel data retrieval used in the developed solution shows performance improvement over sequential data retrieval used in existed approaches. Organizations with a large scale (500000 records and above per table) can get significant reporting performance improvement which has a direct impact on an organization's processes, achieve insights into business data, forecasting, decision support and to meet business goals.

Keywords: ERP, reporting, performance, parallel data processing.

PEMROSESAN SELARI UNTUK PENGAMBILAN DATA DALAM SISTEM PELAPORAN PERANCANGAN SUMBER ODOO ENTERPRISE

ABSTRAK

Proses pelaporan dalam sistem Perancangan Sumber Daya Perusahaan (ERP) memainkan peranan yang penting kerana ia mempunyai maklumat yang berbeza daripada proses pelaporan biasa. Pihak pengurusan boleh menggunakan laporan tesis ini untuk menjadi penunjuk nilai utama bagi penilaian kemajuan dan juga mengenalpasti prestasi perniagaan yang lemah bagi merangka strategi proses penghapusannya. Rangka kerja Odoo, yang sebelumnya dikenali sebagai OpenERP, adalah sistem ERP sumber terbuka yang paling kerap dipasang di seluruh dunia. Ketika sistem ERP menghasilkan data besar dalam operasi hariannya, kebanyakan sistem ERP sumber terbuka seperti kerangka Odoo menggunakan Sistem Pengurusan Pangkalan Data Relasional (RDBMS) sebagai penyimpanan data. Semasa jumlah data meningkat ketika menganalisis dan memproses data tradisional, teknologi penyimpanan tidak cukup mampu untuk menyimpan dan / atau memproses sejumlah besar data dengan berkesan yang menyebabkan prestasi menjadi masalah disebabkan oleh pangkalan data hubungan menggunakan pemprosesan data berturutan. Latensi prestasi ini mempunyai implikasi pada keseluruhan prestasi sistem, sesi pengguna bersamaan, pemprosesan perniagaan, dan pemprosesan laporan yang semuanya mempengaruhi proses organisasi dan membuat keputusan untuk mencapai tujuan perniagaan. Masa pemprosesan laporan meningkat selari dengan peningkatan jumlah data disebabkan oleh data yang diambil dari pangkalan data hubungan, di mana lebih banyak data yang akan diproses, lebih banyak masa perlu menghasilkan laporan. Penyelidikan ini bertujuan menyelesaikan masalah latensi pelaporan Odoo, di mana penyelesaian yang dicadangkan adalah untuk mengimport data dari pangkalan data Odoo dan menyimpannya dalam penyimpanan

data NoSQL untuk melakukan pemrosesan data selari untuk menghasilkan laporan yang diperlukan lebih cepat berbanding pendekatan sedia ada untuk menghasilkan laporan yang sama. Metodologi penyelidikan yang diaplikasikan merangkumi beberapa langkah yang merangkumi tinjauan literatur yang membincangkan perbandingan sistem ERP sebelumnya, pendekatan pelaporan yang ada dan keberhasilan penggunaan pemrosesan data selari dalam pelbagai domain. Langkah lain adalah melakukan eksperimen awal untuk membandingkan prestasi menghasilkan laporan pesanan penjualan menggunakan pendekatan yang ada, langkah-langkah yang lain membincangkan reka bentuk, pengembangan dan penilaian penyelesaian yang dicadangkan oleh penyelidikan. Hasil penyelidikan mendapati bahawa pengambilan data selari yang digunakan dalam penyelesaian yang dikembangkan menunjukkan peningkatan prestasi berbanding pengambilan data berurutan yang digunakan dalam pendekatan yang ada. Organisasi dengan skala besar (500000 rekod dan ke atas per jadual) dapat memperoleh peningkatan prestasi pelaporan yang signifikan yang mempunyai kesan langsung pada proses organisasi, mencapai pandangan mengenai data perniagaan, ramalan, sokongan keputusan dan untuk memenuhi tujuan perniagaan.

Katakunci: Perancangan Sumber Perusahaan, pelaporan, prestasi, pemrosesan data selari.

ACKNOWLEDGEMENTS

Thanks to God Almighty for the favour and wisdom to undertake this research. Special thanks of gratitude to my supervisors Associate Prof. Dr. Siti Hafizah Binti Ab Hamid and Dr. Mumtaz Begun Mustafa who helped me with the useful comments, remarks, guidance through the learning process of this dissertation. To all the members of Faculty of Computer Science and Information Technology (FCSIT), University of Malaya, I appreciate your contribution to my study.

Secondly, I would also like to thank my parents, and my husband who helped me a lot in finalizing this research within the limited time frame, thank you for your much-needed support and encouragement.

TABLE OF CONTENTS

Abstract.....	iii
Abstrak.....	v
Acknowledgements.....	vii
Table of Contents	viii
List of Figures.....	xiii
List of Tables	xv
List of Symbols and Abbreviations.....	xvi
 CHAPTER 1: INTRODUCTION.....	 1
1.1 Overview.....	1
1.2 Research Background	2
1.3 Research Motivation.....	3
1.4 Research Problem Statement	3
1.5 Research Objectives.....	4
1.5.1 Research Objectives	4
1.5.2 Research Questions	5
1.6 Research Scope.....	5
1.7 Research Expected Outcomes.....	6
1.8 Research Significant	6
1.9 Thesis Organization	6
 CHAPTER 2: LITERATURE REVIEW.....	 8
2.1 Enterprise Resource Planning (ERP) System	8
2.1.1 ERP Main Components	15

2.2	Odoo ERP System	16
2.2.1	Odoo Framework.....	17
2.2.2	Odoo Architecture	17
2.2.2.1	Models	17
2.2.2.2	Views	18
2.2.2.3	Controllers.....	18
2.3	Odoo Report.....	20
2.3.1	Odoo Reporting process	21
2.3.2	Odoo Reporting Engine.....	21
2.3.2.1	OpenOffice	22
2.3.2.2	ReportLab.....	22
2.3.2.3	OpenERP Webkit Report	22
2.3.2.4	QWeb Template	22
2.4	NoSQL Database Storage and Parallel Processing.....	26
2.4.1	Parallel Data Processing.....	27
2.4.2	Applications of Parallel Data Processing	27
2.4.3	Hadoop Framework.....	30
2.4.3.1	HDFS Architecture.....	30
2.4.3.2	MapReduce.....	31
2.4.3.3	Hadoop Ecosystems	32
(a)	HBase	32
(b)	Sqoop	33
(c)	Phoenix.....	33
2.4.4	Report Processing in Hadoop	34

2.5	Summary of literature	34
CHAPTER 3: RESEARCH METHODOLOGY		36
3.1	Literature Review	36
3.2	Data Collection	38
3.3	Preliminary Experiment.....	42
3.4	Design and Development.....	45
3.4.1	Design.....	45
3.4.2	Development	46
3.5	Experiment and Evaluation.....	48
3.5.1	Experimental design	48
3.5.2	Evaluation method.....	50
3.6	Summary.....	50
CHAPTER 4: DESIGN		51
4.1	System Architecture.....	51
4.2	UML Diagram	52
4.2.1	Use Case Diagram	52
4.2.2	Activity Diagram.....	53
4.2.3	Flowchart Diagram.....	54
4.3	Development Tools and Environment.....	54
4.4	HBase Database Design, Connection and Access	55
4.5	Phoenix Connection using Phoenixdb.....	57
4.6	Summary.....	57
CHAPTER 5: DEVELOPMENT		58

5.1	Environmental setup	58
5.1.1	Java Installation	58
5.1.2	Hadoop installation and configuration	59
5.1.3	HBase Installation	60
5.1.4	Phoenix Installation	61
5.1.5	Sqoop Installation	61
5.1.6	Installing Thrift API	61
5.1.7	Installing HappyBase API	62
5.2	Data Migration:.....	62
5.3	Development of HBase Module in Odoo	63
5.3.1	HBase connection configuration	63
5.3.2	Data Synchronization	64
5.3.2.1	Real-time synchronization.....	65
5.3.2.2	Scheduled Synchronization	66
5.3.3	Phoenix Connection using PhoenixDB	67
5.4	Implementing Methods in Custom Module.....	69
5.4.1	Custom Module:	69
5.5	Summary.....	71
CHAPTER 6: EXPERIMENTS, EVALUATION AND RESULTS		72
6.1	Experiments and Evaluation	72
6.1.1	Retrieval Time:.....	72
6.1.2	Number of Records	73
6.1.3	Query Execution Time	74

6.2	Results	74
6.2.1	Generating Reports	74
6.2.2	Query Execution Time	75
6.3	Discussion	77
6.4	Summary	78
CHAPTER 7: CONCLUSION		79
7.1	State-of- the-art	79
7.2	Fulfilment of Research Aims, Objectives and Questions	79
7.2.1	Research Objectives	80
7.2.1.1	First Objective	80
7.2.1.2	Second objective	80
7.2.1.3	Third objective	80
7.2.2	Research Questions	81
7.2.2.1	First Question	81
7.2.2.2	Second Question	81
7.2.2.3	Third Question	82
7.3	Contribution	82
7.4	Work Limitation	82
7.5	For Future Research	83
References		85

LIST OF FIGURES

Figure 2.1: MVC architecture (Ganesh et al., 2016).....	19
Figure 2.2: Odoo MVC architecture (Vaja & Rahevar, 2016).....	19
Figure 2.3: Reporting process in Odoo framework (Reis, 2016).....	21
Figure 3.1: Research methodology process flow.	36
Figure 3.2: Sales order form.....	38
Figure 3.3: generate_series() function in INSERT statement.	39
Figure 3.4: Preliminary Experiment SQL query.	43
Figure 3.5: Existing approaches experiments results graph.....	45
Figure 3.6: Development phases.	47
Figure 3.7: Import data into HBase from PostgreSQL.	47
Figure 3.8: HBase connection UI.....	48
Figure 4.1: System architecture.....	52
Figure 4.2: Use Case diagram.	52
Figure 4.3: Activity diagram.	53
Figure 4.4: Flowchart diagram.....	54
Figure 4.5: Sales orders stored in HBase.	56
Figure 5.1: Installed Java version.....	58
Figure 5.2: Sqoop import command.....	62
Figure 5.3: HBase scan.	63
Figure 5.4: HBase configuration class.	63
Figure 5.5: HBase configuration view XML.	64
Figure 5.6: HBase configuration UI form.	64

Figure 5.7: Real-time synchronization methods.	65
Figure 5.8: Scheduler function to synchronize transaction.	66
Figure 5.9: Interval period for scheduler function in XML file.	66
Figure 5.10: Scheduled actions UI form.	67
Figure 5.11: Phoenix configuration class.	67
Figure 5.12: Phoenix configuration view XML.	68
Figure 5.13: Phoenix configuration UI form.	68
Figure 5.14: Phoenix execute query method.	69
Figure 5.15: Custom sale orders report.	70
Figure 5.16: ORM methods inheritance in sale order class.	70
Figure 5.17: ORM methods inheritance in sale order class.	70
Figure 6.1: Experiment SQL query.	72
Figure 6.2: Approaches experiments results graph.	75
Figure 6.3: Query execution time graph in PostgreSQL and HBase.	76

LIST OF TABLES

Table 2.1: Leading open source ERP systems.	9
Table 2.2: ERP systems comparison (Ganesh et al., 2016).	10
Table 2.3: Comparison between ERP systems (Ganesh et al., 2016; Kowanda et al., 2015).	11
Table 2.4: All Approaches developed in Odoo report engine.....	23
Table 2.5: Existing solution to generate reports in Odoo.....	25
Table 2.6: Applying parallel processing in different domains.	29
Table 3.1: Sale order table.	39
Table 3.2: Customer table.	40
Table 3.3: Sale order line table.	41
Table 3.4: Product table.	41
Table 3.5: Experiments' results for the three existing approaches.	44
Table 3.6: Experiments details.....	49
Table 3.7: Query execution experiment table.	49
Table 6.1: Sample of sale orders dataset.	73
Table 6.2: Experiments.	73
Table 6.3: Approaches experiments results.	74
Table 6.4: Query execution time in PostgreSQL and HBase.....	76

LIST OF SYMBOLS AND ABBREVIATIONS

ERP	:	Enterprise Resource Planning
RDBMS	:	Relational Database Management System
HRP	:	Horseradish peroxidase
SQL	:	Structured Query Language
NoSQL	:	Not Only SQL
SAP	:	Systems Applications and Products
RAM	:	Random Access Memory
API	:	Application Programming Interface
SME	:	Small-to-Medium Enterprise
App	:	Application
MVC	:	Model View Controller architecture
UI	:	User Interface
XML	:	eXtensible Markup Language
ORM	:	Object Relational Mapping
CRUD	:	Create, Read, Update and Delete on database
HTML	:	Hypertext Markup Language

SXW	:	Sun Xml Writer
RML	:	Right Middle Lobe
PIR	:	Right Middle Lobe
CPU	:	Central Processing Unit
GFS	:	Google File System
HDFS	:	Hadoop Distributed File System
MR	:	Map Reduce
MPI	:	Message Passing Interface
Sqoop	:	SQL to Hadoop
HBase	:	Hadoop Database
UML	:	Unified Modeling Language
REST	:	Representational State Transfer API
JDK	:	Java Development Kit
URL	:	Universal Resource Location
IP	:	Internet Protocol

CHAPTER 1: INTRODUCTION

1.1 Overview

In recent years, the computer system and its applications have revolutionized the world. Enterprise Resource Planning (ERP) system has been adopted by many organizations all over the world, especially after the emergence of open source-based ERP system with less cost. This makes the ERP implementation applicable for all organizations regardless of their sizes. Integrating subsystems, automating processes and procedures, automating workflows, dashboards, and reports; are all components provided by the ERP system to improve the organization's business process in meeting its business goals (Nah, Lau, & Kuang, 2001).

Reports play an important role in the organization's success, all data generated from automated processes should be converted to meaningful information as regular reports to executive levels whereas to help the organization keeps track the income and expenses which are important for the organization's current progress assessment and set new goals as well. The analytics reports also help to figure out business lacking and defects and they improve strategies in eliminating them (Dezdar & Ainin, 2011).

During the lifetime of the ERP running system, massive data are generated which are involved in the daily operations. Thus, processing large amount of data using the traditional data analysis and relational databases RDBMS has become an issue (Jung, Youn, Bae, & Choi, 2015).

In line with the increased data to be stored and processed, new technologies have been developed that are capable of storing and processing the large amount of data. These Modern technologies can be used to improve the existing open source ERP systems.

1.2 Research Background

Enterprise Resource Planning (ERP) optimizes, automates and integrates most of the business processes and transactions in an organization. Nowadays, many open sources of the ERP systems have been developed and implemented, The Odoo framework, previously known as OpenERP, is the most commonly installed open source ERP system worldwide. Most implemented open source ERP systems such as the Odoo are using Relational Database Management System (RDBMS); therefore, continue daily processes lead to large amount of data stored in the relational database; thus, the database performance decrease. Performance comparison has been conducted by (Jung et al., 2015) between RDBMS and NoSQL databases and it has been found that RDBMS database performance decreased when processing large data while NoSQL database has shown performance improvement when processing large data. Relational database performance decrease has an impact on a concurrent user's sessions, business processing, and report processing. As the report processing time increases, the amount of data increases due to the data retrieving from the relational database, and the more data to be processed the more time needs to generate a report.

The report engine that is used for generating reports in the Odoo has been enhanced over time since the Odoo was developed. With every release of the Odoo, there is an improvement for the exits approach or a new approach has been implemented (Moss, 2017). There developers from the Odoo community contribute to solve the problem of processing big amounts of data to generate reports, they provide projects that connect Odoo framework with standalone open source report engines to cover the lack of Open Report engine such as JasperSoft report server and the Pentaho Interactive Reporting (Kendengis & Santoso, 2018).

1.3 Research Motivation

This research is motivated by the importance of the ERP system for Organizations, and the important role that system plays to keep track of business progress assessment, making decisions and achieving business goals. In the business market, it is important to measure information and the value of the information itself as well as the time of processing this information. The longer it takes for data to be converted into meaningful information, the less value it has for the business, which has a direct impact on making decisions in proper time and achieve business goals.

Report performance latency is caused by the sequential processing for the ever-increasing data stored in the relational databases. Parallel data processing is developed to process large scale data and to handle large loads. There are many open source applications that provide parallel processing; hence, this technique can be used to solve report performance latency in Odoo framework with no additional cost.

1.4 Research Problem Statement

According to the ERP systems comparison study conducted by (Kowanda, Firdaus, Bismark, & Pasaribu, 2015), the OpenERP/Odoo system has several advantages over the other two systems in all the comparison dimensions. While the three systems lacked scalability and handling of large loads, the Open ERP/Odoo has high data storage capacity and high processing time needed which affects the system overall performance (concurrent user sessions, business processing, workflow implementation and report processing). The suggested solution by (Kowanda et al., 2015) is to provide high specification servers for implementation. This can solve the system load performance, but it does not require any data processing manipulation in the system database. Meanwhile the two other systems' problems were solved by the integration with Oracle database, which is enterprise database to handle large loads, thus, overcoming

scalability problem. But it increases implementation costs and the ERP system does not consider open source anymore due to the usage of not free database (Moss, 2017).

With the increasing amount of data, scalability issues and processing big amounts of data is a serious problem (Moniruzzaman & Hossain, 2013). The database that was used in Odoo was the PostgreSQL, which is an open source relational data base. Report performance latency emerges due to the sequential data processing that was applied in the PostgreSQL database to generate report while processing a huge data records using the QWeb Odoo report engine. Some experiments were conducted to compare the performance of the Odoo report between the existed solutions of the (Odoo QWeb, JasperSoft report server, Pentaho Interactive Reporting), and it was found that the report performance decreases as the data records to be processed increases. It was also found that the JasperSoft and Pentaho improve and simplify the report design and reduce the system load. But the SQL Queries execution time still generates report for large data records.

The NoSQL database is designed for large-scale data storage and for parallel data processing, to handle large loads (Moniruzzaman & Hossain, 2013). These features can improve Odoo report processing performance, but this approach has not been implemented yet in the Odoo for report processing.

1.5 Research Objectives

This research aims to improve the performance of reporting process in the Odoo's framework to process large scale data records.

1.5.1 Research Objectives

1. To determine the limitation(s) of the existing report processing approach in the Enterprise Resource Planning (ERP) Odoo framework.

2. To develop a reporting module using parallel data processing approach for generating reports up to million data records with reduced processing time in the Odoo's framework.

3. To evaluate the performance of the proposed reporting module in terms of processing time with the Odoo's existing approaches.

1.5.2 Research Questions

1. What is the limitation(s) of the existing Odoo approach used to generate reports?

2. What are the suitable tools and framework for developing a reporting module using parallel data processing approach to generate large data records report with reduced processing time in Odoo's framework?

3. What is the processing time of the existing approaches and the proposed reporting module for generating report with large data records?

1.6 Research Scope

This research focuses on the report processing performance improvement in the open source ERP system (Odoo) for large amount of data up to one million records. This study focuses on applying parallel data processing for data retrieval from NoSQL database to generate report, to solve Odoo reporting latency while processing large amount of data records from PostgreSQL database to generate the report. A New Odoo reporting module is going to be developed to connect with the NoSQL database, capable of executing the SQL queries on the NoSQL database and applying parallel processing data retrieval to generate reports using suitable API's and tools. Executing the SQL subqueries or inner queries are not covered in this study.

Experiments will be conducted to compare report processing performance between the existing report processing approaches (Odoo QWeb, JasperSoft report server, Pentaho Interactive Reporting) and the developed reporting module to generate the

same report. In these experiments, the reports will run several times with different number of records for each approach. This study is significant only to improve the performance of the large amount of data reports.

1.7 Research Expected Outcomes

New Odoo module is able to connect with the NoSQL database, execute SQL queries on the NoSQL database tables, and retrieve large scale data records in parallel to generate the required report with execution time less than the time consumed by the existing approaches (Odoo QWeb, JasperSoft report server, Pentaho Interactive Reporting) to generate the same report.

The new Odoo module is responsible for setting up connection configurations with the selected NoSQL database, integrates Odoo framework with the selected framework that applies parallel processing approach to generate reports by enabling SQL queries execution on the selected NoSQL database tables using suitable API's tools.

1.8 Research Significant

The ERP systems promise integration of organizational processes and access to integrated data across the entire organizations, given the importance of the ERP reports to keep track of the organization's business process, meeting business goals and setting new goals as well as data retrieval to generate reports in timely manner with direct impact on the organization success.

This research should provide performance improvement for data retrieval in the Odoo ERP reporting system for large data set and provide fast data retrieval time in comparison with the existed approaches.

1.9 Thesis Organization

The rest of the dissertation is organized as follows:

Chapter 2: reviews the relevant literatures of the ERP systems, the Odoo's system framework and architecture. This chapter is divided into three parts. The first part includes ERP system definitions, Odoo system framework and architecture in details, the second part reviews the Odoo Report process, existing approaches and their limitations, the third part reviews the Applications of Parallel Data Processing.

Chapter 3: is dedicated to the research methodology, includes literature review, data collection, preliminary experiments, design, development and evaluation.

Chapter 4: is dedicated to the design of the proposed approach.

Chapter 5: is dedicated to the development and implementation of the proposed approach.

Chapter 6: contains the conducted experiments, performance evaluation of the proposed approach and comparison with the existing approaches.

Chapter 7: contains the findings of our research, limitations of the study, it provides suggestions for future research that can be carried from this one.

CHAPTER 2: LITERATURE REVIEW

This chapter explains the definition of ERP system, comparison between ERP systems in the market and the advantages of using open source ERP systems. It also describes the Odoo's framework architecture and the reporting process. The literature review was carried out to collect the necessary information on the related studies of the existing approaches to report on the Odoo's framework that addresses a specific problem in the related research area. The review also focused on previous studies which showed a real performance improvement of data processing using parallel data processing technique in different fields. Finally, it describes Hadoop's framework as one of the open source systems that apply parallel data processing using MapReduce model.

2.1 Enterprise Resource Planning (ERP) System

(Gripe & Rodello, 2011) described the Enterprise Resource Planning (ERP) as a software that is capable of integrating data from a different company's departments that involve in business processes. This integration involves processing and tracking of data on account journals, account transactions, stock, invoicing, purchased orders and human resources, for example, the ERP is an integration system that organizations used for making decision based on the analysis of data from different incorporated sub-systems. The ERP systems were limited to large organizations because of expensive implementation cost. But after ERP open source system emerged, the ERP systems became widely used regardless of the organization's size (Ganesh, K, C, & A, 2016).

In recent decades ERP systems have showed great importance in the business sector, there are many open sources and enterprise ERP systems has emerged in the

market (Kr.Shukla & Indian, 2013), each has its implementation advantages and disadvantages. The ERP Systems comparison was conducted by (Vartak, Desai, & Kamble, 2014) between the ERP open source systems. A survey was conducted for small and medium enterprises, it found some ERP open source systems compatible with the SMEs. Odoo is one of the most compatible open source ERP systems with high functionalities. Another comparison was done between top 3 open source ERP systems: Open Bravo, Odoo and Adempiere by (Kowanda et al., 2015). At the beginning, a literature review was done to identify the leading open skource ERP systems through previous analysis. The systems were identified as shown in the table below:

Table 2.1: Leading open source ERP systems.

Open ERP (Odoo)	OpenBravo
ERP5	Opentaps
Compiere	Adempiere
WebERP	BlueERP
GNU Enterprise	

After researching on each system, three were selected for analysis in detail, to evaluate the OpenERP/Odoo, Openbravo and Adempiere. After the evaluation, some systems were omitted from the list for one of the following reasons: functional problems, lack of freely database, not mature enough or lack of documentation. Six dimensions were considered for evaluating the systems.

A recent comparison was conducted by (Ganesh et al., 2016) between well-known ERP systems. The SAP and Microsoft Dynamics NAV were selected as they are on top of Commercial ERP systems. The Odoo and OpenBravo were selected from the open source ERP systems. Six factors were considered in the comparison. Different

feedbacks were collected from around 4319 users of different ERP systems. The table below presents the comparison between the selected ERP systems.

Table 2.2: ERP systems comparison (Ganesh et al., 2016).

No	Feature Name	User Feedback in percentage			
		OpenERP (Odoo)	OpenBravo	Microsoft Dynamic NAV	SAP
1	Features & Business Application	84.8%	29.80%	33.52%	78.9%
2	Market Position	47%	38.75%	94.75%	100%
3	Productivity, Agronomy & Ease of Use	82.6%	41.6%	48.8%	74%
4	Customization & Flexibility	85.2%	34.6%	51.8%	69.6%
5	Technical Quality	78.6%	54.1%	56.1%	75.3%
6	Total Cost of Ownership	96.16%	46.83%	42.33%	62.83%

The table below summarizes the evaluation of the three comparisons for the ERP systems; the percentage is the user feedback:

Table 2.3: Comparison between ERP systems (Ganesh et al., 2016; Kowanda et al., 2015).

Features	OpenBravo	OpenERP/Odoo	Adempiere	NAV	SAP
Cost & cost of ownership	<ul style="list-style-type: none"> • Modules free • Customization fee • Maintenance fee • 96.16% 	<ul style="list-style-type: none"> • Modules free • Customization fee reduced. • Maintenance fee • 46.83% 	<ul style="list-style-type: none"> • Fully open source system 	<ul style="list-style-type: none"> • Modules fee • Maintenance fee • 42.33% 	<ul style="list-style-type: none"> • Modules fee • Maintenance fee • 62.83%
Support Availability	<ul style="list-style-type: none"> • OpenBravo forum • Wiki documentation • Reporting bugs and feature requests 	<ul style="list-style-type: none"> • Forum and mailing lists • Wiki documentation and official source repository • Installation manual, developer book and community book available 	<ul style="list-style-type: none"> • Wiki community • Functional, user interface and best practices documentation available 	<ul style="list-style-type: none"> • No available forum • Support fee • Documentation fee 	<ul style="list-style-type: none"> • No available forum • Support fee • Documentation fee
Stability and Maturity	<ul style="list-style-type: none"> • hundred success stories in vendor websites 	<ul style="list-style-type: none"> • Implemented in 45+ countries, 100 documented success stories company's websites 	<ul style="list-style-type: none"> • encourage the contribution from all over the world. • a few success stories of implementation shown on Adempiere website 	<ul style="list-style-type: none"> • Stable and mature 	<ul style="list-style-type: none"> • Stable and mature

Table 2.3: Comparison between ERP systems (Ganesh et al., 2016; Kowanda et al., 2015), continued.

Features	OpenBravo	OpenERP/Odoo	Adempiere	NAV	SAP
Customization and Flexibility	<ul style="list-style-type: none"> • Module customizations • Limits on workflow customization • 85.20% 	<ul style="list-style-type: none"> • Quick and easy module customization including workflows, actions, and reports. • Integration with other applications • 34.60% 	<ul style="list-style-type: none"> • Allow customization without development through Adempiere Application Dictionary 	<ul style="list-style-type: none"> • Not allowed • Customization fee • 51.80% 	<ul style="list-style-type: none"> • Not allowed • Customization fee • 69.60%
Scalability	<ul style="list-style-type: none"> • Able to connect with Oracle database, Suitable for larger deployment 	<ul style="list-style-type: none"> • Required large servers, minimum (1 GB RAM) 	<ul style="list-style-type: none"> • Capability to use Oracle database in addition to PostgreSQL, handle large load 	<ul style="list-style-type: none"> • Scalable using Enterprise database 	<ul style="list-style-type: none"> • Scalable using Enterprise database
User Interface	<ul style="list-style-type: none"> • User friendly interface • Implements all keyboard shortcuts 	<ul style="list-style-type: none"> • Comfortable and well-designed interface. • User friendly: drag and drop, flexible display, configured dynamic dashboards and per-user customizable, data filtering. 	<ul style="list-style-type: none"> • Integrate with JGoodies and Tango project to improve multi-platform support simplified and micro-appropriate design. 	<ul style="list-style-type: none"> • User friendly interface 	<ul style="list-style-type: none"> • User friendly interface

Table 2.3: Comparison between ERP systems (Ganesh et al., 2016; Kowanda et al., 2015), continued.

Features	OpenBravo	OpenERP/Odoo	Adempiere	NAV	SAP
Features and business App	84.80%	29.80%	-	33.52%	78.90%
Productivity and ease of use	82.60%	41.60%	-	48.80%	74%
Market Position	47%	38.75%	-	94.75%	100%
Technical Quality	78.60%	54.10%	-	56.10%	75.30%

From the above comparisons, it seems the three open source systems following a sustainable growth path and can compete the enterprise ERP systems in reducing cost or at least be available to organizations with possible amount of cost. Regarding stability and market position of the vendor, the enterprise systems have the advantage over all three open source systems because all are recent in the market. All three open source systems limitations can be summarized to their scalability as there are still doubts about the ability of those systems to handle big volumes of users or requests have scalability issues to handle large loads; the suggested solution given by (Kowanda et al., 2015) is to provide high specification servers for implementation in Odoo framework. While the two other systems OpenBravo and Adempire, the problem can be solved by integration with Oracle database which is enterprise database that handle large loads, to overcome scalability problem. However, it increases implementation costs, and the ERP system does not consider open source anymore due to the lack of not freely used database (Moss, 2017).

Integrating subsystem-automating processes, automating workflows, dashboards, and reports, are all components provided by the ERP system to improve an organization's business process and meet business goals (Nah et al., 2001). Commercial ERP systems are expensive which make them difficult for small and medium organizations to invest in them. The emergence of the open source based ERP systems with less cost, makes the ERP implementation applicable for all organizational sizes (Ganesh et al., 2016).

(Ganesh et al., 2016; Jindal & Dhindsa, 2013; Vartak et al., 2014) described the main advantages of the ERP open source systems that can solve the commercial ERP system limitations:

1. Less expensive and it reduces cost: open source ERP systems reduce the cost of implementation. Whereas, no license is required to run the system, no maintenance and

support cost, and it uses open source databases and operating systems which are obtained for free. (Vartak et al., 2014) compared between open source ERP systems and compatibility for SME's (small-medium enterprise) and found that open source ERP systems were low cost but, the provided functionality was limited.

2. Flexibility and adaptability: open source ERP systems provide a flexible way of customization to meet the business need for the organization with less effort and complexity.

3. Ownership and vendor independence: organizations have full access to the source code and control the system without vendor intervention. Support and technical knowledge can be obtained from open source communities, which make vendor independence.

4. Quality assurance: because the open source ERP systems rely on contributions from communities, this creates competitive advantage between the expert developers. Hence, code quality can be guaranteed.

5. Easily integration: open source ERP Systems are compatible with multiple technologies which make it easier to integrate with other organization's existing systems.

6. Easily upgradable: upgrading the system can be done without affecting running server due to the separation between system interface and system framework whereas the customization is done at the interface level.

2.1.1 ERP Main Components

1. **Modules:** subsystems or applications each is dedicated to a specific business function, all integrated to accomplish the organization processes. Main Apps such as Accounting, Sales, Purchase, Manufacturing, Project Management and Human Resource Management. Data that generated from different Apps stored in one single database.

2. Workflows: organize a specific task into steps for the user to go through following predefining rules. Thus, eliminates user errors and control transactions quality.

3. Reports: presentation of data generated from ERP system processes converted into a meaningful information.

4. Dashboards: present information in many different dynamic graph types, consider as a key performance indicator to figure out the organization's operations productivity (Nah et al., 2001).

2.2 Odoo ERP System

Odoo framework, previously known as OpenERP has been developed by belgium company in 2005; the name was Tiny ERP, three years later the name has been changed to OpenERP and in 2014 version 8 has been released and the name changed to Odoo.

From Odoo version 9 the Odoo's company begun to release two versions of Odoo, community edition which is totally free and enterprise edition which requires a license; the piece is based on the region and the number of users. Odoo version 10 that has been released in 2016 is not just an ERP system but has gone beyond that to include embedded Business Intelligence tool, E-commerce, and website builder. Thus, make Odoo a Comprehensive system.

As mentioned earlier, many open source ERP systems in the market but what makes Odoo highly recommended and modular solution comparing with other ERP solutions, is each business function is carried out by a dedicated app, which allows growing businesses to start with a few apps and to adopt more as their needs change and expand (Moss, 2017).

2.2.1 Odoo Framework

Odoo framework is compatible with different platform's operating systems and integrated tools. However, Odoo common implementation platform is under Ubuntu operating system (Ganesh et al., 2016). There are many reasons for chosen ubuntu as recommended operating system: ubuntu is open source, is the primary target platform for the Odoo development team and has strong community support for Odoo (Moss, 2017).

Odoo is built using Python programming language and uses the PostgreSQL database for data storage. So, to host Odoo framework python libraries and PostgreSQL database need to be installed first. The Odoo source code can be downloaded from Odoo or GitHub website (Reis, 2016).

2.2.2 Odoo Architecture

Odoo is built upon a Model-View-Controller (MVC) architecture. Since ERP system is complex, so modifying the interface without affecting the data tables and structure is so complicated. This architecture solves the problem of separating the UI from the business logic by adding intermediate component (Ganesh et al., 2016).

MVC pattern makes it easy to provide multiple views of the data, supporting user initiative. The main goal of MVC is to separate the display of the information from the business logic and management of the underlying data. For example, same data can be displayed in different views based on user's role without need to change the model.

2.2.2.1 Models

The model is the data which is stored in the database. Odoo has used PostgreSQL database and the database schema is defined automatically when Odoo modules are

installed. Odoo framework converts the model definitions which is written in python code and creates the corresponding tables structures (columns with data types, constraints, primary and foreign keys) inside of the PostgreSQL database, hence; the data and tables can be restructured without any impact on the user interface. In addition, extending and restructuring data model are applicable in Odoo framework through web interface without need to modify the source code.

2.2.2.2 Views

View is responsible for presenting data and interacting with the user, views are defined in XML (eXtensible Markup Language) documents. The Odoo framework is responsible for recall these view files in a web browser, these views represent the Odoo's user interface and can be modified without affecting the data tables and the data itself.

2.2.2.3 Controllers

The controller is a link between the business logic (model) and the user interface (views). The controller components in Odoo called OpenERP objects are written in Python code in Odoo modules, these objects provide access to the data stored in the database easily through ORM methods (Object-Relational Mapping), which is a middle-ware component layer that facilitates the communication between Odoo and the relational data storage system to perform database Create, Read, Update and Delete (CRUD) operations (Moss, 2017), figure 2.1 illustrates the MVC architecture. Figure 2.2 illustrates the Odoo MVC architecture.

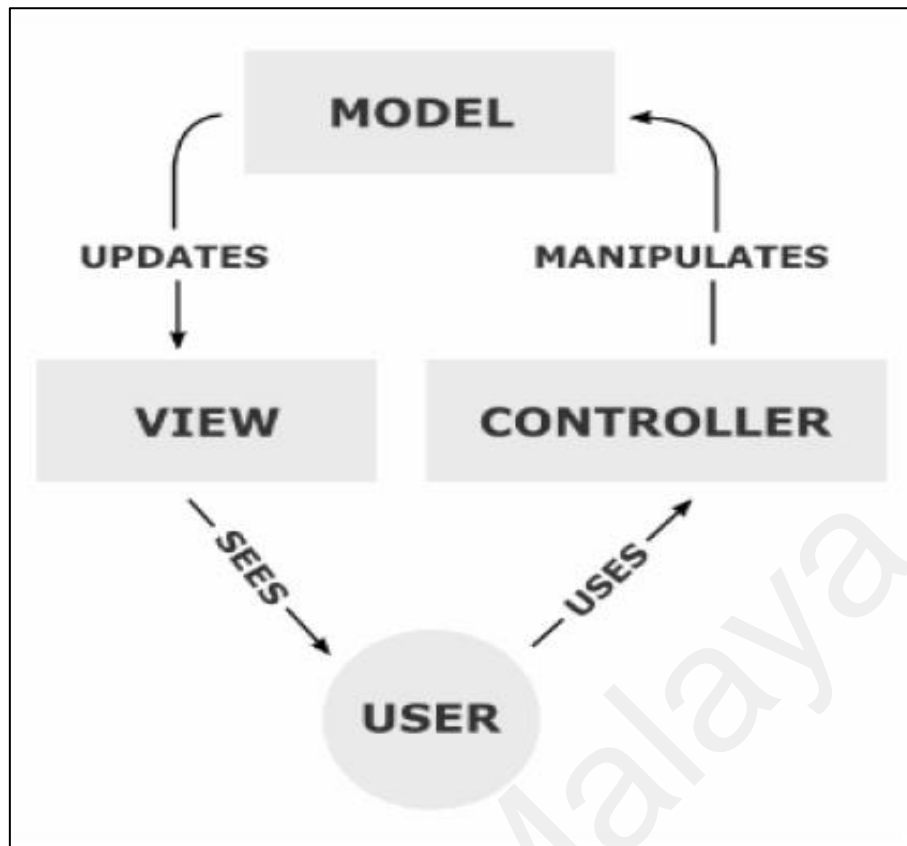


Figure 2.1: MVC architecture (Ganesh et al., 2016).

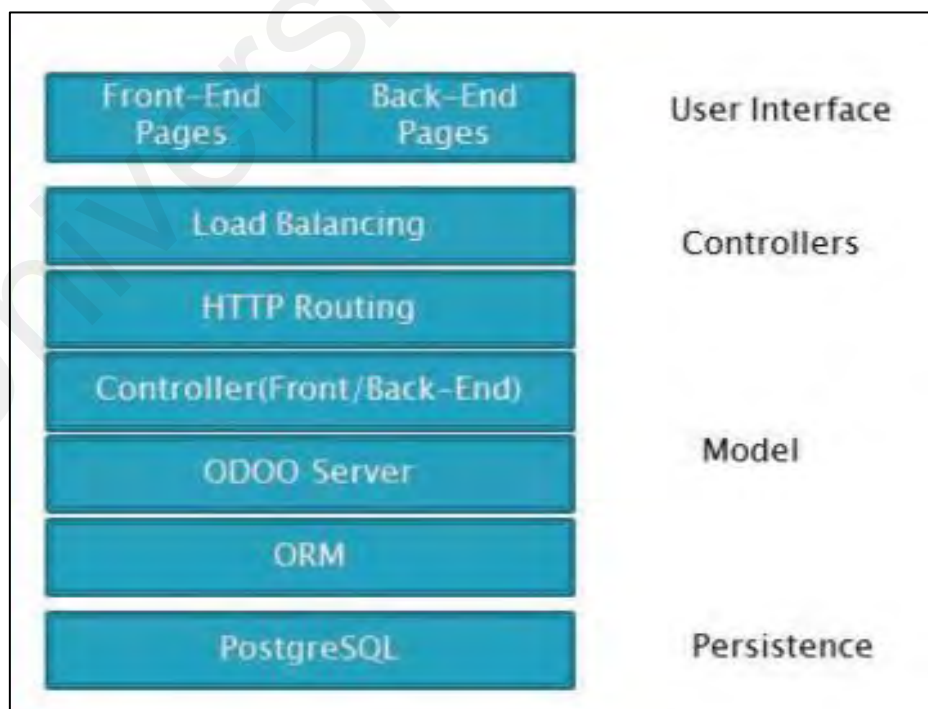


Figure 2.2: Odoo MVC architecture (Vaja & Rahevar, 2016).

2.3 Odoo Report

Reports play an important role in the organization's success, all data generated from automated processes should be converted to meaningful information as regular reports to executive levels whereas helping the organization to keep track of the income and expenses which are important for organization current progress assessment and setting new goals as well. Also, the analytics reports help to figure out business lack and defects then improve strategies to eliminate them (Dezdar & Ainin, 2011).

Finally, Reports document the organization sales, projects, plans and all processes without the need to keep actual paper reports, which provides quick access and therefore leads to make an appropriate decision in a timely manner. ERP Report can be one of the following categories:

- Forms such as invoices, purchase orders, etc.
- Periodic reports that produced periodically: daily, weekly, monthly, and annually.
- Ad hoc reports that are usually one-time reports
- Analytical reports that display what happening in organization operations.

In Odoo and other open source ERP systems normally, there are standard reports with each module which are the commonly used in the module. Also, customization of existing report or creation of new report are available in each reporting engine in open ERP system but need some of the knowledge in both business function side and technical side with the used programming language in each reporting engine (Reis, 2016).

2.3.1 Odoo Reporting process

To process any report and display the report data in Odoo regardless the report engine has been used, the report engine needs to connect to PostgreSQL database and retrieve the required data using SQL queries or ORM methods which convert the python code into SQL query and provide the data needed to generate the report. Finally, the report engine renders the retrieved data into a HTML document and displays the report or converts it to a PDF form if the report display format is PDF. Figure 2.3 shows the reporting process in Odoo framework.

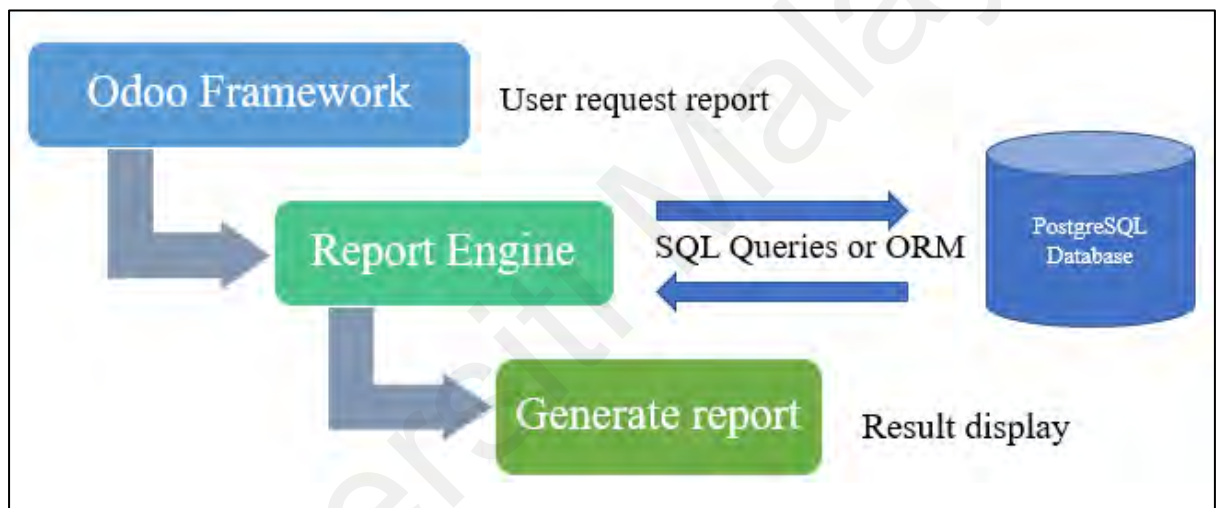


Figure 2.3: Reporting process in Odoo framework (Reis, 2016).

2.3.2 Odoo Reporting Engine

A report engine can turn the system data into meaningful, readable information. Odoo has embedded report engine that used for generating reports, which has been enhanced over time since Odoo has been developed. With every release of Odoo, there is an improvement for the exits approach or a new approach has been implemented to solve the lack and problems that exist (Moss, 2017).

(Reis, 2016) has listed and described all approaches have been developed in Odoo report engine to generate reports in Odoo framework as following:

2.3.2.1 OpenOffice

The first report engine was used for generating reports in version 5 is OpenOffice.org reports, by design the report format in OpenOffice document with SXW extension, this document is only used for developers, to generate the .RML file easily. Then, Open Report engine converts SXW file to RML, fills in desired data and finally converts RML file to PDF or HTML as needed for display.

2.3.2.2 ReportLab

In version 6 the report engine was based on ReportLab library used in Open Report module which is a python script that converts SXW to RML and allows to render the RML document to PDF document. RML also can be used directly to design the report layout, then converts RML to PDF document.

2.3.2.3 OpenERP Webkit Report

Webkit report engine has been developed in version 7 based on Webkit HTML to PDF library (wkhtml2pdf), this approach uses HTML to design report and draw report layout instead of RML then converts it to PDF document.

2.3.2.4 QWeb Template

Final approach has been developed with version 8 is QWeb which is a template engine, XML-based used to generate reports using XML language then rendering report into HTML, report data can be displayed in HTML format or be converted to PDF using Webkit HTML to PDF library (wkhtml2pdf) also. QWeb template uses special XML tags and attributes called directives to generate reports, QWeb parser searches for these directives and replaces them with HTML code. Then, QWeb engine produces the HTML document, QWeb templates report rendered on the server side and uses a python

QWeb implementation. This approach also provides ability to modify existing report without need to change the source code of the report (Moss, 2017). Table 2.4 summarizes all approaches have been developed in Odoo Report engine to generate reports in Odoo framework.

Table 2.4: All Approaches developed in Odoo report engine.

Approach	Version	Solved problem	Limitations	Benefits
OpenOffice (Moss, 2013; Reis, 2016)	V5	Generate report simply.	<ul style="list-style-type: none"> • Bad layout, inaccurate design. • Many limitations compared to full featured report writers. 	<ul style="list-style-type: none"> • For developer with no XML, RML experience.
ReportLab (Moss, 2013; Reis, 2016)	V6	Layout design.	<ul style="list-style-type: none"> • Difficult to generate report from different tables. 	<ul style="list-style-type: none"> • Developer can design good report layout. • Used to generate reports from direct single table. • Used RML to design the report layout.
OpenERP Webkit Report (Moss, 2013; Reis, 2016)	V7	Support Multi header and footer per report or company.	<ul style="list-style-type: none"> • No enhancement on data retrieving level. 	<ul style="list-style-type: none"> • Modify report header and footer from client interface without need to change XML or RML file. • Used HTML to design report.
QWeb (Moss, 2017; Reis, 2016)	V8	Layout design Display report in HTML or PDF format.	<ul style="list-style-type: none"> • No enhancement on data retrieving level. 	<ul style="list-style-type: none"> • Enhance report design through the template framework. • Modify existing report without need to change the source code.

In Odoo community the developers provide projects that connect Odoo framework with standalone open source report engines to cover the lack in Open Report engine. (Moss, 2017) has mentioned two approaches in his book which connect directly with PostgreSQL database server to generate report. The first approach is to connect Odoo with JasperReport Server, then can use iReport or JasperStudio to design report, this approach allows user to design report from complex and different tables easily and generate corresponding SQL queries automatically. But the limitation of this approach is also using SQL Quires to retrieve data and required parameters need to be send to JasperSoft server to see reports. The second approach is to connect Odoo with Pentaho Interactive Reporting (PIR), allows user to design complex report easily but also use SQL to retrieve data from database and Some reporting functionality limit in community edition and available only in enterprise edition.

The two approaches above reduce report processing time by separating report engine server from Odoo server. Thus, processing reports is not affected by number of concurrent user sessions, but the execution time increase while processing SQL queries for large scale data records, due sequential data processing in PostgreSQL database. Although the two approaches have been mentioned by (Moss,2017) as an alternative for Odoo QWeb reporting, but there are no previous studies that have used any of these two approaches in a comparison with Odoo Qweb to generate report in Odoo ERP system. (Kendengis & Santoso, 2018) has mentioned that Pentaho and JasperSoft also have been used to cope Odoo limitation features to process Odoo data and extract a very specific information that help in decision making. Table 2.5 below shows the existing approaches can be used to generate Odoo reports.

Table 2.5: Existing solution to generate reports in Odoo.

Approach	Year	Problem Solved	Limitations	Benefits
QWeb (Moss, 2017; Reis, 2016)	2012	<ul style="list-style-type: none"> • Layout design • Display report in HTML or PDF format. 	<ul style="list-style-type: none"> • Sequential data processing applied. • Takes long time to generate report with 1 million record or more. 	<ul style="list-style-type: none"> • Enhance report design through the template framework. • Modify existing report without need to change the source code.
JasperSoft report server (Kendengis & Santoso, 2018; Moss, 2013, 2017)	2011	<ul style="list-style-type: none"> • allows the user to design report from complex and different tables easily. • Improve system performance. • Capability to extract a specific information that help in decision making. 	<ul style="list-style-type: none"> • using SQL Quires to retrieve data. • Sequential data processing applied. • Takes long time to generate report with 1 million record or more 	<ul style="list-style-type: none"> • Design complex reports using UI (iReport). • reduce system load by separating report engine server from Odoo server.
Pentaho Interactive Reporting (PIR) (Kendengis & Santoso, 2018; Moss, 2013, 2017)	2011	<ul style="list-style-type: none"> • Create report design from complex and different tables. • Improve system performance. • Capability to extract a specific information that help in decision making. 	<ul style="list-style-type: none"> • Using SQL Quires to retrieve data. • Sequential data processing applied. • Takes long time to generate report with 1 million record or more. • Some reporting functionality limit in the community edition and available only in the enterprise edition. 	<ul style="list-style-type: none"> • Capability of integration with different systems • Integrated BI software (OLAP, data mining, ETL and reporting) • reduce system load by separating report engine server from Odoo server.

2.4 NoSQL Database Storage and Parallel Processing

NoSQL refers to not only SQL and it has been emerged as an alternative for the relational database to store, process and manage the huge amount of data which is beyond the capability of traditional relational database storages. NoSQL systems are distributed, non-relational databases designed to handle large-scale data storage and perform parallel data processing across several commodity servers. They also use non-SQL languages to retrieve the data stored. (Moniruzzaman & Hossain, 2013). As described earlier, PostgreSQL is relational database and during Odoo lifetime the data to be stored is increased, hence; shows need to increase the database storage capacity as well as longer time to process data is required. (Jung et al., 2015) in their study assessed the performance between RDBMS and NoSQL databases and has found that RDBMS database (PostgreSQL) performance decreased when processing large data while NoSQL database (MongoDB) has shown performance improvement when processing large data. Another performance comparison has been conducted by (Agarwal & Rajan, 2017) between RDBMS (PostgreSQL) and NoSQL (MongoDB) databases and the result of the comparison experiment has found that PostgreSQL time increases as the size of dataset increases whereas MongoDB performs better by an average factor of 10 which increases exponentially as the data size increases. NoSQL databases has an advantage of the ease of access, speed, and scalability over RDBMS, NoSQL storages provide performance and horizontal scalability (**horizontal scaling means that storage scale by adding additional machines of resources whereas vertical scaling means scale by adding more CPU or RAM to the existing machine**) which made them suitable for systems requiring massive amounts of storage hence, the capability of extending the database easily and low cost (Nayak, Poriya, & Poojary, 2013). NoSQL database can be an alternative storage for PostgreSQL in Odoo framework handle scalability fast data retrieval.

2.4.1 Parallel Data Processing

Processing a massive volume of data simultaneously in a timely manner, the concept based on divide and distribute the large-scale data across several commodity servers and process them in parallel hence, reduce the execution time, improve speed of processing and achieves high performance (Moniruzzaman & Hossain, 2013). Parallel data processing has been emerged after the rapid growth of companies that have to process a huge amount of data such as google, Microsoft and Facebook. Also, the emergence of companies that provide cloud computing systems. Parallel data processing commonly used to perform complex tasks and computations on large scale data set such as data mining, data analysis, data retrieval and resource utilization in cloud computing (Warneke & Kao, 2011).

Many companies have been built data processing frameworks such as Google's MapReduce, Microsoft's Dryad and Yahoo!'s Map-Reduce-Merge. All these frameworks share similar programming models that provide of parallel programming, fault tolerance, and execution optimizations while developers can customize on demand (Warneke & Kao, 2011).

2.4.2 Applications of Parallel Data Processing

(Enaya, 2016) in his thesis has described the Odoo latency problem and he has explained one of the causes of the problem which is the huge data that is generated from the attachment and mail message modules. The two modules are special modules because they can be used from any other modules in Odoo system. During system lifetime, the stored data in these modules is increased and the system performance is decreased consequentially. The performance problem has been solved by using NoSQL data storage to store the data of attachment and mail message modules, then customizing the ORM layer in Odoo framework to enable communicating with the NoSQL data

storage to access and retrieve the required data. After customization has been done, two experiments have been conducted to compare the performance of Odoo system and the customized Odoo. And has been found that, for small and SME companies the Odoo performs better without using NoSQL system. While the customized Odoo, performs faster when the data of attachment and mail message modules are huge.

As mentioned earlier, Odoo scalability issue lead to system performance latency which has an impact on concurrent user sessions, business processing, and report processing. (Enaya, 2016) in his thesis has been solved the Odoo performance latency caused by the huge data generated from attachment and mail message modules, while no enhancement has been done for report processing performance.

There are many deployments of parallel data processing in different domain for the purpose of performance enhancement. (Warneke & Kao, 2009) an efficient parallel data processing has been applied to the cloud computing using Nephele framework. Nephele is the first data processing framework that includes dynamic allocating/deallocating different compute resources during tasks scheduling and execution. Therefore, provides resource utilization improvement, efficiency on parallel data processing and reduce cost.

A survey has been done by (Lee, Lee, Choi, Chung, & Moon, 2011) to provide a studies of data analytics using MapReduce framework through defining the architecture of MapReduce and how parallel data processing works, discussing pros and cons of MapReduce also discuss the improvement strategies to enhance MapReduce framework from the related works. At the end of the survey, it has been found that MapReduce provides good scalability and fault-tolerance for massive data processing, applies flexible parallel processing for different data analysis and automatic optimization for high performance.

(Bu, Howe, Balazinska, & Ernst, 2010) have developed HaLoop; a modified version of the Hadoop MapReduce framework, which is designed for large-scale data mining and data analysis applications. HaLoop is built on top of MapReduce to support iterative applications and improves their efficiency by providing optimization that includes a loop-aware task scheduler and applies several caching mechanisms. HaLoop is a new programming model and iterative program that handles loop control rather than to be manually programmed. It also offers a programming interface to express iterative data analysis applications. The experimental evaluation results demonstrate that HaLoop improves the overall performance of iterative data analysis applications. Table 2.3 below summarizes the application of parallel data processing in different domains.

Table 2.6: Applying parallel processing in different domains.

Researcher	Domain	Used Tool	Problem Solved	Result
(Warneke & Kao, 2009)	Cloud Computing	Nephelae Framework	<ul style="list-style-type: none"> - resource utilization - Data processing cost and time 	<ul style="list-style-type: none"> - Provides dynamic allocate/deallocate resources for task scheduling and execution. - improve the overall resource utilization. - efficient parallel data processing and reduce cost.
(Bu et al., 2010)	Data Mining and Data analysis	HaLoop	<ul style="list-style-type: none"> - performance latency in iterative data mining/ data analysis applications 	<ul style="list-style-type: none"> - Provides loop-aware task Scheduling. - Provides caching mechanisms: caching for loop-invariant, data caching to support fixpoint evaluation. - Improves the performance of iterative data analysis applications.
(Lee et al., 2011)	Data Analytics	Hadoop: MapReduce	<ul style="list-style-type: none"> - Scalability issue - Data processing cost and time 	<ul style="list-style-type: none"> - Fault tolerance - automatic optimization for high performance.
(Enaya, 2016)	ERP System	NoSQL Storage Hadoop: Phoenix Project	<ul style="list-style-type: none"> - Slow system performance caused by a large size of tables 	<ul style="list-style-type: none"> - inefficient problem Solved. - System performance improved.

2.4.3 Hadoop Framework

Hadoop is one of the common big data technologies was created by Doug Cutting in 2005. It is an eventual result of the Nutch search engine project of Dough Cutting. Nutch project has derived from GFS (Google File System) and MapReduce projects created by Google in 2003 – 2004. Cutting joined Yahoo and started a new project and named it like his son's toy elephant. In 2006, Apache Hadoop project was started for the development of HDFS (Hadoop Distributed File System) and Hadoop MapReduce, Now Hadoop is top level project of the Apache software foundation. (Singh, Singh, Garg, & Mishra, 2015)

Hadoop is one of an open source parallel processing framework, provides a distributed file system and a framework for processing and analysis large data set using MapReduce programming model. The main feature of Hadoop is the partitioning of data and computation across many (thousands) of hosts and instead of moving data for computation, parallel computations run on clusters having the data; Hadoop framework scale up by adding commodity server (Shvachko, Kuang, Radia, & Chansler, 2010).

2.4.3.1 HDFS Architecture

HDFS is a distributed file system, which provides unlimited storage, scalable and fast access to data retrieval. Thousands of nodes in a cluster hold petabyte scale of data and if there is a need for more storage, new nodes will be added. It uses a block-structured file system and stores many copies of the files after splitting the file into blocks. Block size and number of replicated copies are configurable. Storing data in a distributed manner provides high fault tolerance and availability during execution of Big Data applications on Compare HDFS with other DFS (Singh et al., 2015). HDFS separates the file system metadata from application data and stores them independently (Ayma et al., 2015) Metadata are the file name, permission, replication and location of

each block of the file (Singh et al., 2015). HDFS stores all the files as replicated blocks and retrieve them by request. By default, it stores three independent copies of each data block (replication) to ensure reliability, availability, performance and failure issues (Ayma et al., 2015).

HDFS provides fast access to data retrieval rather than traditional relational data storage due Master-Slave architecture, there is a single master node called NameNode and multiple slave nodes called DataNodes. Master node responsible to manage all slave nodes, DataNodes store all data and replicated copies of blocks. NameNode is the administrator of file system operations like metadata, file creation, permissions etc. Without NameNode the cluster does not operate, and write/read data cannot be applied, it stores all the metadata of the files in its memory in order to be fast accessed and apply write/read data on request (Singh et al., 2015).

2.4.3.2 MapReduce

MapReduce is a programming model developed recently by Google to facilitate parallel programming and distributed execution on large clusters. It is based on a no single point of failure architecture that is guaranteed by the underlying distributed file system (GFS) which divides the data into smaller segments, stores it and safely replicates it across all nodes. The idea behind MR is to provide abstraction from the underlying hardware and eliminate the complexities of typical parallel programming models such as MPI (Message Passing Interface). This is achieved by introducing two key functions for processing the data: A map function which divides the input data set into smaller segments, apply data processing parallelly on input data set segments and a reduce function which merges the output of all data set segments. Therefore, parallel data processing applied without knowing whether the job was split into 100 segments or 2 segments (Osman, El-Refaey, & Elnaggar, 2013).

MapReduce does not intend to replace relational databases; it's intended is to provide a simple way of programming to process and manage the huge amount of data which is beyond the capability of traditional relational database storages so they can run fast by running in parallel on multiple devices. Hadoop's main aim is to apply parallel computing and data processing on distributed large clusters of commodity machines. MapReduce accomplishes fault tolerance by automatically gathers data from different nodes and combines the result in a single node (Pellakuri & Rao, 2014).

2.4.3.3 Hadoop Ecosystems

Hadoop framework composed of sub-projects, each project dedicated for a specific purpose (storing, analyzing, data processing or maintaining), to meet the research aim which is improve the performance of reporting process in Odoo framework to process large scale data records by apply parallel data processing to retrieve data, the following subprojects are required.

(a) HBase

HBase is an Apache open source project, is a column oriented NoSQL database, runs on top of HDFS with the characteristics of distributed, fault tolerant and high scalable. It is highly recommended for real time read/write random access for large scale data records in databases. Because HBase is column oriented, the data stored in labelled tables, each table has columns and rows stored as a multidimensional sparse, each row has primary key and the data is accessed through this key. HBase allows performing transactional operations: updates, inserts, deletes (Narasimhan & Bhuvaneshwari, 2014), all data accesses through the table primary key and any scan of HBase table converted to a Map/Reduce job, parallel scan in terms of Map/Reduce job results into faster response time and better overall throughput. Row updates are atomic, reading and

writing operations are accomplished at the same time thus, consistency not guaranteed. Recent versions provide blocking rows while read/write operation are run (Vora, 2011).

HBase also supports Master/Slave architecture and uses ZooKeeper for cluster management and coordination. ZooKeeper responsible for maintaining configuration information, naming, providing distributed synchronization, and providing group services. ZooKeeper plays a role of coordinator for HBase clusters.

A performance evaluation experiment has been done by (Vora, 2011) between HBase and SQL database MySQL. The data have been used is image files and stored in HDFS while the meta data are stored in HBase and MySQL. Results have been found that HBase capable to read and retrieve information faster than MySQL and able to serve more clients at the same time, and for write data also HBase is performing better than MySQL but not as much as in read operation. As conclusion, HBase is highly recommended for write-once read-many applications.

(b) Sqoop

Sqoop is a software tool used for data exchange between Hadoop and relational databases. Sqoop is import data from the external relational databases into NoSQL Database system storage (HDFS, HBASE or HIVE). It also enables export data from NoSQL Database system storage into relational databases. Sqoop provides parallel data transferring by using simple SQL query and saves jobs to be run many times for the data update purpose (Narasimhan & Bhuvaneshwari, 2014)

(c) Phoenix

Phoenix is an Apache open source project, comprise a relational layer over the HBase database, it provides an SQL interface to access HBase data. Phoenix executes

the SQL query by splitting the query into several HBase scan processes. Then, it applies these processes in parallel across the HBase database. Phoenix maps each HBase table with Phoenix table and stores the table meta data for fast access (Enaya, 2016).

2.4.4 Report Processing in Hadoop

Storing data in NoSQL database, provides the capability to apply parallel processing and distributed execution using map reduce program to retrieve the data. Map reduce program can be used directly by writing map reduce job using multiple programming languages like Java, Python, Ruby ,C++ (Ayma et al., 2015), Perl and C (Pellakuri & Rao, 2014). The other option is to use Apache projects that implement MapReduce job implicitly such as (HBase, phoenix and pig).

2.5 Summary of literature

Odoo framework is the most commonly installed open source ERP system worldwide, many previous studies have been conducted to compare Odoo with other open source and Enterprise ERP systems, Odoo outperformed other systems. However, during Odoo system lifetime, the data amount increases resulting in scalability issue and report processing latency due sequential data processing in the scalable relational database (PostgreSQL). Many contributions from Odoo community developers to cover the lack in Open Report engine such as Jasper soft and Pentaho that have improved the system performance, but report processing latency exists while processing large data records.

Performance improvement of data processing has been approved using parallel data processing technique from many studies in different fields, hence can be applied in Odoo reporting to generates reports. Hadoop framework is one of the open source systems that apply parallel data processing using MapReduce model. Other Hadoop

components provide data sorting, analysing, and maintaining. All components can work together or separately.

Universiti Malaya

CHAPTER 3: RESEARCH METHODOLOGY

This chapter briefly describes the methodology employed to achieve the research objectives. The aim of this research is to improve the performance of reporting process in Odoo framework and to identify a solution that can reduce the processing time of large scale data records. In order to meet the research aim, the researcher adopted a methodology that comprises of several major steps, which are literature review, data collection, data analysis, development and implementation, evaluation, and testing. Figure 3.1 illustrates the process flow of the research methodology.

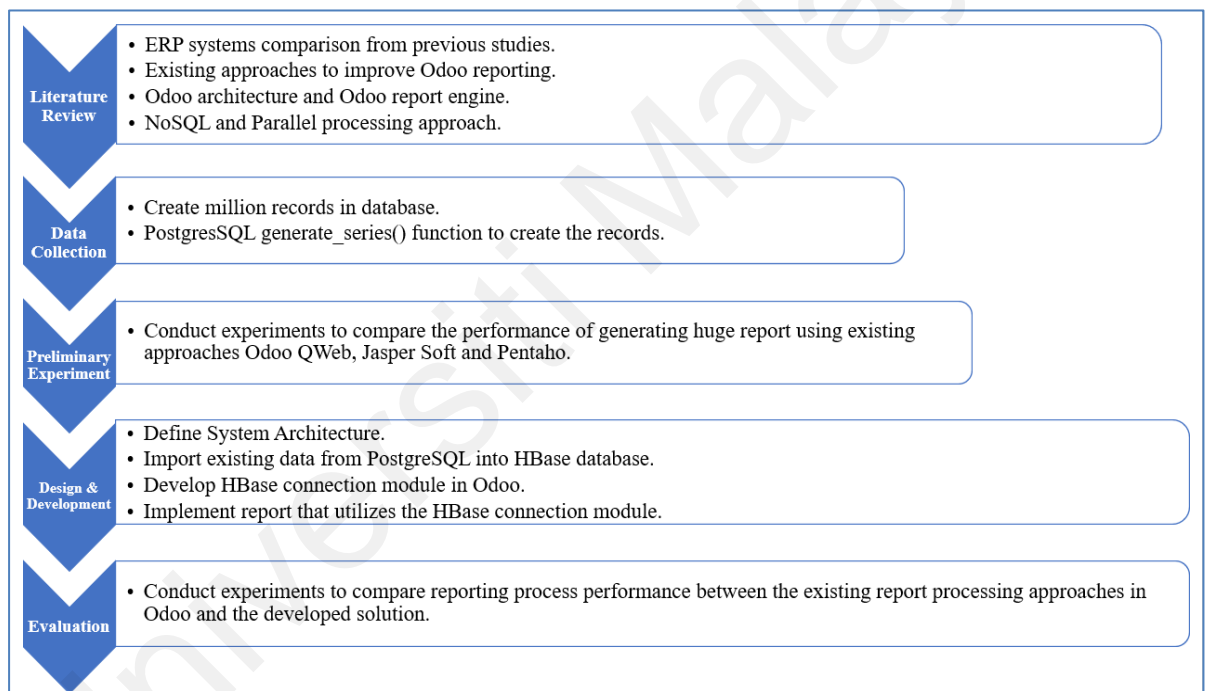


Figure 3.1: Research methodology process flow.

3.1 Literature Review

In this research, the ERP systems comparison with the previous studies was conducted to compare between the open source and the Enterprise ERP systems. Based on the analysis of those studies, the Odoo's framework was selected.

The literature review of related research that used the Odoo's reporting process was to find out the existed approaches that were used to recover the lack of Odoo reporting process, and to investigate the validity of those approaches to improve the Odoo's report processing performance. The literature review also comprised of the existing approaches from Odoo community developers, and they provide projects that connect Odoo framework with standalone open source report engines to cover the lack in Odoo report engine. One approach is to connect the Odoo with Jasper Report Server. This approach allows users to design report from different tables easily to generate corresponding SQL queries automatically. The second approach is to connect the Odoo with Pentaho Interactive Reporting (PIR). This approach allows users to design complex report easily. The two approaches reduce report processing time by separating report engine server from Odoo server. Thus, processing reports are not affected by the number of concurrent user sessions. But the limitation of those two approaches is using the SQL queries to retrieve data, which increases the execution time while processing SQL queries for large scale data records, due to sequential data processing in the PostgreSQL database.

Performance improvement for Odoo reporting can be achieved by applying parallel data processing, instead of sequential processing which applied by the three existing approaches. Meanwhile, the previous studies showed the successful deployment of parallel data processing in various domains. The concept is based on the division of large-scale data into small parts and processes them in parallel. Thus, reducing the execution time, improving speed of processing, and to achieve high performance (Moniruzzaman & Hossain, 2013).

3.2 Data Collection

Dataset in Odoo framework is generated from daily transactions and different processes in a company that uses Odoo as its ERP system. So, it is quite difficult to get existing data for Odoo. The dataset that was used in this work is the data of sale orders from Sales module which contains order details such as order name, order date, customer_id, untaxed amount, taxes, total and order status. The data was collected purposely for this work using the PostgreSQL function called `generate_series()` which generates a set of data based on a specific parameter. First, random data records were created from the Sale Orders through Odoo's framework UI; all the order details were inserted as in figure 3.2 below.

The screenshot shows the Odoo Sales Order form for SO006. The left sidebar contains navigation links: Dashboard, Sales, Customers, Quotations, Sales Orders (selected), Products, Invoicing, Reports, Configuration, and Sales Teams. The main content area shows the order details for SO006, including the customer 'Think Big Systems' and a table of order lines. The summary table at the bottom right shows the following values:

Field	Value
Untaxed Amount	\$750.00
Taxes	\$0.00
Total	\$750.00

Figure 3.2: Sales order form.

Then the PostgreSQL `generate_series()` function were used with SQL INSERT statement that created one million records randomly from the existing records of sale order, PgAdmin tool was used to access the PostgreSQL and to run `generate_series` function, the function format as shown in figure 3.3 below.

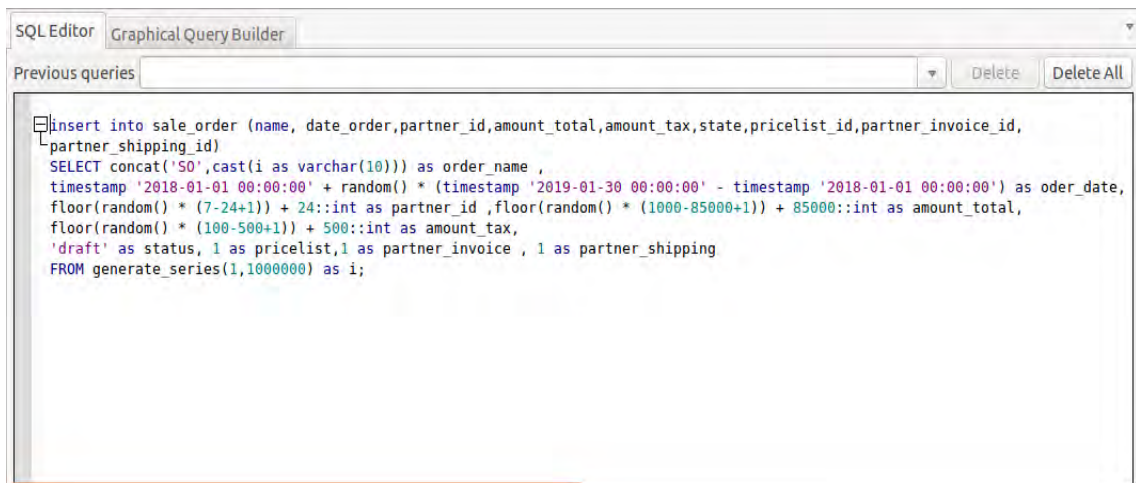


Figure 3.3: generate_series() function in INSERT statement.

the record size average is 85 bytes. Table 3.1 shows sample records from sale order table.

Table 3.1: Sale order table.

ID	Order Name	Order Date	Customer Name	Untaxed Amount	Taxes	Total	Status	row_size
integer	character varying	timestamp without time zone	character varying	numeric	numeric	numeric	character varying	bigint
1005	S021	2018-11-10 00:00:00	The Jackson Group	5132	406	4726	draft	85
1006	S022	2018-11-10 00:00:00	Arthur Gomez	32280	301	31979	draft	85
1007	S023	2018-11-10 00:00:00	Edward Foster	29316	476	28840	draft	85
1008	S024	2018-11-10 00:00:00	Camptocamp	76963	246	76717	draft	85
1009	S025	2018-11-10 00:00:00	Joseph Walters	46812	256	46556	draft	85
1010	S026	2018-11-10 00:00:00	Edward Foster	73251	385	72866	draft	85
1011	S027	2018-11-10 00:00:00	Joseph Walters	52284	142	52142	draft	85
1012	S028	2018-11-10 00:00:00	Think Big Systems	54836	266	54570	draft	85
1013	S029	2018-11-10 00:00:00	Joseph Walters	54197	223	53974	draft	85
1014	S030	2018-11-10 00:00:00	Arthur Gomez	4475	456	4019	draft	85
1015	S031	2018-11-10 00:00:00	Peter Mitchell	20489	475	20014	draft	85
1016	S032	2018-11-10 00:00:00	James Miller	80344	225	80119	draft	85
1017	S033	2018-11-10 00:00:00	China Export	53978	359	53619	draft	85
1018	S034	2018-11-10 00:00:00	Agrolait	77756	321	77435	draft	85
1019	S035	2018-11-10 00:00:00	Tang Tsui	35600	140	35460	draft	85
1020	S036	2018-11-10 00:00:00	Peter Mitchell	5773	483	5290	draft	85

The customer_id field is a foreign key many to one relation refers to the res_partner table in the Base module that contains the customers' details, the customers demo data that created when the Base module installed, were used in this work as shown in table below.

Table 3.2: Customer table.

	id integer	name character varying
1	7	ASUSTeK
2	8	Agrolait
3	9	China Export
4	10	Delta PC
5	11	The Jackson Group
6	12	Camptocamp
7	13	Think Big Systems
8	14	Tang Tsui
9	15	Joseph Walters
10	16	Richard Ellis
11	17	James Miller
12	18	Edward Foster
13	19	Arthur Gomez
14	20	Julia Rivero
15	21	Peter Mitchell
16	22	Thomas Passot
17	23	Michel Fletcher
18	24	Chao Wang
19	25	David Simpson
20	26	John M. Brown
21	27	Charlie Bernard
22	28	Jessica Dupont
23	29	Phillipp Miller

The `sale_order_ids` field in the sale order table is a foreign key one to many relation refers to `sale_order_line` table. It contains sale order products details such as `product_id`, `description`, `ordered_quantity`, `unit_price`, `price_subtotal`, `price_tax`, `price_total`. The table below shows sample of `sale_order_line` table and the record size in bytes.

Table 3.3: Sale order line table.

	ID Integer	description text	product_id integer	order_id integer	price_unit numeric	product_uom_qty numeric	price_subtotal numeric	price_tax numeric	price_total numeric	row_size bigint
1	1	Laptop E5023	30	1	2950.00	3.000	8850.0	0.0	8850.0	196
2	2	Pen drive, 16GB	5	1	145.00	5.000	725.0	0.0	725.0	200
3	3	Headset USB	4	1	65.00	2.000	130.0	0.0	130.0	196
4	4	Service on demand	7	2	75.00	24.000	1800.0	0.0	1800.0	200
5	5	On Site Assistance	8	2	38.25	30.000	1147.5	0.0	1147.5	216
6	6	On Site Monitoring	7	3	30.75	10.000	307.5	0.0	307.5	216
7	7	Toner Cartridge	4	3	70.00	1.000	70.0	0.0	70.0	200
8	8	Service on demand	7	4	75.00	16.000	1200.0	0.0	1200.0	208
9	9	Webcam	5	4	45.00	10.000	450.0	0.0	450.0	196
10	10	Multimedia Speakers	33	4	150.00	3.000	450.0	0.0	450.0	212
11	11	Switch, 24 ports	4	4	70.00	2.000	140.0	0.0	140.0	208
12	12	External Hard disk	5	5	405.00	1.000	405.0	0.0	405.0	200
13	13	PC Assamble + 2GB RAM	10	6	750.00	1.000	750.0	0.0	750.0	212
14	14	Laptop E5023	30	7	2950.00	5.000	14750.0	0.0	14750.0	212
15	15	GrapWorks Software	34	7	173.00	1.000	173.0	0.0	173.0	208
16	16	Datacard	5	7	40.00	1.000	40.0	0.0	40.0	200
17	17	USB Adapter	4	7	18.00	1.000	18.0	0.0	18.0	204
18	18	Laptop Customized	31	8	3645.00	2.000	7290.0	0.0	7290.0	208
19	19	Mouse, Wireless	23	8	12.50	2.000	25.0	0.0	25.0	220
20	20	[CONS_DEL01] Server	34	9	60000.00	1.000	60000.0	9000.0	69000.0	240

The product_id field in the sale_order_line table is a many to one relation refers to product_template table in the Sales module which includes all the company's products, products demo data were used as shown in table 3.4 below.

Table 3.4: Product table.

	id integer	default_code character varying	name character varying	list_price numeric
1	1	SERV_ORDER	Prepaid Consulting	90.00
2	2	SERV_DEL	Cost-plus Contract	180.00
3	3	SERV_COST	External Audit	180.00
4	4	PROD_DEL	Switch, 24 ports	70.00
5	5	PROD_DEL02	Datacard	40.00
6	6	PROD_ORDER	Zed+ Antivirus	280.00
7	7		GAP Analysis Service	30.75
8	8		Support Services	38.25
9	9	PCSC234	Computer SC234	450.00
10	10		iPad Retina Display	750.00
11	11	E-COM05	Bose Mini Bluetooth Speaker	247.00
12	12	E-COM06	Custom Computer (kit)	147.00
13	13	E-COM07	iPad Mini	320.00
14	14	E-COM08	Apple In-Ear Headphones	79.00
15	15	E-COM09	iMac	1799.00
16	16	E-COM10	Apple Wireless Keyboard	47.00
17	17	E-COM11	Mouse, Optical	14.00
18	18		iPod	16.50
19	19	M-Wir	Mouse, Wireless	12.50
20	20	RAM-SR5	RAM SR5	85.00
21	21	C-Case	Computer Case	25.00
22	22	HDD-SH1	HDD SH-1	975.00
23	23	MBi9	Motherboard I9P57	1950.00
24	24	CPUi5	Processor Core i5 2.70 Ghz	2100.00
25	25	CARD	Graphics Card	885.00
26	26	LAP-E5	Laptop E5023	2950.00
27	27	LAP-CUS	Laptop Customized	3645.00

3.3 Preliminary Experiment

Preliminary experiment has been conducted to compare the performance of generating report using the existed approaches because there are no specific studies used any of these three approaches in their applications or research. To conduct the preliminary experiment, the following tools and environments were used:

- Operating System: Ubuntu 14.0.1, 64 bit
- Odoo version 10
- Database: PostgreSQL9.2
- Eclipse Luna IDE
- Jasper Soft Studio 6.6.0
- Pentaho Business Intelligence 3.7.0
- postgresql-9.2-1002.jdbc4.jar

The hardware was used as the following:

- RAM 4 GB
- Hard disk 1 TB
- Processor Intel Core i5 7500 (7th Gen), Quad-Core, 3.8 GHz.

Jasper soft studio installed as plug-in in Eclipse IDE, connected with PostgreSQL by adding PostgreSQL IP and port; after connection established the database was selected then SQL query created to generate report. In Pentaho, report designer UI was used to connect with PostgreSQL by adding PostgreSQL JDBC jar file, PostgreSQL server IP, port, database name, username and Password. Then, SQL query created to retrieve the data from the defined PostgreSQL data source to generate report. To use Odoo QWeb custom report was created to retrieve the date from PostgreSQL and generate the report

using same SQL query was used in Jasper Soft and Pentaho. The following figure is the SQL query that was used to generate reports in QWeb, Jasper and Pentaho.

```
select s.id as "ID", s.name as "Order Name", s.date_order as "Order Date", p.name as "Customer Name",  
s.amount_total as "Untaxed Amount", s.amount_tax as "Taxes", (s.amount_total - s.amount_tax) as "Total",  
s.state as "Status" from sale_order s  
join res_partner p on (s.partner_id= p.id);
```

Figure 3.4: Preliminary Experiment SQL query.

Seven experiments were conducted using different sizes of data records that were created in the database to compare the performance of generating sale orders report using Odoo's QWeb, and the two other existing approaches (Jasper Soft and Pentaho), in each experiment; the number of records determined in the SQL query. The experiments found that the reporting process performance decreased when the number of data records retrieved from databases increased, and more than 500 seconds required in generating the sale orders report with one million data records in the three approaches. Table 3.5 presents the experiments execution time in generating sale orders report in the three approaches.

Table 3.5: Experiments' results for the three existing approaches.

Experiment		Odoo QWeb	Jasper Soft	Pentaho
1	Execution time/second	5.24	0.44	1.50
	No of Records	1000	1000	1000
2	Execution time/second	8.02	1.21	7.32
	No of Records	10000	10000	10000
3	Execution time/second	10.17	3.40	18.62
	No of Records	30000	30000	30000
4	Execution time/second	14.89	8.32	20.05
	No of Records	50000	50000	50000
5	Execution time/second	20.55	23.94	53.20
	No of Records	100000	100000	100000
6	Execution time/second	107.55	239.45	278.04
	No of Records	500000	500000	500000
7	Execution time/second	511.45	503.60	532.30
	No of Records	1000000	1000000	1000000

The experiments show that the execution time of the three existing approaches QWeb, Jasper Soft and Pentaho increased in generating report while the number of records increased as well. The first four experiments show that the execution time to generate report using Jasper soft was less than the execution time to generate the same report using Odoo QWeb and Pentaho. Experiments number five and six Odoo QWeb used less execution time compared with other approaches, and the execution time gap between the three approaches decreased. But in experiment seven, the number of retrieved records was one million; it shows that the execution time of the three approaches increased and took long time to generate report. In conclusion, the experiments proved that the three existing approaches could not cope with huge report.

Figure 3.5 presents the graph of the experiments. The graph shows the execution time of the three approaches in the first five experiments which increased slightly while the

number of records increased as well, from experiment six with 500000 records; there is a noticeable increase in the execution time.

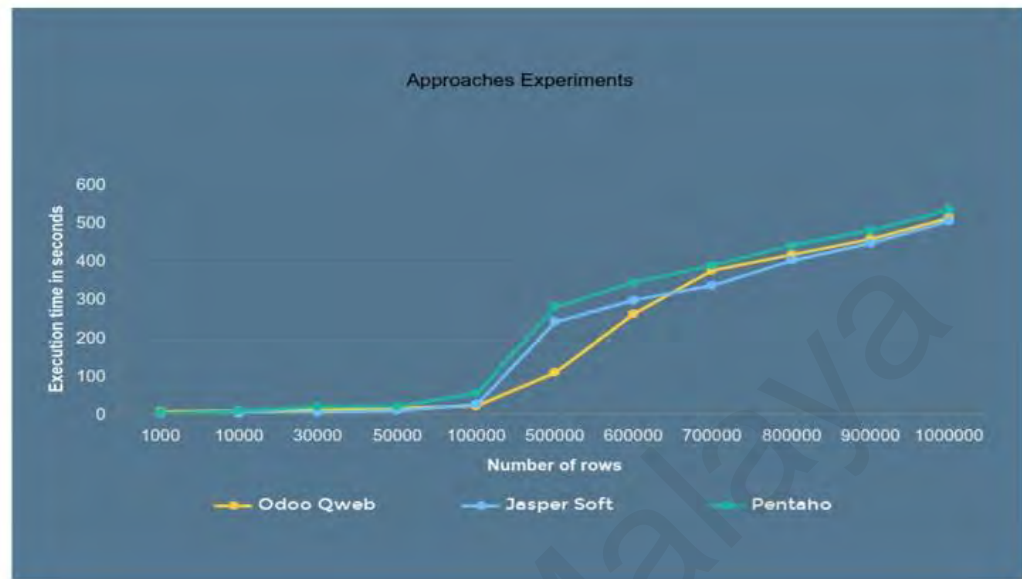


Figure 3.5: Existing approaches experiments results graph.

3.4 Design and Development

3.4.1 Design

In this research, Hadoop framework and sub systems were used to deploy parallel processing using map reduce program to retrieve the data. As mentioned in the literature review, MapReduce program can be used directly either by writing MapReduce job using different programming languages or using apache projects that implement MapReduce job implicitly; Phoenix was selected to apply parallel data retrieval to generate report. HBase database was used to store the data and has been chosen because from the literature the column-oriented database type is similar to the relational database which make it the most relevant type to store relational data conveniently (Zafar, Yafi, Zuhairi, & Dao, 2016). HBase, Hypertable and Cassandra all are open source column-oriented database, (Li & Manoharan, 2013) in their performance comparison between NoSQL databases (MongoDB, RavenDB, CouchDB, Cassandra,

Hypertable and Couchbase) and MS SQL database it has been found that Cassandra is slow in read operation; Hypertable performs moderate in read operation while Couchbase and MongoDB are the fastest for read operation. Another performance evaluation has been conducted by (Jogi & Sinha, 2016) between Cassandra, HBase and MySQL for heavy write operation it has been found that Cassandra provides fast write speeds among other databases, while from the conducted experiment by (Vora, 2011) it has been found that HBase provides high read and write speeds but it highly recommended for write-once read-many applications. So, HBase is the most compatible NoSQL database to integrates with Odoo framework for data retrieval to generate report, further details discussed in chapter four section 4.4 and 4.5.

The design of the proposed solution is to use a single node of Hadoop cluster. Although multiple nodes could be applied, the single node contains Hadoop and sub-projects (HDFS, HBase and Phoenix) are quite enough to develop the proposed solution and conduct the experiments. Odoo's framework is connected to the HBase database through a new Odoo module, using API libraries for data synchronization to maintain the data consistency between the two databases and queries execution. Further details of the design include the system architecture and case diagrams which are discussed in chapter four, section 4.1 and 4.2.

3.4.2 Development

The deployment of a parallel data processing is to expedite the data retrieval from the database to improve the performance of the reporting process in the Odoo's framework which involves three main phases as shown in figure 3.6.

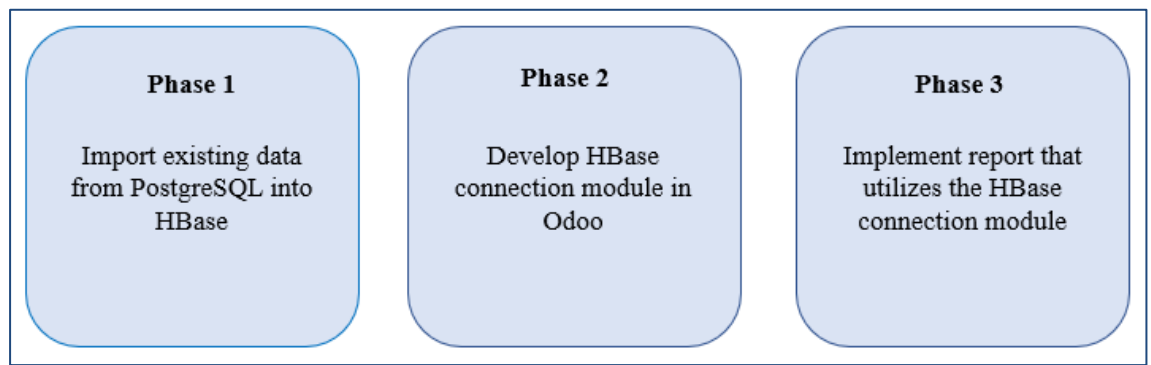


Figure 3.6: Development phases.

The first phase is to import existing data into HBase from PostgreSQL. At this phase, the tables and columns to be imported should be defined, the corresponding tables in HBase are created and then Sqoop tool is used to import data. Sqoop provides Map Reduce job in terms of parallel data transferring results faster than data import. Figure 3.7 illustrates import data process.

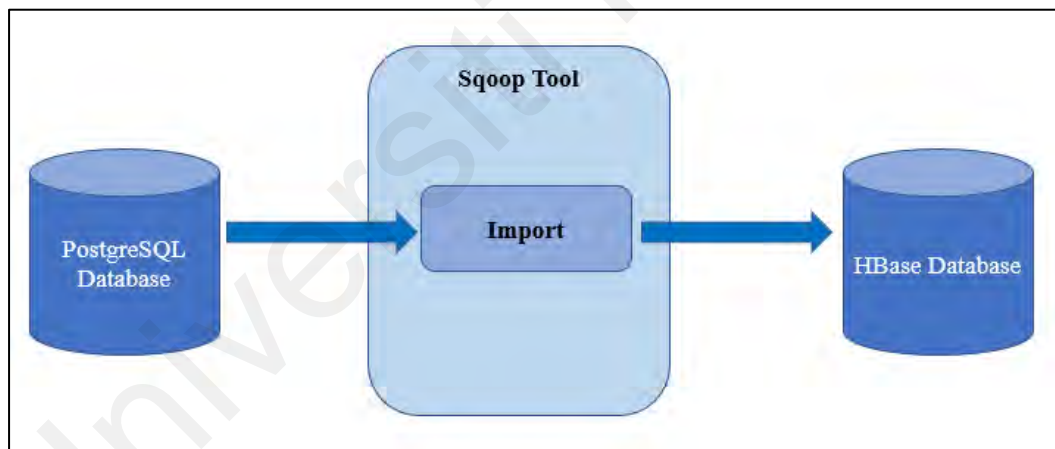


Figure 3.7: Import data into HBase from PostgreSQL.

The second phase is the development of the Odoo module that is responsible of setting up connection string configurations with the HBase through UI (user interface) and enables the APIs for SQL queries execution against the HBase, Odoo module also provides real-time synchronization component that tracks new changes in Odoo and synchronize them up with the HBase. Figure 3.8 shows the HBase connection UI form.

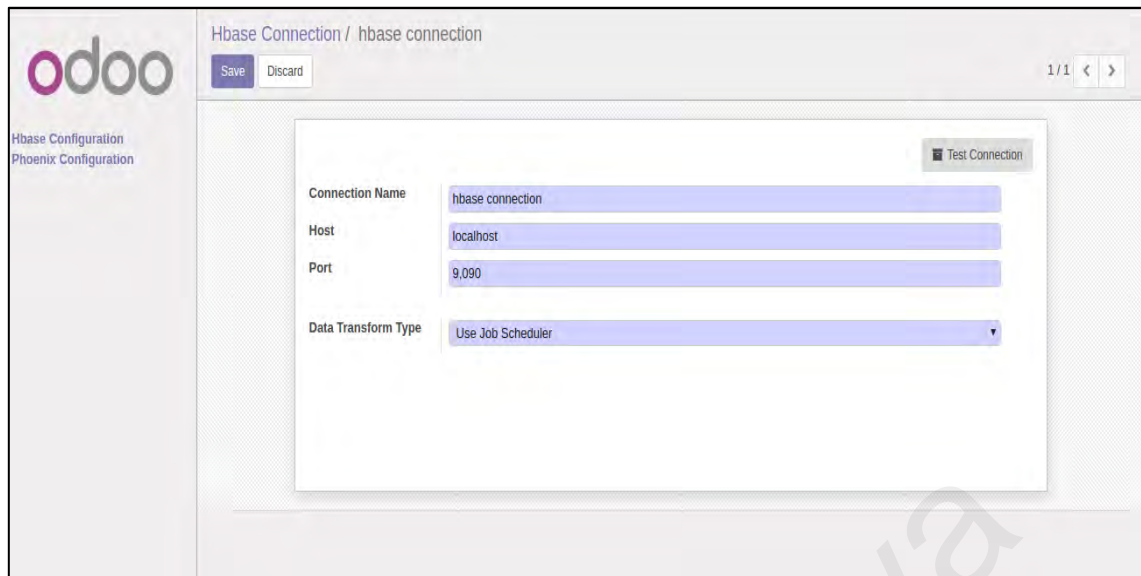


Figure 3.8: HBase connection UI.

The third stage is to implement custom sale orders report in sale module that utilizes the HBase connection module; hence, the data is retrieved from the HBase instead of the PostgreSQL.

3.5 Experiment and Evaluation

3.5.1 Experimental design

The same experiments were conducted to compare reporting processing performance in the existing approaches (QWeb, Jasper Soft and Pentaho) was conducted to the developed solution. Seven experiments were run with different number of data records to retrieve from database, and to compare the performance of the generating sale orders report in the existing approaches and the developed solution. The number of records for each experiment to be retrieved from the same dataset which consists of 1 million record in the database is determined in the SQL query that used to run the experiments to generate reports. Table below shows the number of records for each experiment and the experiment size in Kilobyte.

Table 3.6: Experiments details.

Experiment	No of Records	Size in KB
1	1000	2.34
2	10000	825.84
3	30000	2665.47
4	50000	4505.50
5	100000	9105.47
6	500000	45908.50
7	1000000	91905.46

Another experiment was conducted to compare query execution time between the PostgreSQL database (sequential data processing) and the HBase (parallel data processing) using Phoenix console, experiment table shown below.

Table 3.7: Query execution experiment table.

Experiment		PostgreSQL	HBase
1	Execution time/second		
	No of Records	1000	1000
2	Execution time/second		
	No of Records	10000	10000
3	Execution time/second		
	No of Records	30000	30000
4	Execution time/second		
	No of Records	50000	50000
5	Execution time/second		
	No of Records	100000	100000
6	Execution time/second		
	No of Records	500000	500000
7	Execution time/second		
	No of Records	1000000	1000000

3.5.2 Evaluation method

In this research, the evaluation factors to measure the performance of the developed solution and the existed approaches to generate reports in the Odoo's framework are retrieval time from the database, execution time to generate the report and the number of data record to measure the report processing performance. These factors were used in many studies to evaluate data retrieval performance, (Enaya, 2016) used retrieval time and number of records to measure the performance of the data retrieval time.

3.6 Summary

The main objective of this research is to improve the Odoo's reporting process performance, using parallel processing for data retrieval. The listed methods in this chapter were used to achieve the objectives of this work, the methodology applied in this research has shown that data retrieval time using the developed approach is faster in generating report and therefore, the reporting process performance improved considerably.

CHAPTER 4: DESIGN

This chapter discusses details of the design of the proposed solutions. This includes the system architecture and case diagrams; it also describes the development tools and the environment used to develop the proposed solution. Finally, the chapter explains the API's have selected to connect the Odoo with the HBase and Phoenix.

4.1 System Architecture

This research aims to solve the Odoo reporting latency problem by performing parallel data processing to generate reports, and the corresponding architecture for the proposed solution of the Odoo framework that is connected to the PostgreSQL database, Hadoop framework, HBase NoSQL database, Phoenix and developed Odoo HBase App. The existed data in the PostgreSQL is imported to the HBase using Sqoop tool. The HBase stores meta data in the HBase files while the actual data are stored in the Hadoop Distributed File System. So, should there be configuration based on the system needs and data size to define the number of master/slave nodes in the Hadoop. In this research, single node scheme is applied, and all data is stored in a single node cluster. The CRUD operation on the PostgreSQL is synchronized to the HBase database through the Odoo's HBase App using HappyBase API and running the Thrift server, and the SQL queries is executed by Phoenix through the Odoo's HBase App using Phoenixdb API and running query server to retrieve data and generate reports, figure 4.1 illustrates the system architecture.

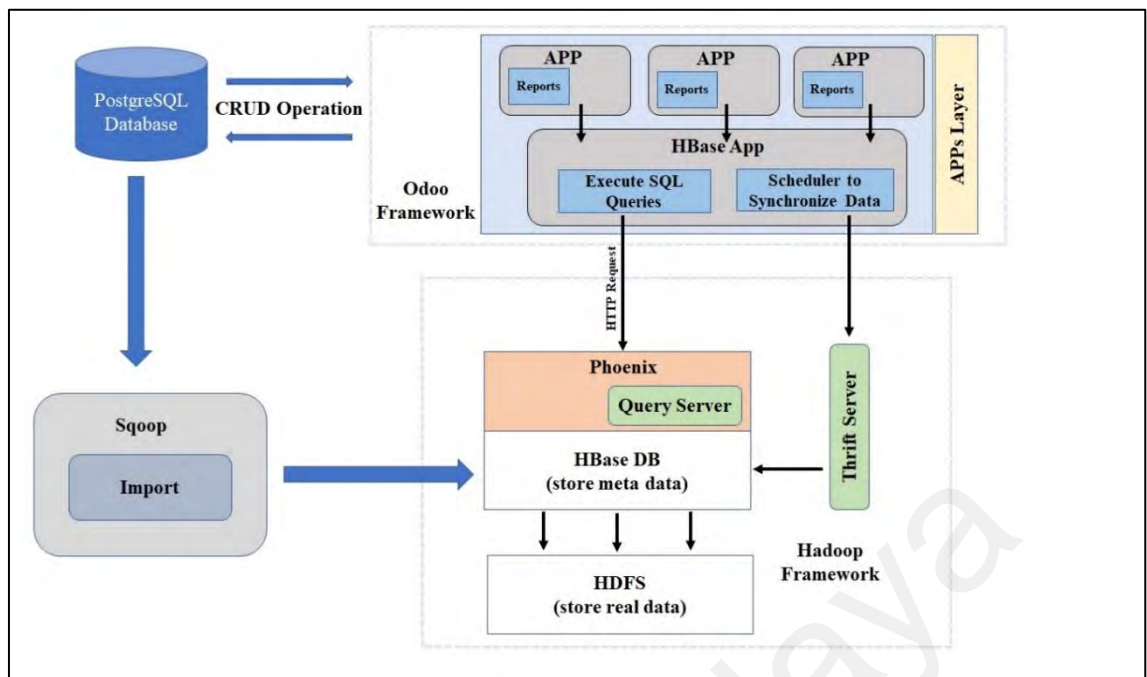


Figure 4.1: System architecture.

4.2 UML Diagram

The following section describes the use case diagram, activity diagram and flow chart for the proposed solution in the Odoo's framework to generate reports.

4.2.1 Use Case Diagram

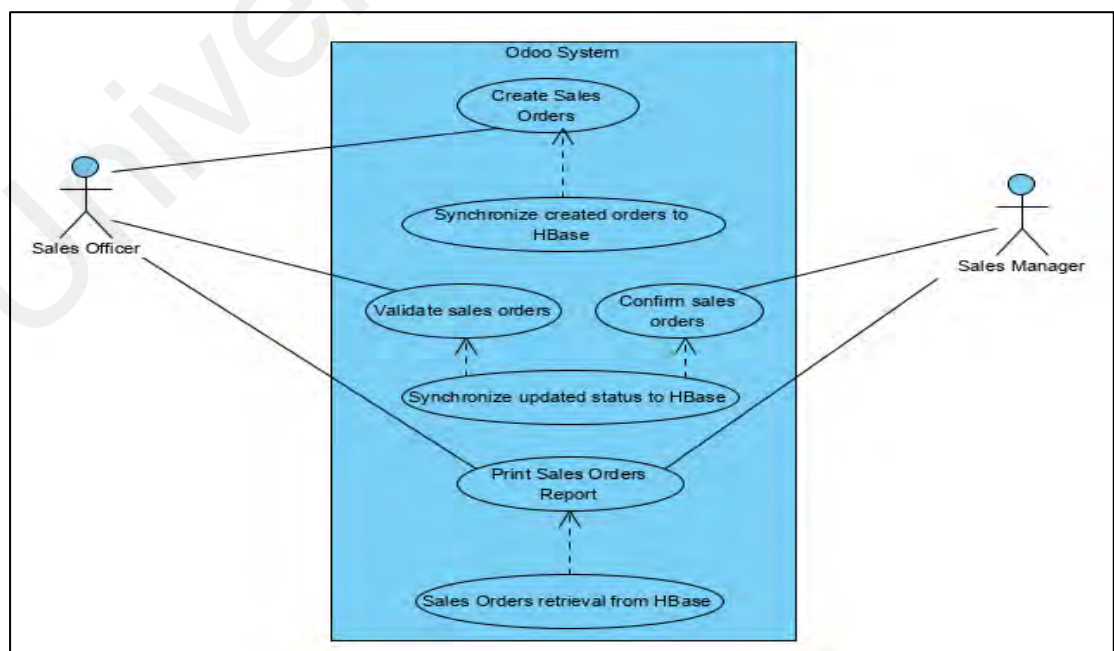


Figure 4.2: Use Case diagram.

Create Sales Order: the actor sales officer creates the sales order by filling up all information related to the order, once the order created in the database is synchronized automatically to the corresponding table (sales order table) in the HBase database.

Confirm Sales Order: the actor sales officer validates the sales order, the status update synchronized automatically to the sales order table in the HBase database.

Validate Sales Order: the actor sales manager confirms the sales order, the status update synchronized automatically to the sales order table in the HBase database.

Print Sales Order Report: the actor sales officer and the actor sales manager can print the sales order report, the sales order data retrieved from the sales order table in the HBase database. Then, the report displayed.

4.2.2 Activity Diagram

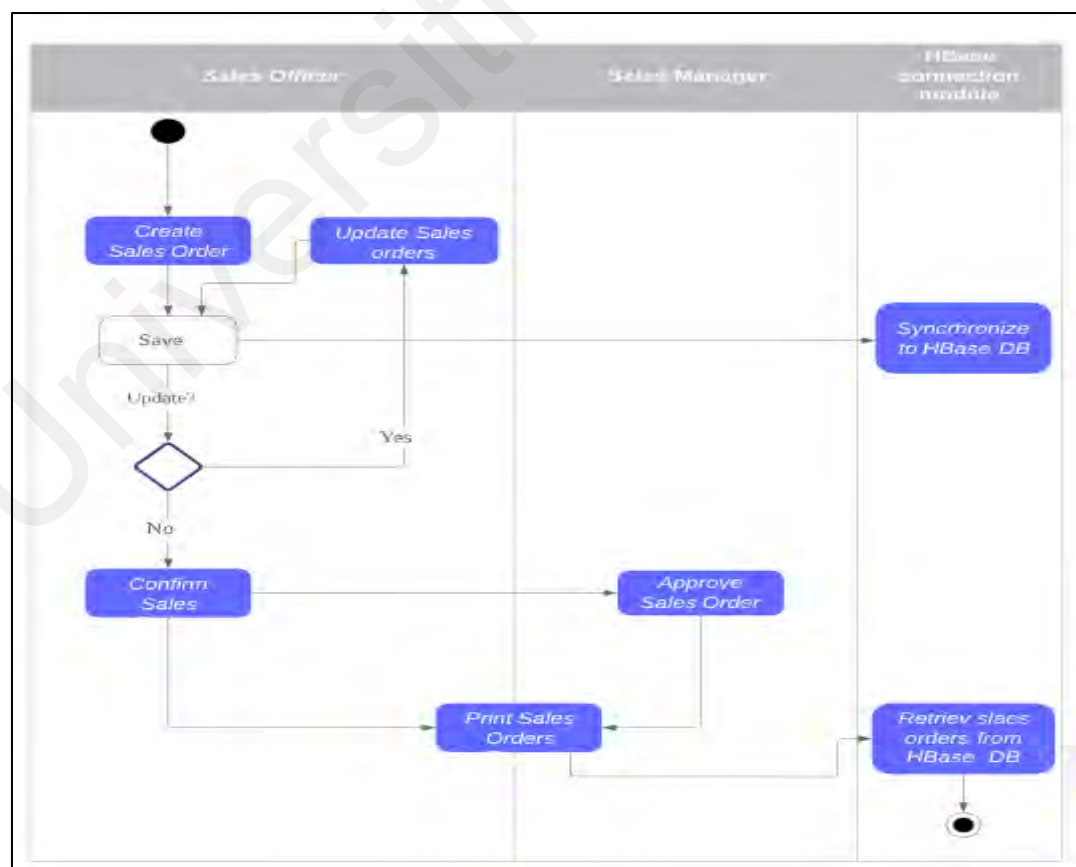


Figure 4.3: Activity diagram.

Activity is started by a Sales officer actor who creates sales orders. When the Sales officer saves the record, the Hbase connection module will synchronize the created sales order to HBase database. If the Sales officer updates the order, the changes will synchronize to the HBase database. The Sales officer actor confirms the order and the Sales manager actor approves it. Sales officers and Sales managers can print sales order reports and the Hbase connection module will retrieve the data from HBase database.

4.2.3 Flowchart Diagram

The following flowchart shows the flow of creating a sales order, update, confirm and approve the created sales order processes. The final process is the print sales order report which retrieve the data from HBase database:

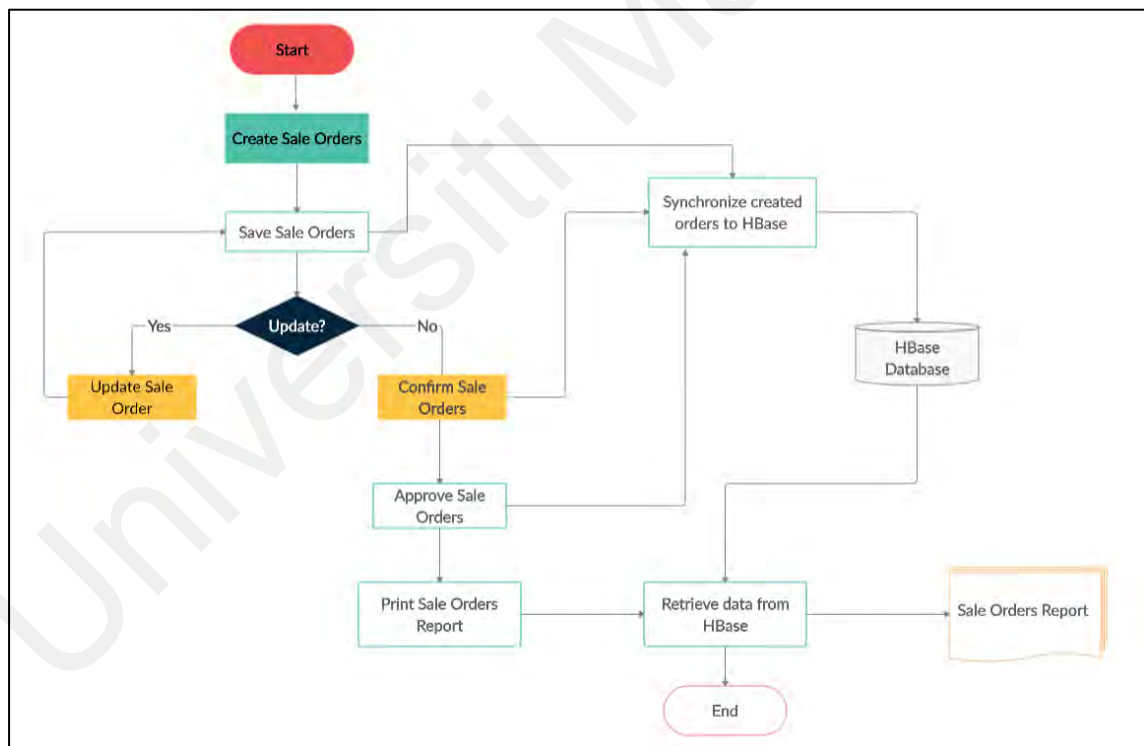


Figure 4.4: Flowchart diagram.

4.3 Development Tools and Environment

In this research, the following tools and environments were used:

- Operating System: Ubuntu 14.0.1.

- Odoo version 10
- Database: PostgreSQL9.2
- Hadoop version 2.7.3.
- HBase version 1.2.6.
- Phoenix version 4.14.0
- Java version 1.8.0 is compatible with Hadoop version 2.7.3
- Import tool: Sqoop.

The selected version of the Hadoop and subcomponent were chosen based on the compatibility with each other to ensure the system running properly without bugs or faults.

4.4 HBase Database Design, Connection and Access

All data in sale order table is of structured data type and has relational columns with other tables. As mentioned earlier in the research methodology chapter; from literature it was found that the column-oriented database is the most relevant type among all NoSQL database types to store relational structured data. The columns are stored on the disk consequentially as Key-Value pairs format; the key is the column, and the value is the data itself. After further studies the HBase was selected, where the actual data is stored in HDFS (Hadoop Distributed File System) while the meta data is stored in the HBase, which has high capability to be read and retrieved information faster than other databases and it is also able to serve more clients at the same time (Vora, 2011). The HBase is highly recommended for write-once read-many applications. So, it is the most compatible database to retrieves data in Odoo framework for report processing.

NoSQL databases and HBase allow data insertion without predefined schema, only the tables should be defined. This can be done by either using “create table” command in the HBase shell or defined corresponding table name, columns and family column (to

group similar columns) while running the data import command for a specific table in the Sqoop tool. Hence, if the table does not exist in the HBase database the table will be created first then the data will be imported. In this research, the dataset used is the data from sales orders table, the corresponding table in the HBase is sales_orders and columns are: ID, order name, order date, beneficiary name, total price and order status. All columns grouped under family column "OrderInfo". HBase stored data in the HDFS consequentially as Key-Value pairs, the key is the row key which is the record ID in PostgreSQL and the value contains the column name prefixed by the family column, the column value and the timestamp for column value versioning purpose. Figure 4.5 shows sample of stored sales orders in the HBase using scan table command in the HBase shell.

```
hbase(main):001:0> scan 'sale_order'
ROW COLUMN+CELL
1 column=SaleOrderInfo:amount_tax, timestamp=1577991274628, value=0.0
1 column=SaleOrderInfo:amount_total, timestamp=1577991274628, value=9705.0
1 column=SaleOrderInfo:amount_untaxed, timestamp=1577991274628, value=9705.0
1 column=SaleOrderInfo:company_id, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:create_date, timestamp=1577991274628, value=2019-12-10 13:41:01.047127
1 column=SaleOrderInfo:create_uid, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:date_order, timestamp=1577991274628, value=2018-11-10 00:00:00.0
1 column=SaleOrderInfo:invoice_status, timestamp=1577991274628, value=no
1 column=SaleOrderInfo:name, timestamp=1577991274628, value=S0001
1 column=SaleOrderInfo:partner_id, timestamp=1577991274628, value=8
1 column=SaleOrderInfo:partner_invoice_id, timestamp=1577991274628, value=8
1 column=SaleOrderInfo:partner_shipping_id, timestamp=1577991274628, value=8
1 column=SaleOrderInfo:pricelist_id, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:state, timestamp=1577991274628, value=draft
1 column=SaleOrderInfo:team_id, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:user_id, timestamp=1577991274628, value=5
1 column=SaleOrderInfo:write_date, timestamp=1577991274628, value=2019-12-10 13:41:01.047127
1 column=SaleOrderInfo:write_uid, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:amount_tax, timestamp=1577991274628, value=120
13 column=SaleOrderInfo:amount_total, timestamp=1577991274628, value=5500
13 column=SaleOrderInfo:date_order, timestamp=1577991274628, value=2018-11-10 00:00:00.0
13 column=SaleOrderInfo:name, timestamp=1577991274628, value=S09
13 column=SaleOrderInfo:partner_id, timestamp=1577991274628, value=13
13 column=SaleOrderInfo:partner_invoice_id, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:partner_shipping_id, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:pricelist_id, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:state, timestamp=1577991274628, value=draft
14 column=SaleOrderInfo:amount_tax, timestamp=1577991274628, value=120
14 column=SaleOrderInfo:amount_total, timestamp=1577991274628, value=5500
```

Figure 4.5: Sales orders stored in HBase.

To connect the Odoo framework with the HBase database through the developed Odoo's HBase App, The HBase provides various Client APIs to access it from other programming languages. The two main supported APIs clients are: REST API and Thrift API, from the evaluation experiment done by (Enaya, 2016) on both APIs, it was found that Thrift API is faster than Representational State Transfer (REST) API, so, the Thrift API was selected to connect between the Odoo and the HBase. To synchronize

data between the PostgreSQL and the HBase, there are two Python libraries that can be used to communicate with the HBase: HappyBase, StarBase. StarBase library does not support Thrift API, it only supports the REST API. So, HappyBase library was chosen for data synchronization through the Thrift API.

4.5 Phoenix Connection using Phoenixdb

To read data from the HBase tables and retrieve a specific data for generating reports, HappyBase API cannot be used for three reasons. First, it retrieves data records for given keys only which might not be known in some report generating cases. The second reason is HappyBase only read data from a single table at time whereas some reports are complicated and need to retrieve data from multiple tables using joins and nested queries. The third reason is HappyBase does not apply parallel processing for data retrieving which is the aim of this research to solve reporting process latency.

As mentioned earlier, Phoenix is a relational layer over the HBase database; it provides an SQL interface to access the HBase data that applies parallel processing to execute queries and retrieves data. Phoenixdb is a Python library to access Phoenix using remote query server; after that access the HBase database to retrieve the required data in parallel.

4.6 Summary

This chapter explains the design of the proposed solutions, and it illustrates the system architecture and the UML diagrams. It also listed the required development tools and environments, the compatibility issues between Hadoop and sub- components were explained. Finally, the selected APIs to connect the Odoo with the HBase and Phoenix are described in detail.

CHAPTER 5: DEVELOPMENT

This chapter discusses detailed development and implementation of the proposed solution to achieve the research aim and objectives. The development process includes environmental setup, data migration, the development of the new Odoo module and implementing methods which was presented in the newly developed module to create report that retrieves data from the HBase database.

5.1 Environmental setup

To develop the proposed solution, Ubuntu14.0.1 operating system should be installed; and Odoo, Hadoop and sub components will be installed as well in the Ubuntu operating system.

5.1.1 Java Installation

Hadoop requires Java to be installed, Java version 1.8.0 is compatible with the Hadoop version 2.7.3, and the following command is to install java JDK: `$ sudo apt-get install default-jdk`, the following command to verify that Java is installed: `$ java -version`. If Java is installed, version details displayed as in figure 5.1 below.

```
root@tutorials:~# java -version
java version "1.7.0_51"
OpenJDK Runtime Environment (IcedTea 2.4.4) (7u51-2.4.4-0ubuntu0.13.10.1)
OpenJDK 64-Bit Server VM (build 24.45-b08, mixed mode)
root@tutorials:~#
```

Figure 5.1: Installed Java version.

5.1.2 Hadoop installation and configuration

To install Hadoop, the source should be downloaded from Apache website; then, unzip the file in /usr/local/Hadoop directory, and to complete the setup of the Hadoop, the following files should be modified:

- ~/.bashrc:

```
#HADOOP VARIABLES START

export JAVA_HOME = /usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL = /usr/local/hadoop
export PATH = $PATH:$HADOOP_INSTALL/bin
export PATH = $PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME = $HADOOP_INSTALL
export HADOOP_COMMON_HOME = $HADOOP_INSTALL
export HADOOP_HDFS_HOME = $HADOOP_INSTALL
export YARN_HOME = $HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR = $HADOOP_INSTALL/lib/native
export HADOOP_OPTS = "-Djava.library.path = $HADOOP_INSTALL/lib"

#HADOOP VARIABLES END
```

- /usr/local/hadoop/etc/hadoop/hadoop-env.sh:

```
export JAVA_HOME = /usr/lib/jvm/java-7-openjdk-amd64
```

- /usr/local/hadoop/etc/hadoop/core-site.xml:

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

- /usr/local/hadoop/etc/hadoop/yarn-site.xml:

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-
services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>
```

- /usr/local/hadoop/etc/hadoop/mapred-site.xml.template:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

- /usr/local/hadoop/etc/hadoop/hdfs-site.xml:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
  </property>
</configuration>
```

- Run HBase command:

```
$ /usr/local/HBase/bin/start-dfs.sh
$ /usr/local/HBase/bin/start-yarn.sh
```

5.1.3 HBase Installation

To install HBase, the following steps were taken:

- Download Hbase source from apache website and unzip file in /usr/local/HBase directory

- Edit ~/.bashrc file:

```
export HBASE_HOME = /usr/local/HBase
export PATH = $PATH:$HBASE_HOME/bin
```

- Edit hbase-site.xml in /usr/local/HBase/conf:

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:8030/hbase</value>
  </property>
  <property>
```

```

        <name>hbase.zookeeper.property.dataDir</name>
        <value>/home/hadoop/zookeeper</value>
    </property>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
</configuration>

```

- Run HBase command:

```
$ /usr/local/HBase/bin/start-hbase.sh
```

Now the HBase was installed and ready to be used.

5.1.4 Phoenix Installation

To install HBase, the following steps were undertaken:

- Download the phoenix-bin.tar from Apache website and expand it.
- Add the phoenix-server.jar to the class path into the HBase lib directory.
- Restart HBase using the following commands:

```

$ /usr/local/HBase/bin/stop-hbase.sh
$ /usr/local/HBase/bin/start-hbase.sh

```

- Add the phoenix-[version]-client.jar to the class path of any Phoenix client.

Type the following command to enter the terminal interface to execute SQL from the command line: `$ sqlline.py localhost.`

5.1.5 Sqoop Installation

- Download sqoop from Apache website and unzip file in /usr/lib/sqoop directory
- Edit ~/.bashrc file:

```

export SQOOP_HOME = /usr/lib/sqoop
export PATH = $PATH:$SQOOP_HOME/bin

```

- Check if the sqoop is installed successfully, and type the command:

```
$ sqoop version
```

5.1.6 Installing Thrift API

```
$ wget sudo dpkg -i automake_1.15-3_all.deb
```

```
$ wget
http://sourceforge.net/projects/boost/files/boost/1.60.0/boost_1
_60_0.tar.g z
$ tar xvf boost_1_60_0.tar.gz
$ cd boost_1_60_0
$ ./bootstrap.sh
$ sudo ./b2 install
```

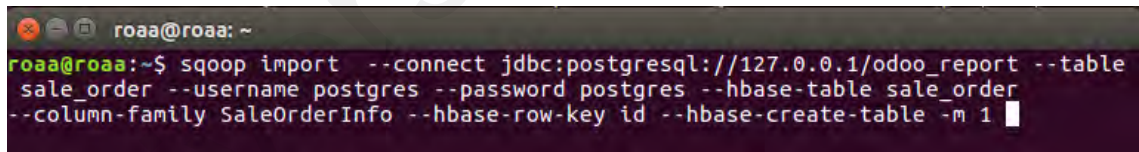
To access the HBase through the thrift, need to open the thrift service on the HBase by running this command: `$ hbase thrift -p 9090 start`

5.1.7 Installing HappyBase API

HappyBase library is chosen to send request to the HBase through the Thrift API, type the following command to install: `$ pip install happybase`

5.2 Data Migration:

Sqoop tool has been used to migrate data from PostgreSQL to HBase. Define the SQL database IP, database name, database username and password, the table name to be imported. The import command is shown in figure 5.2 below.



```
roaa@roaa:~$ sqoop import --connect jdbc:postgresql://127.0.0.1/odoo_report --table
sale_order --username postgres --password postgres --hbase-table sale_order
--column-family SaleOrderInfo --hbase-row-key id --hbase-create-table -m 1
```

Figure 5.2: Sqoop import command.

In previous command, `--hbase-table` clause refers to corresponding HBase table and if the table does not exist in the HBase, create the table first with the defined column family and row key. `-m 1` is the number of reducers to split and import the data. To list the table in HBase, enter HBase shell using command: `$ hbase shell`

`sale_order` table and all relation tables (`sale_order_line`, `product_template` and `res_partner`) were migrated to HBase db, following figure 5.3 displays scan table command for `sale_order` table and the results.


```

hbase(main):001:0> scan 'sale_order'
ROW COLUMN+CELL
1 column=SaleOrderInfo:amount_tax, timestamp=1577991274628, value=0.0
1 column=SaleOrderInfo:amount_total, timestamp=1577991274628, value=9705.0
1 column=SaleOrderInfo:amount_untaxed, timestamp=1577991274628, value=9705.0
1 column=SaleOrderInfo:company_id, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:create_date, timestamp=1577991274628, value=2019-12-10 13:41:01.047127
1 column=SaleOrderInfo:create_uid, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:date_order, timestamp=1577991274628, value=2018-11-10 00:00:00.0
1 column=SaleOrderInfo:invoice_status, timestamp=1577991274628, value=no
1 column=SaleOrderInfo:name, timestamp=1577991274628, value=S0801
1 column=SaleOrderInfo:partner_id, timestamp=1577991274628, value=8
1 column=SaleOrderInfo:partner_invoice_id, timestamp=1577991274628, value=8
1 column=SaleOrderInfo:partner_shipping_id, timestamp=1577991274628, value=8
1 column=SaleOrderInfo:pricelist_id, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:state, timestamp=1577991274628, value=draft
1 column=SaleOrderInfo:team_id, timestamp=1577991274628, value=1
1 column=SaleOrderInfo:user_id, timestamp=1577991274628, value=5
1 column=SaleOrderInfo:write_date, timestamp=1577991274628, value=2019-12-10 13:41:01.047127
1 column=SaleOrderInfo:write_uid, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:amount_tax, timestamp=1577991274628, value=120
13 column=SaleOrderInfo:amount_total, timestamp=1577991274628, value=5500
13 column=SaleOrderInfo:date_order, timestamp=1577991274628, value=2018-11-10 00:00:00.0
13 column=SaleOrderInfo:name, timestamp=1577991274628, value=S09
13 column=SaleOrderInfo:partner_id, timestamp=1577991274628, value=13
13 column=SaleOrderInfo:partner_invoice_id, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:partner_shipping_id, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:pricelist_id, timestamp=1577991274628, value=1
13 column=SaleOrderInfo:state, timestamp=1577991274628, value=draft
14 column=SaleOrderInfo:amount_tax, timestamp=1577991274628, value=120
14 column=SaleOrderInfo:amount_total, timestamp=1577991274628, value=5500

```

Figure 5.3: HBase scan.

5.3 Development of HBase Module in Odoo

As described in the system architecture section the manipulated data (CRUD operations) in Odoo is synchronized to the corresponding HBase table through the HBase module. Also, the SQL query execution to retrieve data from HBase tables for a specific report done through HBase module. The development of HBase module consists of several parts and was created in Odoo with the following specifications:

5.3.1 HBase connection configuration

HBase connection configuration should be defined; therefore, the HBase configuration class is created in python as shown in figure 5.4 below.

```

class HbaseConfig(models.Model):
    _name = "hbase.connection.hbase.config"
    _description = "Hbase Configuration"

    name = fields.Char(string="Connection Name", required=True)
    host_ip = fields.Char(string="Host", required=True)
    port = fields.Integer(string="Port", required=True)
    transform_type = fields.Selection([
        ('direct', 'Direct Transform'),
        ('scheduler', 'Use Job Scheduler'),
    ], string='Data Transform Type', required=True, default='scheduler')

    @api.multi
    def start_connection(self):
        for record in self:
            connection = happybase.Connection(host= record.host_ip, port= record.port, protocol='compact', timeout= 1500000)
            return True

```

Figure 5.4: HBase configuration class.

The corresponding form view is as in figure 5.5

```
<record id="view_hbase_connection_hbase_config_form" model="ir.ui.view">
  <field name="name">hbase.connection.hbase.config.form</field>
  <field name="model">hbase.connection.hbase.config</field>
  <field name="arch" type="xml">
    <form string="Connection Configuration">
      <sheet>
        <div class="oe_button_box" name="button_box">
          <button name="start_connection" string="Test Connection" type="object"
            icon="fa-archive">
          </button>
        </div>
        <group>
          <field name="name"/>
          <field name="host_ip"/>
          <field name="port"/>
        </group>
        <group>
          <field name="transform_type"/>
        </group>
      </sheet>
    </form>
  </field>
</record>
```

Figure 5.5: HBase configuration view XML.

The HBase configuration UI form as in figure 5.6 below.

Figure 5.6: HBase configuration UI form.

5.3.2 Data Synchronization

There is a trade-off between the system performance and the data to be up-to-date in the HBase database. Hence, the system user can define the need and choose between a real-time synchronization which data will be synchronized automatically when data transaction created or scheduled synchronization using Odoo's scheduled actions to

synchronize the manipulated data in a predefined interval (hourly, daily or weekly). These two options of synchronization maintain the consistency between the two databases and can ensure the report generated by the developed module produces correct results.

5.3.2.1 Real-time synchronization

Each model in Odoo has four main functions for the CRUD operations (Create, Read, Write and Unlink), to synchronize CRUD operations into the HBase tables two methods were developed using happybase API to access the HBase tables, one for creating and updating operations, and the second one for deleting operations. Figure 5.7 illustrates the two methods.

```
class OrmFunctions(models.Model):
    _name = "hbase.connection.orm.functions"
    _description = "reflect ORM functions in HBase"

    def start_connection(self):
        hbase_conn = self.env['hbase.connection.hbase.config'].browse()
        connection=happybase.Connection(host= hbase_conn.host_ip,port=hbase_conn.port,autoconnect=False,compat='0.96',transport='buffered')
        connection.open()
        return connection

    def create_write_record(self,context):
        connection= self.start_connection()
        table = connection.table(context.get('table_name'))
        hbase_values={}
        for val in context['values']:
            if context['values'][val]:
                key= context.get('family_column') + ":" + val
                hbase_values.update({key:str(context['values'][val])})
        table.put(context.get('record_id'), hbase_values)
        return True

    def unlink_record(self,context):
        connection= self.start_connection()
        table = connection.table(context.get('table_name'))
        table.delete(context.get('record_id'))
        return True
```

Figure 5.7: Real-time synchronization methods.

5.3.2.2 Scheduled Synchronization

The second method is to log all CRUD operations in log table using audit log module, then create scheduler function to synchronize all transactions logged in audit log into the HBase tables within the pre-defined period. The scheduler functions as shown in figure below.

```
@api.multi
def start_connection(self):
    for record in self:
        connection = happybase.Connection(host= record.host_ip,port= record.port,protocol='compact',timeout= 1500000)
        return True

@api.model
def transaction_scheduler(self):
    log= self.env['auditlog.log']
    orm_func = self.env['hbase.connection.orm.functions']
    for record in self.browse(1):
        if record.transform_type == 'scheduler':
            log_ids = log.search([('transferred', '=', False)])
            if log_ids:
                context={}
                vals={}
                for log in log_ids:
                    if log.method in ('create','write'):
                        for line in log.line_ids:
                            vals.update({line.field_name:line.new_value})
                            context.update({'record_id':str(log.res_id),'table_name':'sale_order', 'family_column':'SaleOrderInfo', 'values': vals})
                            orm_func.create_write_record(context)
                    elif log.method == 'delete':
                        context.update({'record_id':str(log.res_id),'table_name':log.model_id.name})
                        orm_func.unlink_record(context)
                        log.write({'transferred': True})
            return True
```

Figure 5.8: Scheduler function to synchronize transaction.

The interval period for scheduler function to be implemented should be defined in the xml file as shown in figure below.

```
<data noupdate="1">
  <record forcecreate="True" id="ir_cron_transaction_scheduler_action" model="ir.cron">
    <field name="name">Transactions Transfer Job</field>
    <field name="user_id" ref="base.user_root"/>
    <field name="interval_number">1</field>
    <field name="interval_type">work_days</field>
    <field name="numbercall">-1</field>
    <field eval="False" name="doall"/>
    <field eval="'hbase.connection.hbase.config'" name="model"/>
    <field eval="'transaction_scheduler'" name="function"/>
    <field eval="''()" name="args"/>
  </record>
</data>
```

Figure 5.9: Interval period for scheduler function in XML file.

The interval period can be changed manually from scheduled actions UI form in Odoo's system as presented in figure below.

The screenshot shows the Odoo web interface for the 'Scheduled Actions / Transactions Transfer Job' form. The left sidebar contains the Odoo logo and a navigation menu with categories like Dashboard, Users, General Settings, Translations, Technical, and Workflows. The 'Scheduled Actions' menu item is highlighted. The main content area shows the form for the 'Transactions Transfer Job'. At the top, there are buttons for 'Edit', 'Create', and 'Run Manually'. The form fields are as follows:

Name	Transactions Transfer Job	Active	<input checked="" type="checkbox"/>
User	Administrator	Priority	5
Information		Technical Data	
Interval number	1	Interval Unit	Work Days
Next Execution Date	12/05/2018 23:59:00	Number of Calls	-1
Repeat Missed	<input checked="" type="checkbox"/>		

Figure 5.10: Scheduled actions UI form.

5.3.3 Phoenix Connection using PhoenixDB

As mentioned earlier, Phoenix is a relational layer over the HBase database, it provides an SQL interface to access the HBase data. Phoenixdb is a Python library to access Phoenix using remote query server. Hence, accessing the HBase database and executing the SQL query in parallel to retrieve data. The Phoenix connection configuration should be defined (query server URL: IP/Port), therefore Phoenix configuration class is created in python as shown in figure 5.11 below

```
class PhoenixConfig(models.Model):
    _name = "hbase.connection.phoenix.config"
    _description = "Phoenix Configuration"

    name = fields.Char(string="Connection Name", required=True)
    host_url = fields.Char(string="Query Server URL", required=True, default='http://localhost:8765/')

```

Figure 5.11: Phoenix configuration class.

The corresponding form view is as in figure 5.12.

```

<record id="view_hbase_connection_phoenix_config_form" model="ir.ui.view">
  <field name="name">hbase.connection.phoenix.config.form</field>
  <field name="model">hbase.connection.phoenix.config</field>
  <field name="arch" type="xml">
    <form string="Connection Configuration">
      <sheet>
        <group>
          <field name="name" />
          <field name="host_url" />
        </group>
      </sheet>
    </form>
  </field>
</record>

```

Figure 5.12: Phoenix configuration view XML.

The phoenix configuration UI form as in figure 5.13.

The screenshot shows the Odoo Phoenix Configuration UI form. The form is titled "Phoenix Connection / phoenix connection" and has "Save" and "Discard" buttons. It contains two input fields: "Connection Name" with the value "phoenix connection" and "Query Server URL" with the value "http://localhost:8765/". The Odoo logo and "Hbase Configuration" / "Phoenix Configuration" are visible in the top left.

Figure 5.13: Phoenix configuration UI form.

To retrieve data from the HBase table, open the connection based on the given URL, execute the given query and return the result as shown in figure 5.14.

```

def get_connection(self):
    for record in self.browse(1):
        conn = phoenixdb.connect(str(record.host_url), autocommit=True)
        cursor = conn.cursor()
        return cursor

@api.multi
def execute_query(self, query, variables=None):
    cursor = self.get_connection()
    cursor.execute(query, variables)
    result = cursor.fetchall()
    return result

```

Figure 5.14: Phoenix execute query method.

5.4 Implementing Methods in Custom Module

Now the Odoo's framework is integrated with the HBase database, the sale orders data is synchronized and the HBase module manages the access to the HBase tables. The final step is to generate a new report that utilizes the developed HBase module; hence, the data is retrieved from the HBase instead of the PostgreSQL.

5.4.1 Custom Module:

The procedure of developing the new report is the same as using the Odoo's report engine, in identifying report render, and designing report's template in the xml file. But execute_query() function in the HBase module is called to execute requested query on the HBase instead of requesting cr.execute() to execute requested query on the PostgreSQL. Custom sale orders report was developed in the sale module as shown in figure 5.15 below.

```

class SaleCustomSaleOrderReport(models.AbstractModel):
    _name = 'report.sale_custom.report_sale_order'

    def get_data_from_report(self, data):
        res = []
        res.append({'data': [], 'report_option': data['report_option']})
        Sale_order = self.env['sale.order']

        if data['report_option'] == 'sql':
            self.env.cr.execute(''' SELECT s.name AS "Order Name", p.name AS "Customer Name", s.date_order AS "Order Date",
                                   s.amount_total AS "Untaxed Amount", s.amount_tax AS "Taxes", (s.amount_total- s.amount_tax) AS "Total",
                                   s.state AS "Status" FROM sale_order s JOIN res_partner p ON (s.partner_id= p.id) limit %s''',
                                (data['no_of_employees'],))
            orders_dict=self.env.cr.dictfetchall()
            for order in orders_dict:
                res[0]['data'].append({
                    'name': order['Order Name'],
                    'partner': order['Customer Name'],
                    'date': order['Order Date'],
                    'untaxed_amount': order['Untaxed Amount'],
                    'tax': order['Taxes'],
                    'total': order['Total'],
                    'state': order['Status'],})
        else:
            phoenix = self.env['hbase.connection.phoenix.config']
            sql_query = '''SELECT "sale_order"."name" AS "Order Name", "res_partner"."name" AS "Customer Name", "sale_order"."date_order" AS
                               "Order Date", "sale_order"."amount_total" AS "Untaxed Amount", "sale_order"."amount_tax" AS "Taxes",
                               ("sale_order"."amount_total" - "sale_order"."amount_tax") AS "Total", "sale_order"."state" AS "Status"
                               FROM "sale_order" JOIN "res_partner" ON ("sale_order"."partner_id"= "res_partner"."id") LIMIT ? '''
            result = phoenix.execute_query(sql_query,[data['no_of_orders']])
            res[0]['data'] = result

        return res

```

Figure 5.15: Custom sale orders report.

The sale order class was inherited and the ORM methods (create, write and unlink) were modified to cope real-time synchronization for sale_order table as shown.

```

class SaleOrder(models.Model):
    _inherit = 'sale.order'
    @api.model
    def create(self, values):
        context={}
        hbase_conn = self.env['hbase.connection.hbase.config']
        orm_func = self.env['hbase.connection.orm.functions']
        father = super(Employee, self).create(values)
        context.update({'record_id':str(father.id), 'table_name': 'sale_order', 'family_column': 'SaleOrderInfo', 'values': values})
        for record in hbase_conn.browse(1):
            if record.transform_type == 'scheduler':
                orm_func.create_write_record(context)
        return father

    @api.multi
    def write(self, vals):
        context={}
        hbase_conn = self.env['hbase.connection.hbase.config']
        orm_func = self.env['hbase.connection.orm.functions']
        father = super(Employee, self).write(vals)
        context.update({'record_id':str(self.id), 'table_name': 'sale_order', 'family_column': 'SaleOrderInfo', 'values': vals})
        for record in hbase_conn.browse(1):
            if record.transform_type == 'scheduler':
                orm_func.create_write_record(context)
        return father

    @api.multi
    def unlink(self):
        context={}
        hbase_conn = self.env['hbase.connection.hbase.config']
        orm_func = self.env['hbase.connection.orm.functions']
        context.update({'record_id':str(self.id), 'table_name': 'sale_order'})
        resources = self.mapped('resource_id')
        super(Employee, self).unlink()
        for record in hbase_conn.browse(1):
            if record.transform_type == 'scheduler':
                orm_func.unlink_record(context)
        return resources.unlink()

```

Figure 5.16: ORM methods inheritance in sale order class.

5.5 Summary

This chapter explains the development of the proposed solution module configuration, data synchronization and parallel query function execution using phoenixdb API. It also describes in detail the implementation of the developed methods in the custom module.

Universiti Malaya

CHAPTER 6: EXPERIMENTS, EVALUATION AND RESULTS

This chapter illustrates the experiments that were conducted and the evaluation of parallel data retrieval to generate reports in terms of retrieval time. It compares the developed solution with the Odoo QWeb, Jasper Soft and Pentaho. It also discusses the obtained results.

6.1 Experiments and Evaluation

The data was created in the PostgreSQL and migrated to corresponding HBase tables in chapter five, are used to conduct the experiments for the developed solution and the three existing approaches. Each experiment was run with a different number of data records, starting from 1000 records to one million records. As mentioned in the research methodology the number of records for each experiment to is determined in the SQL query that used to run the experiments to generate reports. In this research, the evaluation factors are retrieval time, the number of data record and the query execution on the database, to measure the performance of the developed solution and the existed approaches in data retrieval to generate reports in the Odoo framework.

6.1.1 Retrieval Time:

To compare the retrieval time to generate report for developed solution, QWeb, Jasper Soft and Pentaho; the same SQL query is executed, and the number of data records are equal in each experiment. The following figure is the SQL query that was used to run all the experiments to generate reports.

```
select s.id as "ID", s.name as "Order Name", s.date_order as "Order Date", p.name as "Customer Name",  
s.amount_total as "Untaxed Amount", s.amount_tax as "Taxes", (s.amount_total - s.amount_tax) as "Total",  
s.state as "Status" from sale_order s  
join res_partner p on (s.partner_id= p.id);
```

Figure 6.1: Experiment SQL query.

6.1.2 Number of Records

The number of retrieved data records from database to generate report has influence on report performance, in this research, the data of sales orders from Sale module was used, each record contains order details: order name, order date, customer name, untaxed amount, taxes, total and order status, the record size average is 85-byte, sample of the sale orders is shown in table 6.1 below.

Table 6.1: Sample of sale orders dataset.

ID integer	Order Name character varying	Order Date timestamp without time zone	Customer Name character varying	Untaxed Amount numeric	Taxes numeric	Total numeric	Status character varying	row_size bigint
1005	5021	2018-11-10 00:00:00	The Jackson Group	5132	406	4726	draft	85
1006	5022	2018-11-10 06:00:00	Arthur Gomez	32280	301	31979	draft	85
1007	5023	2018-11-10 00:00:00	Edward Foster	29316	476	28840	draft	85
1008	5024	2018-11-10 00:00:00	Camptocamp	76963	246	76717	draft	85
1009	5025	2018-11-10 00:00:00	Joseph Walters	46012	256	46556	draft	85
1010	5026	2018-11-10 00:00:00	Edward Foster	73251	385	72866	draft	85
1011	5027	2018-11-10 00:00:00	Joseph Walters	52284	142	52142	draft	85
1012	5028	2018-11-10 00:00:00	Think Big Systems	54836	266	54570	draft	85
1013	5029	2018-11-10 00:00:00	Joseph Walters	54197	223	53974	draft	85
1014	5030	2018-11-10 00:00:00	Arthur Gomez	4475	456	4019	draft	85
1015	5031	2018-11-10 00:00:00	Peter Mitchell	20489	475	20014	draft	85
1016	5032	2018-11-10 00:00:00	James Miller	80344	225	80119	draft	85
1017	5033	2018-11-10 00:00:00	China Export	53978	359	53619	draft	85
1018	5034	2018-11-10 00:00:00	Agrolait	77756	321	77435	draft	85
1019	5035	2018-11-10 00:00:00	Tang Tsui	35600	140	35460	draft	85
1020	5036	2018-11-10 00:00:00	Peter Mitchell	5773	483	5290	draft	85

The experiments were conducted seven times with different number of data records, the table below shows number of records for each experiment and the experiment size in Kilobyte.

Table 6.2: Experiments.

Experiment	No of Records	Size in KB
1	1000	2.34
2	10000	825.84
3	30000	2665.47
4	50000	4505.50
5	100000	9105.47
6	500000	45908.50
7	1000000	91905.46

6.1.3 Query Execution Time

To compare query execution time between the PostgreSQL database (sequential data processing) and the HBase (parallel data processing). Direct experiments on both databases were conducted.

6.2 Results

6.2.1 Generating Reports

Odoo reports can be displayed in either PDF or HTML, the HTML file is first created and if the report type is PDF; then, the HTML file is rendered to the PDF file. HTML to PDF process takes long time to be converted using Python Web kit called wkhtml2pdf, whereas the research aim is to improve the Odoo reporting performance using parallel data processing and we do not want the report performance measurement that would affect by another factors. So, all the reports in different approaches were generated in the HTML format.

After running SQL query (figure 6.1) to retrieve data and generate report in the Odoo using the HBase connection module (developed solution), Odoo QWeb, Jasper Soft and Pentaho, the results obtained as follows.

Table 6.3: Approaches experiments results.

Experiment		Odoo QWeb	Jasper Soft	Pentaho	Odoo HBase Module
1	Execution time/second	5.24	0.44	1.50	4.06
	No of Records	1000	1000	1000	1000
2	Execution time/second	8.02	1.21	7.32	4.28
	No of Records	10000	10000	10000	10000
3	Execution time/second	10.17	3.40	18.62	5.21
	No of Records	30000	30000	30000	30000

Table 6.3: Approaches experiments results, continued.

Experiment		Odoo QWeb	Jasper Soft	Pentaho	Odoo HBase Module
4	Execution time/second	14.89	8.32	20.05	9.69
	No of Records	50000	50000	50000	50000
5	Execution time/second	20.55	23.94	53.20	15.02
	No of Records	100000	100000	100000	100000
6	Execution time/second	107.55	239.45	278.04	52.35
	No of Records	500000	500000	500000	500000
7	Execution time/second	511.45	503.60	532.30	104.7
	No of Records	1000000	1000000	1000000	1000000

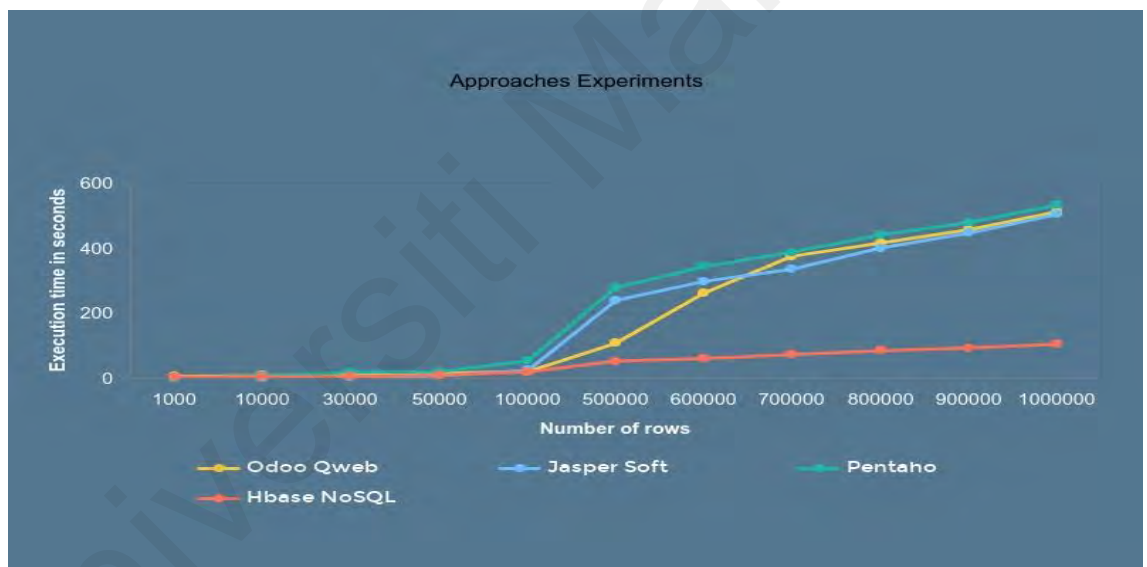


Figure 6.2: Approaches experiments results graph.

6.2.2 Query Execution Time

Another experiment was conducted to compare query execution time between the PostgreSQL database (sequential data processing) using PgAdmin tool and the HBase (parallel data processing) using Phoenix console. The following results were obtained.

Table 6.4: Query execution time in PostgreSQL and HBase.

Experiment		PostgreSQL	HBase
1	Execution time/second	0.207	0.205
	No of Records	1000	1000
2	Execution time/second	1.00	0.55
	No of Records	10000	10000
3	Execution time/second	3.20	2.11
	No of Records	30000	30000
4	Execution time/second	5.40	2.19
	No of Records	50000	50000
5	Execution time/second	10.90	3.44
	No of Records	100000	100000
6	Execution time/second	54.60	17.20
	No of Records	500000	500000
7	Execution time/second	108.00	37.61
	No of Records	1000000	1000000

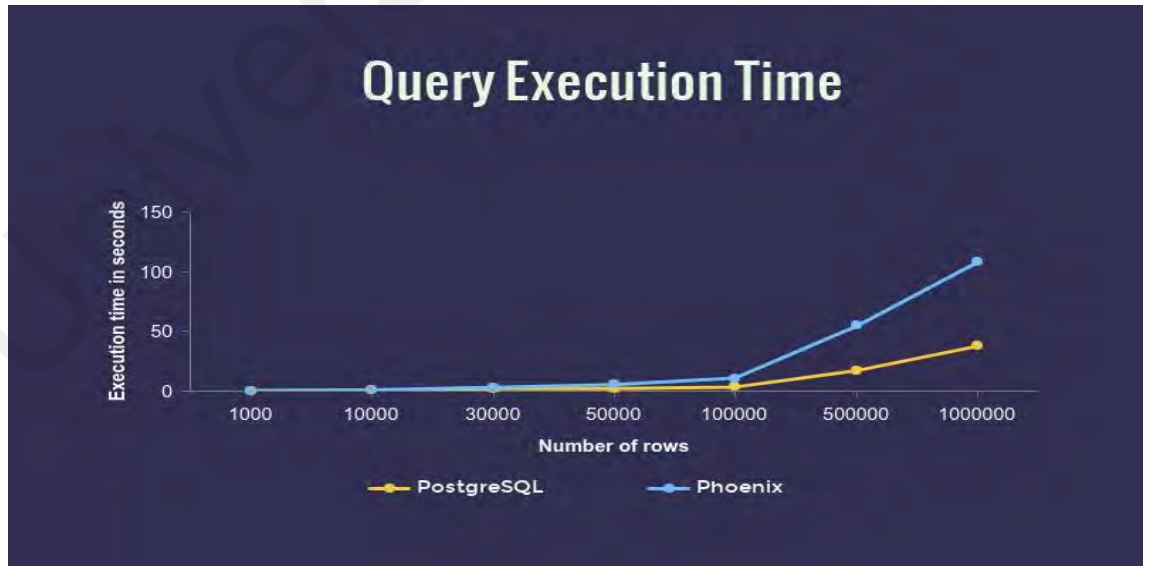


Figure 6.3: Query execution time graph in PostgreSQL and HBase.

6.3 Discussion

The experiments' results show that using the Odoo's HBase module to generate report is faster in retrieving records compared with the QWeb, Jasper Soft and Pentaho. It shows that when the number of records increases to 500000 records and more, the execution time gap between the developed solution and the previous approaches is increases. So, the developed solution is recommended for use when the number of records to generate the report is 500000 or more, so as to get the benefits of using parallel processing reports. As it was noticed the developed solution was stable and slightly increases when the number of records is increased.

(Tripathi, 2011) described the two main factors that the ERP users evaluate for the success of a report engine, depending on the process reports. This includes easiness of generating various report and high report performance in term of report processing time, which are the two factors that are achieved by the developed solution. Whereas the process of generating report is the same as using the Odoo's report engine, which identifies report render, design report template in xml file. Write SQL query to retrieve report data, all these procedures are easy and well-known for the Odoo developer; no extra knowledge is needed. The second factor is proven by the conducted experiments that showed the improvement performance among other existed approaches.

As conclusion from the experiment results, parallel data retrieval shows performance improvement over sequential data retrieval. On the other hand, PostgreSQL has been used as the data storage system in Odoo framework whereas a deployment of Hadoop and sub systems are required to switch data storage from RDBMS to NoSQL and apply parallel data retrieval to generate report, that means additional cost to be considered. Hence, there is a tradeoff between the benefit gained and the deployment cost for reporting performance improvement in terms of the organization's business goals

achievement. Organizations that process large amount of data record (minimum 500000 record per table) can get significant performance improvement when deploying developed module.

6.4 Summary

In this chapter, the performance of the Odoo HBase module (developed solution) in generating reports was evaluated in terms of report processing time and number of records compared to other existed approaches. According to the results, the performance of the new Odoo HBase module is much better when the number of records is 500000 or above compared to other existed approaches. It is also noted that the execution time increased slightly when the number of records increases, which makes the Odoo HBase module most suitable while retrieving large data records to generate reports.

CHAPTER 7: CONCLUSION

This chapter summarizes the major findings of this research, which highlight the Odoo reporting process performance using parallel processing. The chapter presents the objectives of this research, as well as the steps taken to achieve those objectives. Furthermore, it discusses the research contributions and its limitations, as well as suggestions for future researchers.

7.1 State-of- the-art

Nowadays, reports play an important role in the organization's success, and they provide quick information, therefore it helps in making appropriate decision in a timely manner. The Odoo reporting latency while processing large amount of data record has direct impact on an organization processes, decisions in proper time and achievement of business goals.

Improving Odoo's reporting performance by speeding up the data retrieval time is important to organizations as operations can be faster and reports can timely be generated. Furthermore, using an open source framework to apply parallel data retrieval can guarantee report performance improvement with low additional cost.

7.2 Fulfilment of Research Aims, Objectives and Questions

The overall aim of this research was to improve the performance of reporting process in the Odoo's framework to process large scale data records. To fulfil the overall aim of this study, it formulated three research objectives and three corresponding questions, which are as follows:

7.2.1 Research Objectives

7.2.1.1 First Objective

The first objective was to determine the limitation of the existing report processing approaches in the ERP (Enterprise Resource Planning) the Odoo's framework. This objective was achieved by reviewing the existing literature to find the gap and because there is no previous comparison has been done by other researchers to compare between the three existed reporting system (Odoo QWeb, Jasper Soft and Pentaho), a preliminary experiment was conducted to prove the limitation on the existed approaches. The experiment found that reporting performance decreased when the number of records generated reports increased to 500000 records.

7.2.1.2 Second objective

The second objective was to develop a reporting module using parallel data processing approach for generating reports up to million data records with reduced processing time in the Odoo's framework. This objective was met by developing the new Odoo HBase module that is responsible to configure configuration and connect to the HBase database to access data, execute SQL query and retrieve data in parallel to generate reports.

7.2.1.3 Third objective

The third objective was to evaluate the performance of the proposed solution in terms of processing time with the Odoo's existing approaches. This objective was achieved by conducting experiments to generate the same report from the same dataset using the developed HBase module, QWeb, Jasper Soft and Pentaho. The different sizes of data records in each experiment were defined, the execution time to generate reports with different sizes of data records were calculated. Another experiment was conducted

to compare query execution time between PostgreSQL database (sequential data processing) using PgAdmin tool and the HBase (parallel data processing) using Phoenix console.

The results from all the experiments show that the execution time of the developed solution is faster than all the existed approaches, and the execution time gap between the developed solution and the existed approaches is noticeable when the number of generated record reports is 500000 records or above. Recommended to use for data retrieval of 500000 record or more to generate reports.

7.2.2 Research Questions

7.2.2.1 First Question

The first question was ‘what is the limitation of the existing Odoo approach used to generate reports?’ To answer this question an experiment was conducted to prove that all the existed approaches in the Odoo were unable to cope with huge report. The experiment found that reporting performance decreased when the number of records generated reports is up 500000 records.

7.2.2.2 Second Question

The second question was ‘what are the suitable tools and framework for developing the proposed solution to generate large data records report with reduced processing time in Odoo’s framework?’ This objective was answered by developing the new Odoo HBase module that was capable to connect to the HBase database, access data, execute SQL query and retrieve data in parallel to generate reports.

7.2.2.3 Third Question

The third question was ‘what is the processing time of the existing approach and the proposed solution for generating report with large data records?’ This question was answered by conducting experiments to generate same report from same dataset using the developed HBase module, QWeb, Jasper Soft and Pentaho. Then, the execution time to generate reports with different sizes of data records was calculated.

7.3 Contribution

The main contribution of this research is report processing performance improvement for large data records in the Odoo’s framework, and to reduce the processing time compared with the existed approaches. In addition, the developed solution that improves the report processing performance in the Odoo’s framework is general and not prepared for a specific module. That means, it can be used by any module in Odoo. Furthermore, the HBase connection configurations for the HBase integration are dynamic, if server IP or port is changed; no need to modify the code only that can edit HBase configuration form in Odoo, and this feature provides flexible implementation to generating reports.

Additionally, the developed module can ensure the data consistency between PostgreSQL and HBase database due to the developed data synchronization methods which provide two options: real-time synchronization and scheduled synchronization. This data synchronization maintains the consistency between the two databases and can ensure the report generated by the developed module produces correct results.

7.4 Work Limitation

The developed solution used (Phoenixdb API library) to access Phoenix through remote query server. One limitation of the API in this research is that it does not

support subquery feature at all. A Subquery or Inner query or Nested query is a query within another SQL query, and it is embedded within the SELECT clause, the WHERE clause or the FROM clause. A subquery is used to return data that is used in the main query as a condition to filter the data that are retrieved.

Although phoenix supports subqueries in the WHERE clause and the FROM clause, it has two limitations in the subqueries. First, it does not support subqueries in having clause. The having clause is used when the specified condition includes aggregate functions that perform a calculation on a set of values to return a single value like average, count, minimum, maximum and sum. Meanwhile, the select statement should only return rows that meet aggregate values condition, because the WHERE clause could not be used with aggregate functions. Second, it does not support subqueries in the SELECT clause. Such limitations may prevent some complicated report to be generated with the developed solution (Apache, n.d.).

7.5 For Future Research

As mentioned earlier the developed solution in this research used a single node of Hadoop cluster. However, for future research, it is recommended to add more nodes and re-evaluate the report performance, and the number of nodes should be defined based on the dataset size because data distribution and replication provide high data processing performance. But on the other hand, data transfer latency between nodes should be considered. Additionally, Phoenix has been used as SQL layer over the HBase to execute the SQL query in parallel, other Hadoop projects can be used to apply parallel processing to execute the SQL query. Then, compare the performance between the two different Hadoop projects, and further studies should be conducted to find out other Hadoop projects that function like the Phoenix.

Another evaluation can be done using multiple report requests at the same time while conducting the experiments, then re-evaluate the report performance for each report and compare them with the single report request performance system to figure out the impact of the concurrent report processing on the report performance.

Besides, a Map-Reduce job can be developed, to retrieve data from the HBase database directly in parallel, to generate the required reports without the need for the SQL layer. This approach can be complicated and may require more experience in the Map-Reduce modelling.

REFERENCES

- Agarwal, S., & Rajan, K. (2017). Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries. *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*, 17. <https://doi.org/https://doi.org/10.7275/R5736P26>
- Apache. (n.d.). *Subqueries*. Apache Software Foundation. Retrieved January 11, 2019, from <https://phoenix.apache.org/subqueries.html>
- Ayma, V. A., Ferreira, R. S., Happ, P., Oliveira, D., Feitosa, R., Costa, G., Plaza, A., & Gamba, P. (2015). Classification algorithms for big data analysis, a map reduce approach. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 40(3/W2), 17–21. <https://doi.org/10.5194/isprsarchives-XL-3-W2-17-2015>
- Bu, Y., Howe, B., Balazinska, M., & Ernst, M. D. (2010). HaLoop- Efficient Iterative Data Processing.pdf. *Proceedings of the VLDB Endowment*, 3(1), 285–296.
- Dezdar, S., & Ainin, S. (2011). The influence of organizational factors on successful ERP implementation. *Management Decision*, 49(6), 911–926. <https://doi.org/10.1108/00251741111143603>
- Enaya, M. F. (2016). *An Experimental Performance Comparison of NoSQL and RDBMS Data Storage Systems in the ERP System Odoo*. University of Magdeburg.
- Ganesh, A., K, S., C, S., & A, M. (2016). OpenERP / Odoo – An Open Source Concept to ERP. *IEEE 6th International Conference on Advanced Computing*, 112–116. <https://doi.org/10.1109/IACC.2016.30>
- Gripe, F., & Rodello, I. (2011). A theoretical analysis of key points when choosing open source ERP systems. *JISTEM Journal of Information Systems and Technology Management*, 8(2), 414–458. <https://doi.org/10.4301/s1807-17752011000200010>

- Jindal, N., & Dhindsa, K. (2013). Comparative Study of Open ERP and its Technologies. *International Journal of Computer Applications*, 73(20), 42–47.
- Jogi, V. D., & Sinha, A. (2016). Performance evaluation of MySQL, Cassandra and HBase for heavy write operation. *2016 3rd International Conference on Recent Advances in Information Technology, RAIT 2016*, 586–590. <https://doi.org/10.1109/RAIT.2016.7507964>
- Jung, M. G., Youn, S. A., Bae, J., & Choi, Y. L. (2015). A study on data input and output performance comparison of MongoDB and PostgreSQL in the big data environment. *Proceedings - 8th International Conference on Database Theory and Application, DTA 2015*, 14–17. <https://doi.org/10.1109/DTA.2015.14>
- Kendengis, Y., & Santoso, L. W. (2018). Integration Between ERP Software and Business Intelligence in Odoo ERP: Case Study A Distribution Company. *Advances In Natural And Applied Sciences*, 12(4), 16–21. <https://doi.org/10.22587/anas.2018.12.4.4>
- Kowanda, D., Firdaus, M., Bismark, R., & Pasaribu, F. (2015). Opportunity of Free Open Source ERP System as a Competitive Advantage for Small and Medium Enterprise. *1st Unnes International Conference on Research Innovation & Commercialization for the Better Life 2015*.
- Kr.Shukla, A., & Indian, A. (2013). Enterprise Resource Planning: An Open Source System. *International Journal of Computer Applications*, 84(17), 19–21. <https://doi.org/10.5120/14678-1541>
- Lee, K., Lee, Y., Choi, H., Chung, Y. D., & Moon, B. (2011). Parallel data processing with MapReduce. *ACM SIGMOD Record*, 40(4), 11–20. <https://doi.org/10.1145/2094114.2094118>
- Li, Y., & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings*, 15–19. <https://doi.org/10.1109/PACRIM.2013.6625441>

- Moniruzzaman, A. B. M., & Hossain, S. A. (2013). NoSQL Database : New Era of Databases for Big data Analytics - Classification , Characteristics and Comparison. *International Journal of Database Theory and Application*, 6(4), 1–14.
- Moss, G. (2013). *Working with OpenERP*. Birmingham: Packt Publishing Ltd.
- Moss, G. (2017). *Working with Odoo 10* (2nd ed.). Birmingham: Packt Publishing Ltd.
- Nah, F., Lau, J., & Kuang, J. (2001). Critical factors for successful implementation of enterprise systems. *Business Process Management Journal*, 7(3), 285–296.
- Narasimhan, R., & Bhuvaneshwari, T. (2014). Big Data – A Brief Study. *International Journal of Scientific & Engineering Research*, 5(9), 350–353.
- Nayak, A., Poriya, A., & Poojary, D. (2013). Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*, 5(4), 16–19.
- Osman, A., El-Refaey, M., & Elnaggar, A. (2013). Towards real-time analytics in the cloud. *Proceedings - 2013 IEEE 9th World Congress on Services, SERVICES 2013*, 428–435. <https://doi.org/10.1109/SERVICES.2013.36>
- Pellakuri, V., & Rao, D. R. (2014). Hadoop Mapreduce Framework in Big Data Analytics. *International Journal of Computer Trends and Technology (IJCTT)*, 8(3), 115–119.
- Reis, D. (2016). *Odoo 10 Development Essentials*. Birmingham : Packt Publishing Ltd.
- Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, 1–10.

- Singh, S., Singh, P., Garg, R., & Mishra, P. K. (2015). Big Data: Technologies, Trends and Applications. *International Journal of Computer Science and Information Technologies*, 6(5), 4633–4639.
- Vaja, D., & Rahevar, M. (2016). Improve performance of ORM caching using In-Memory caching. *International Conference on Computing, Analytics and Security Trends (CAST)*, 112–115.
- Vartak, P., Desai, A., & Kamble, S. (2014). Empirical Study of Open Source ERP Systems. *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY*, 3(5), 1–5.
- Vora, M. N. (2011). Hadoop-HBase for large-scale data. *Proceedings of 2011 International Conference on Computer Science and Network Technology (ICCSNT)*, 1, 601–605. <https://doi.org/10.1109/ICCSNT.2011.6182030>
- Warneke, D., & Kao, O. (2009). Nephele: Efficient Parallel Data Processing in the Cloud Categories and Subject Descriptors. *Proceedings of the 2nd ACM Workshop on Many-Task Computing on Grids and Supercomputers 2009, MTAGS*, 09. <https://doi.org/10.1145/1646468.1646476>
- Warneke, D., & Kao, O. (2011). Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 22(6), 985–997. <https://doi.org/10.1109/TPDS.2011.65>
- Zafar, R., Yafi, E., Zuhairi, M. F., & Dao, H. (2016). Big Data: The NoSQL and RDBMS review. *ICICTM 2016 - Proceedings of the International Conference on Information and Communication Technology*, 120–126. <https://doi.org/10.1109/ICICTM.2016.7890788>