ADAPTIVE MAPREDUCE TASK SCHEDULER IN HETEROGENEOUS ENVIRONMENT USING DYNAMIC CALIBRATION

LU XINZHU

FACULTY COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

ADAPTIVE MAPREDUCE TASK SCHEDULER IN HETEROGENEOUS ENVIRONMENT USING DYNAMIC CALIBRATION

LU XINZHU

DISSERTATION SUBMITTED IN FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

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

2017

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Lu Xinzhu

Matric No: WGA120061

Name of Degree: Master of Computer Science

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Field of Study: Distributed System

I do solemnly and sincerely declare that:

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

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

ABSTRACT

MapReduce is a popular programming model for processing large-scale datasets in a distributed environment. Currently, the MapReduce implementation is based on the assumption that every compute node has the same capacity. However, in a heterogeneous environment, such assumptions may hinder the MapReduce performance where compute nodes are of varying capacity. Current works showed that make-span could be reduced if workloads are assigned in proportion to the capacity of the heterogeneous compute node. However, these approaches are static in nature where work load is assigned to each compute node based on historical data. This research is an attempt to propose an adaptive MapReduce Task scheduler, namely Adaptive MapReduce Task Scheduler Using Dynamic Calibration (AMTS-DC) to address the unbalanced node capacity problem. The proposed AMTS-DC algorithm uses the heartbeat and data locality to dynamically adapt and balance tasks assigned to each compute node. Based on the heartbeats received during early stage of the job, AMTS-DC is able to estimate the capacity of each compute node. After that, uncomputed local blocks at each compute node are reassigned so that compute nodes with greater capacity are able to reserve more local blocks. Experiment results show that AMTS-DC have relatively better performance when compare to Hadoop FIFO and Dynamic Data Placement Strategy (DDP) in dynamic heterogeneous environment. AMTS-DC has been further enhanced with the introduction of historical data and the enhanced version is named Enhanced Adaptive MapReduce Task Scheduler using Dynamic Calibration (EAMTS-DC). Experimental results show that EAMTS-DC performs better than AMTS-DC.

ABSTRAK

MapReduce adalah model pengaturcaraan popular untuk memproses set data yang besar-besaran dalam persekitaran teragih. Pelaksanaan MapReduce semasa adalah berasaskan andaian bahawa setiap nod pengiraan mempunyai kapasiti yang sama. Walau bagaimanapun dalam persekitaran yang heterogen, andaian itu boleh menghalang prestasi MapReduce di mana nod pengiraan terdiri daripada nod-nod yang mempunyai keupayaan berbeza. Kerja semasa menunjukkan bahawa masa menyiapkan kerja dapat dipendekkan jika beban kerja diberi kepada nod pengiraan heterogen mengikut kadar kapasiti nod.Walau bagaimanapun pendekatan-pendekatan ini adalah statik di mana beban kerja yang diberikan kepada setiap nod pengiraan adalah berdasarkan kepada data sejarah. Kajian ini adalah satu percubaan untuk mencadangkan satu penyesuaian penjadual tugas MapReduce, iaitu Adaptive MapReduce Task Scheduler using Dynamic Calibration (AMTS-DC) untuk menangani masalah kapasiti nod yang tidak seimbang. Algoritma AMTS-DC yang dicadangkan menggunakan denyutan jantung dan data tempatan untuk secara dinamik menyesuaikan dan seimbangkan tugas-tugas yang diberikan kepada setiap nod pengiraan. Berdasarkan denyutan jantung yang diterima semasa peringkat awal kerja, penjadual boleh menganggarkan keupayaan setiap nod pengiraan. Selepas itu, blok data tempatan yang belum diproses pada setiap nod pengiraan akan ditugaskan semula supaya nod pengiraan dengan kapasiti yang lebih besar dapat menempah blok tempatan yang lebih ramai. Keputusan experimen menunjukkan bahawa AMTS-DC mempunyai prestasi yang lebih baik apabila dibandingkan dengan Hadoop FIFO dan Dynamic Data Placement Strategy (DDP) dalam persekitaran yang dinamik. AMTS-DC telah dipertingkatkan lagi dengan pengenalan data sejarah dan versi yang dipertingkatkan dinamakan Enhanced Adaptive MapReduce Task Scheduler using Dynamic Calibration (EAMTS-DC). Keputusan eksperimen menunjukkan bahawa EAMTS-DC mempunyai prestasi yang lebih baik bila dibandingkan dengan AMTS-DC.

ACKNOWLEDGEMENTS

I wish to record my thanks to my supervisor Assoc. Prof. Dr. Phang Keat Keong for his keen interest, constant support and encouragement leading to the completion of the thesis.

I am grateful to Assoc. Prof. Dr. Ling Teck Chaw for the inspiring guidance. I also wish to thanks my friends at the Network Research Lab, Faculty of Computer Science and Information Technology for their advice and support: Chun Yong Chong, Muhammad Tayyab Chaudhry and Chai Yit Sheng. Last but not least, I am grateful to my parents and family for their continuous support and love.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AMTS-DC	Adaptive MapReduce Task Scheduler Using Dynamic Calibration
CPU	central processing unit
DDP	Dynamic Data Placement
EAMTS-DC	Enhanced Adaptive MapReduce Task Scheduler Using Dynamic Calibration.
FIFO	First In First Out
HDFS	Hadoop Distributed File System
LATE	Longest Approximate Time to End

CHAPTER 1: INTRODUCTION

This thesis presents the potential of dynamic MapReduce task scheduling in cloud computing through the proposed adaptive MapReduce task schedulers. The emergence of the big data era and the proliferation of data centers is the inevitable consequence of human ability to transform each and every event and interaction into digital data. A lot of valuable data could be derived by analyzing and extracting value from the data. However, the ever-increasing volume of data, leads to the search for more efficient data-intensive processing framework. An important aspect of this area of research is in the design and implementation of efficient scheduling mechanisms to reduce make-span and network traffic across computational cluster, data center and the Internet.

1.1 Motivation

Schedulers are important component of the MapReduce framework. There are schedulers whose function is to schedule jobs. For instance, the Fair Scheduler and Capacity Scheduler are well known job schedulers. However, there is another type of MapReduce schedulers whose focus is in the scheduling of sub-job (i.e. task) within a MapReduce function. Once a MapReduce job is submitted, the data file is partitioned into chunks of blocks of fixed sized. Hadoop Distributed File System (HDFS) will replicate, distribute and store the data blocks at the related data nodes. After that, Hadoop tasks scheduler will be initiated to process these data blocks. Our proposed task scheduler will re-schedule the replicated local data blocks to reduce unnecessary data block transfer.

This thesis addresses these challenges from following angles

i. An understanding of task scheduling in MapReduce in a heterogeneous environment

ii. Proposed dynamic measurement and calibration of computing capacity of heterogeneous compute-nodes
iii. Proposed adaptive MapReduce Task Scheduler in a Heterogeneous Environment
iv. Proposed enhancement of the Task Scheduler using historical run records

1.2 Problem Statement

In current Hadoop implementation of MapReduce, data blocks are queued and accessed sequentially using the first-in-first-out (FIFO) approach. FIFO is inefficient since no priority has been given to the processing of local data blocks. This increased network communication overhead. Unnecessary movement of data block between data-nodes increases network communication overhead, delays access of data block by the compute node and results in a longer make-span (i.e. the job completion time). In this research, our main concern is to improve the current task schedulers.

1.3 Objectives

The general objectives of this thesis are as follows:

i. To review and compare the current task-scheduler techniques in MapReduce.
ii. To propose mechanism to estimate dynamically the capacity of the compute nodes using Hadoop heartbeat
iii. To propose MapReduce task scheduler in Heterogeneous environment based on the dynamically estimated compute node capacity
iv. To strengthen the proposed MapReduce task scheduler in Heterogeneous environment based on historical run time record
v. To validate the proposed schedulers using prototype and simulations with respect to make-span and network traffic

1.4 Research Questions

The following research questions are dealt in this thesis:

i. How to monitor and model task completion status using the heartbeat?
ii. Is it possible to dynamically calibrate the computing capacity of compute nodes?
iii. Is it possible to assigned proper portion of local tasks to a compute node based on the calibrated capacity so as to minimize the make-span and reduce network traffic?

1.5 Scope

The following is the scope of this research:

i. Survey of literature related to schedulers in MapReduce
ii. Details study of schedulers is limited to the task schedulers
iii. Identify existing task schedulers and their limitations
iv. Design and implement the proposed task schedulers
v. Evaluated the proposed task schedulers using prototype and simulator

To ensure the validity of this research, a prototype is developed using actual servers with virtualization technology.

1.6 Thesis Outline

The thesis is organized into seven chapters as follows:

Chapter 1 is an introduction to the background of research, motivation, problem statement, objectives, research questions and the scope of the research. Chapter 2 is a literature review of the current works. Chapter 3 outlines the research methodology. Chapter 4 discusses the proposed adaptive MapReduce task scheduler using dynamic calibration (AMTS-DC). Chapter 5 provides the prototype and simulation results to validate AMTS-DC. Chapter 6 discusses the proposed enhanced adaptive MapReduce task scheduler using dynamic calibration (EAMTS-DC). Chapter 7 concludes with overall assessment of the proposed schedulers. The achievement and limitations of the proposed schedulers and ideas for future enhancements are also presented.

CHAPTER 2: LITERATURE REVIEW

2.1 Overview of the MapReduce model

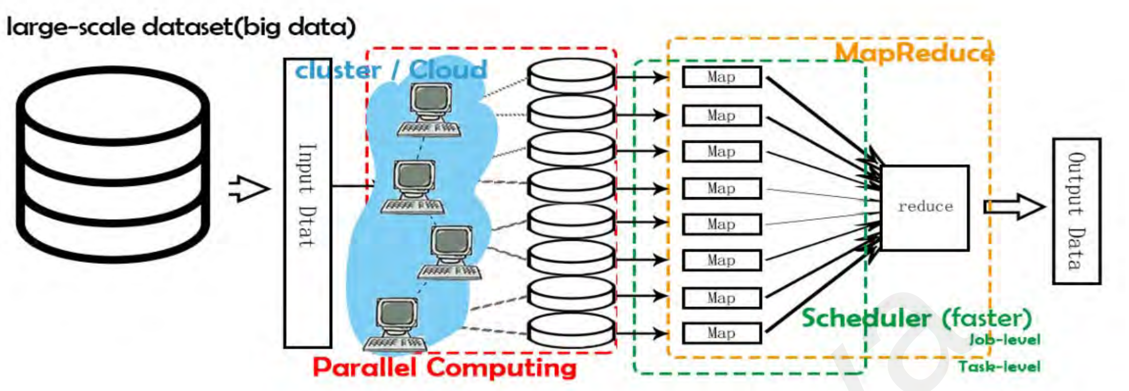


Figure 2.1 Overview of the MapReduce model

Traditional Information Systems with centralized server to store and process data do not scale well to handle large volume of data. MapReduce is one important effort in solving the big data processing issue. It is a programming model associated with the implementation details that can process large data sets in parallel on multiple compute-nodes (Dean & Ghemawat, 2008). Figure 2.1 provides an overview of the MapReduce model. The two components of MapReduce, namely the map() and reduce() functions, are inspired by functional programming. The Map() function filters and sorts (such as sorting words into queues with one queue for each word) while the Reduce() function summarizes (such as counting the frequency of words). The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication (Dean & Ghemawat, 2008).

The three main advantages of Hadoop are reliability, efficiency and flexibility.

i. Reliability: Suppose that a data-node in a computing cluster and storage fails, since Hadoop maintains multiple copies of the data, replicated data will be accessed instead.

ii. Efficient: Data are processed in parallel on multiple nodes to improve the overall efficiency/speed of processing.

iii. Flexibility: After setting up a Hadoop cluster, users need not know the underlying working principle of Hadoop to submit a job. Users are also allow to redesign and implement features in which the user deems necessary based on their requirement in the modelling of the data. Hadoop is a distributed computing platform that allows users to easily build and use.

2.2 Hadoop Distributed File System (HDFS)

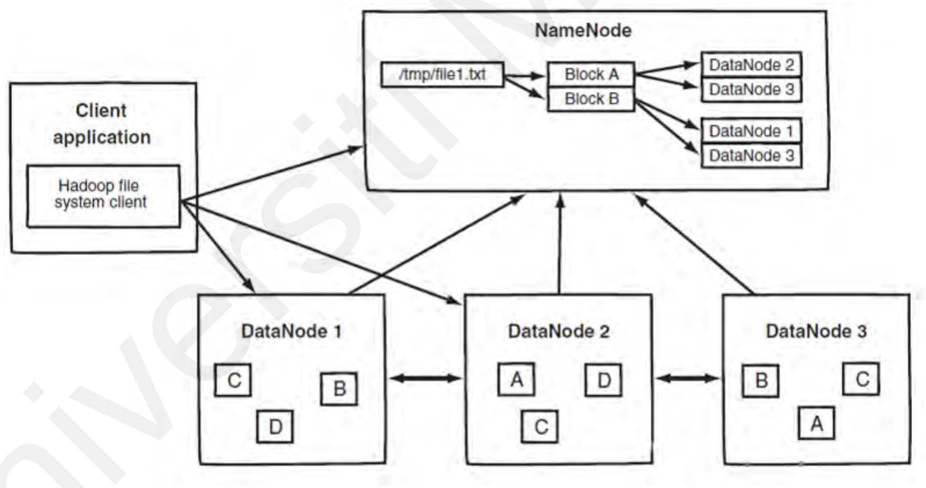


Figure 2.2 Hadoop HDFS (Alex Holmes, 2012).

The Hadoop Distributed File System (HDFS) is the storage component of Hadoop. HDFS is very similar to existing distributed file system but is optimized for high throughput and works best when handling large volume of data. It is designed to run on commodity hardware with the following assumptions and goals:

i. Handle hardware failure & provide fault tolerance

ii. Large data sets

iii. Simple coherency model

iv. "Moving computation is cheaper than moving data"

v. Portability across heterogeneous hardware and software platforms

HDFS has a master/slave architecture. Figure 2.2 depicts the HDFS architecture showing an HDFS client talking to the master NameNode. The NameNode, as a master server, manages the file system and regulates file access by clients. Data files are partitioned into blocks. Data blocks can be replicated to produce identical copies of each block. In Figure 2.2, data replication is set to 2. The NameNode determines the mapping of data blocks to data nodes. The DataNodes upon receiving instruction from the NameNode will perform block operations such as block creation, deletion, and replication.

2.3 MapReduce

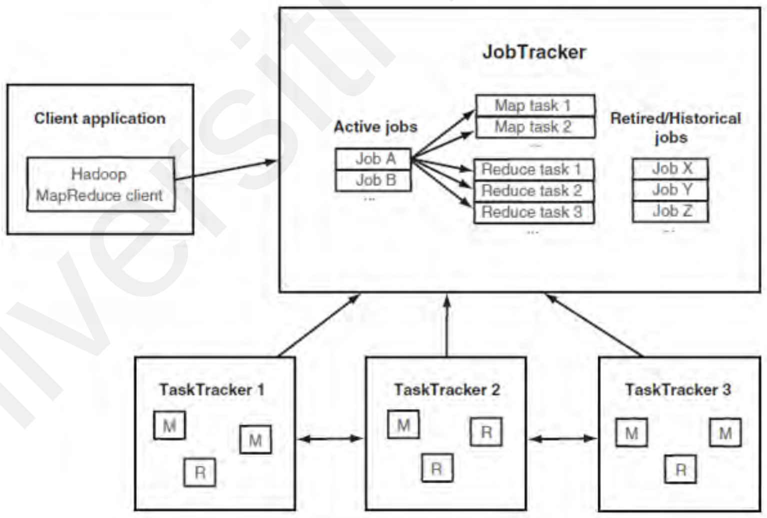


Figure 2.3 MapReduce Architecture and Job Submission (Alex Holmes, 2012).

Figure 2.3 depicts the MapReduce architecture. Major entitles include the client application, the JobTracker and a number of TaskTrackers. The JobTracker, running on a master node, manages and coordinates the jobs. A TaskTracker, running on each compute node, will launch and coordinate the tasks executed within that node. In Figure

2.3, a MapReduce client application is talking to the JobTracker. The JobTracker accepts the job request from the client and schedules the job to the TaskTrackers in the form of smaller tasks. The TaskTrackers then perform the assigned map and reduce tasks.

In order to provide fault tolerance, Hadoop replicates data blocks during HDFS upload. Additional of the same data is replicated and stored in different nodes. When there is a node failure and render some data blocks inaccessible, these data blocks can be accessed via other nodes.

2.4 Schedulers and related works

This sections looks into current works in MapReduce schedules to identify useful insight which can be include in the proposed task scheduler.

2.4.1 Default Schedulers in Hadoop

Three schedulers are available in the Hadoop implementation (Tom White, 2009):

i. First-in-First-out (FIFO) scheduler
ii. Capacity scheduler
iii. Fair scheduler

FIFO Scheduler is the default Hadoop scheduler. Jobs are scheduled based on the first-come first-server principle. In other words, FIFO Scheduler places jobs in a queue and runs them based on the order of job submission. A job received is partitioned into smaller tasks. Tasks are put into a queue and the tasks can be accessed by the JobTrackers. One of the advantage of FIFO is in its simplicity. The disadvantage of FIFO is that no consideration is given for capacity of the node. Capacity Scheduler allows the sharing a large cluster by organizations and guarantee each organization a minimum capacity. The benefit of Capacity Scheduler is that an organization can use and access excess capacity not being used by others. Fair Scheduler assigns resources to jobs and ensure that all jobs get an equal share of resources over time. The scheduler ensures short jobs to finish within reasonable mask-span without starving long jobs.

Zaharia et al. (2008) have developed the LATE (Longest Approximate Time to End) algorithm. The 3 principles of LATE are: tasks are prioritized for speculation, fast nodes are chosen to execute tasks, and speculative tasks are capped to prevent thrashing. Hadoop assumes all compute-nodes has similar computing capacity. The LATE algorithm is one of the earliest work taking into the issue of heterogeneous environment. Late algorithm, by calculating the remaining number of tasks, to calculate the actual number of slow task, these relatively slow to do the task. However, the problem with LATE is that it is difficult to identify really slow task.

Quan Chen et al. (2010) proposed SAMR: a Self-Adaptive MapReduce scheduling algorithm. SAMR can adapt to the variation in the environment by computing dynamical task progress. SAMR attempts to improve the computation of LATE. By analyzing historical record, SAMR is able to accurately identify actual slow task and backup the slow task to conserve storage. SAMR separate slow task into 1) slow map task and 2) slow reduce task. Backup tasks will be assigned to faster nodes. SAMR performed better than LATE.

Xie et al. (2010) pointed out that ignoring the data locality in heterogeneous environments could hindered MapReduce performance. They addressed the problem of data placement across nodes so that the data to be processed by each node is balanced. Data is allocated in proportion to the node capacity. However, as data replication is removed from the system, fault tolerance of Hadoop is not preserved.

He et al. (2011) proposed the MatchMarking algorithm to improve data locality at the task allocation level. A local task, compared to non-local task, always has a higher priority to be executed. In addition, each slave has a marker to mark a node to ensure that each node has fair chance to get its local task. In other words, MatchMarking gives every slave node a fair chance to grab local tasks before any non-local tasks are assigned to any slave node. The drawback of MatchMarking is no consideration has been given to heterogeneous environment.

Lee et al. (2014) proposed the Dynamic Data Placement (DDP) Strategy for Hadoop in Heterogeneous Environments. DDP adapts and balances data stored in each node based on the computing capacity of each node. Higher capacity nodes get more data. DDP is static (even though the title contains the term "Dynamic") and is based on historical run time data. DDP is very accurate should the computational environment remains static. However, in non-static environment, i.e. should there be fluctuation in the computation capacity of nodes within the make-span, DDP may not be able to perform well. Due to the exclusion of the HDFS and data replication, DDP does not support fault tolerance.

Gu et al. (2013) proposed SHadoop to improv MapReduce performance by optimizing job execution mechanism in Hadoop clusters. SHadoop replaces the heartbeat mechanism by instant messaging to monitor task to speed up the scheduling and execution of performance-sensitive task. SHadoop improves make-span especially that of short jobs.

Xu et al. (2015) proposed the Dynamic Task Splitting Scheduler (DTSS) to address the tradeoffs issue between fairness and data locality during job scheduling. On a non-data-local node, DTSS dynamically splits a task and execute the split task immediately to improve fairness. The drawback of DTSS is the copying of data blocks from local node to remote node as this increased network communication cost.

Anjos et al. (2015) proposed MRA++, a novel MapReduce framework design that considers the heterogeneity of nodes during data distribution, task scheduling and job control. MRA++ characterizes a smaller number of machines as stragglers and executes a larger number of tasks concurrently in a heterogeneous environment. A knowledge base of execution time is used prior to the data distribution. MRA++ has shorten make-span. The disadvantage of MRA++ is the exclusion of slow machines and this is seen as a waste of resource.

Cheng et al. (2014) proposed ANT. ANT is a self-adaptive genetic algorithm based task tuning mechanism. Ant will first divide the heterogeneous nodes into a homogeneous sub-cluster based on their hardware configurations. Each subcluster is treated as a homogeneous cluster and self-tuning algorithm is apply to the subcluster. ANT has higher complexity.

Polo et al (2010) proposed a task scheduling mechanism that enables a MapReduce runtime to dynamically allocate resources in a computing cluster. The allocation is based on the observed progress rate achieved by the jobs, and the completion time goal associated with each job. The proposed task scheduler is able to predict the performance of concurrent jobs and dynamically allocate resource for the jobs.

Selvarani and Sadhasivam (2010) proposed a task group scheduler in environment with heterogeneous resourse cost and computation capacity. Tasks are grouped according to the processing capability of the available resources. The cost-based scheduler allows the mapping of tasks to available resources. The scheduler groups a number of user jobs together according to a particular resource's processing capabilities. It then send the grouped jobs to that resource. However, the mechanism of sending jobs to resources does not suit the modern architecture and the actual movement may incur higher overhead.

Mao et. al (2011) proposed a task level scheduler. Both hardware configuration and real-time workload of the nodes are taken into consideration to shorten make-span and improve hardware resource utilization. Tasks are assigned to Task Trackers according to the workload of slave nodes. The scheduler is able to adaptively adjusts the MapReduce map and reduce slot to increase cluster resources usage.

Tang et. al (2012) proposed the MapReduce Task Scheduler for Deadline (MTSD) scheduler by taking into account data locality and cluster heterogeneity. MTSD allows a user to specify a job's deadline and so that the job can be completed before the deadline. MTSD improved data locality and shortened average task completion time. However, since the node classification algorithm used by MTSD has not been incorporated into the Hadoop distribution file system, fault tolerant of MTSD could not be preserved.

Althebyan et. al (2014) proposed the MTL scheduler that is based on a multi-threading principle. In the algorithm, a cluster is divided into multiple blocks where each one of them is scheduled by a special thread. In particular, multi –threading approach is used. When there are jobs to be processed, the threads start searching in their blocks node for local map task. MTL is able to improve performance. However, the proposed MLT lacks details of the relationship between the threads and Hadoop architecture. MLT may not be feasible in real Hadoop cluster.

Dai and Bensaou (2016) proposed a Hadoop MapReduce task scheduler called dynamic priority multi-queue scheduler (DPMQS). DPMQS improves data locality of jobs. Jobs that are near to completing their map phase are given higher priority so that waiting time could be reduced. PMQS uses heartbeat and a privilege threshold value to monitor current job progress rate. However, the proposal lack technical details on how DPMQS could be incorporated in Hadoop.

Table 2.1 Summary of strength and weakness of major current works

Methods	Description	Strength	Weakness	Note
LATE Zaharia et al. (2008)	LATE (Longest Approximate Time to End) Find slow tasks by computing remaining time of all the tasks**.**	always speculatively execute the task that might finish farthest into the future; shorter make span	Does not take into account of data locality when launching speculative map tasks Cannot accurately identify really slow tasks	Task level Non-standard heuristic to estimate time
SARE Chen et al. (2010)	SARE improves LATE algorithm by calculating task progress dynamically	SAMR does not launch backup tasks on slow nodes and thus ensuring that the backup tasks will not be slow tasks. Performed better than LATE (Zaharia et al. (2008)	Data locality is only considered when launching backup tasks History data might get obsolete/inaccurate	Task level
Data Locality Xie et al. (2010)	Allocates data according to node capacity; (Heterogeneous)	Move data from one node to another in execution Time to Improve performance	Removal of data replication; Negative impact on the data loss if there are node failures	Task Level HDFS
SAMR Chen et al. (2011)	The main idea is to give every slave node a fair chance to grab local tasks	Higher data locality Reduces network communicatio n	Does not consider heterogeneous environment	Task level
DDP Lee et al. (2014)	Adapt and balance data stored in each node based on the computing capacity of each node in a heterogeneous Hadoop cluster	Higher capacity nodes get more data Based on historical run time data More efficient	Historical data may not be accurate or able to reflect the true situation should the environment becomes more dynamic; Data replication is removed and thus lacks fault tolerance	Static and not dynamic (Though the title contains the term "Dynamic"); Task level

SHadoop Gu et al. (2014)	Optimizes the setup and cleanup tasks of a MapReduce job Replaces heartbeat mechanism by instant messaging to monitor task	Improves make-span especially for short jobs Dynamic	Non-standard mechanism as heartbeat has been replaced by instant messaging	Job/Task level
DTSS Xu & Cai (2015)	Dynamically splitting a task and executing the split task on a non-data-local node to improve fairness	Split task to improve fairness	Increased network communication	Job/Task level
MRA++ Anjos et al. (2015)	MRA++ classifies a smaller number of machines as stragglers and executes tasks concurrently in a heterogeneous environment.	knowledge base of execution times is used prior to the data distribution Shorten make-span	Need chunk (block) copy from local to remote node Higher network overhead Slow machines are excluded from processing(waste of resources)	Task and Job Setup Phase
ANT Cheng et al.(2014)	ANT is a self-adaptive genetic algorithm based task tuning mechanism.	Improves make-span Automated configuration	Higher computational complexity	Task level

2.5 Gap analysis

The literature survey has identified gaps that exist in the current works and are listed as follows:

i. Lack of dynamic load predication/estimation mechanism to enable fair division of tasks among heterogeneous nodes.

ii. The current practice of using of static historical run time record to divide tasks among heterogeneous nodes may not work since the underlying parameters such as hardware/network configuration/work load of the computing cluster may change over time.

iii. Lack of fault tolerance support after data blocks are partitioned and distributed to the data/compute nodes. Most of the time only one set of data block is available for the entire cluster.

iv. Lack of dynamic mechanism in predicting/estimating computing capacity of the heterogeneous compute nodes

v. Lack of effective mechanism to reduce data movement among compute nodes. Local compute nodes should be given higher priority as far as local data are concern.

All the above gaps will be handled in our proposed research work and the efficiency of the proposed algorithms will be proven by comparing with previous works.

2.6 Summary

This chapter provide an overview of MapReduce and Hadoop Distributed File System. Current major works in the area of MapReduce scheduler are summarized in Table 2.1, with the strength/advantages and weakness/disadvantages identified. Based on this finding, we hope to receive helpful idea and better insight when designing our proposed task schedulers.

CHAPTER 3: RESEARCH METHODOLOGY

This chapter describes the methodology carried out to achieve the objectives of this research work. It comprises of approach and strategy in which this research were carried out. The tools and software used at different stages are discussed.

3.1 Outline of Methodology

The goal of this research is to design MapReduce task schedulers that should be able to utilize the available heterogeneous compute nodes efficiently. The task schedulers should also utilize lesser network resources during computation. By efficient we meant that our proposed method, given a MapReduce job, would be using shorter make-span and lesser data movement to complete the job. Lesser data movement among compute nodes is achieved, by ensuing that within each compute node, higher priority is given to local data.

To accomplish the objectives of the research, a scientific method depicted in Figure 3.1 is utilized.

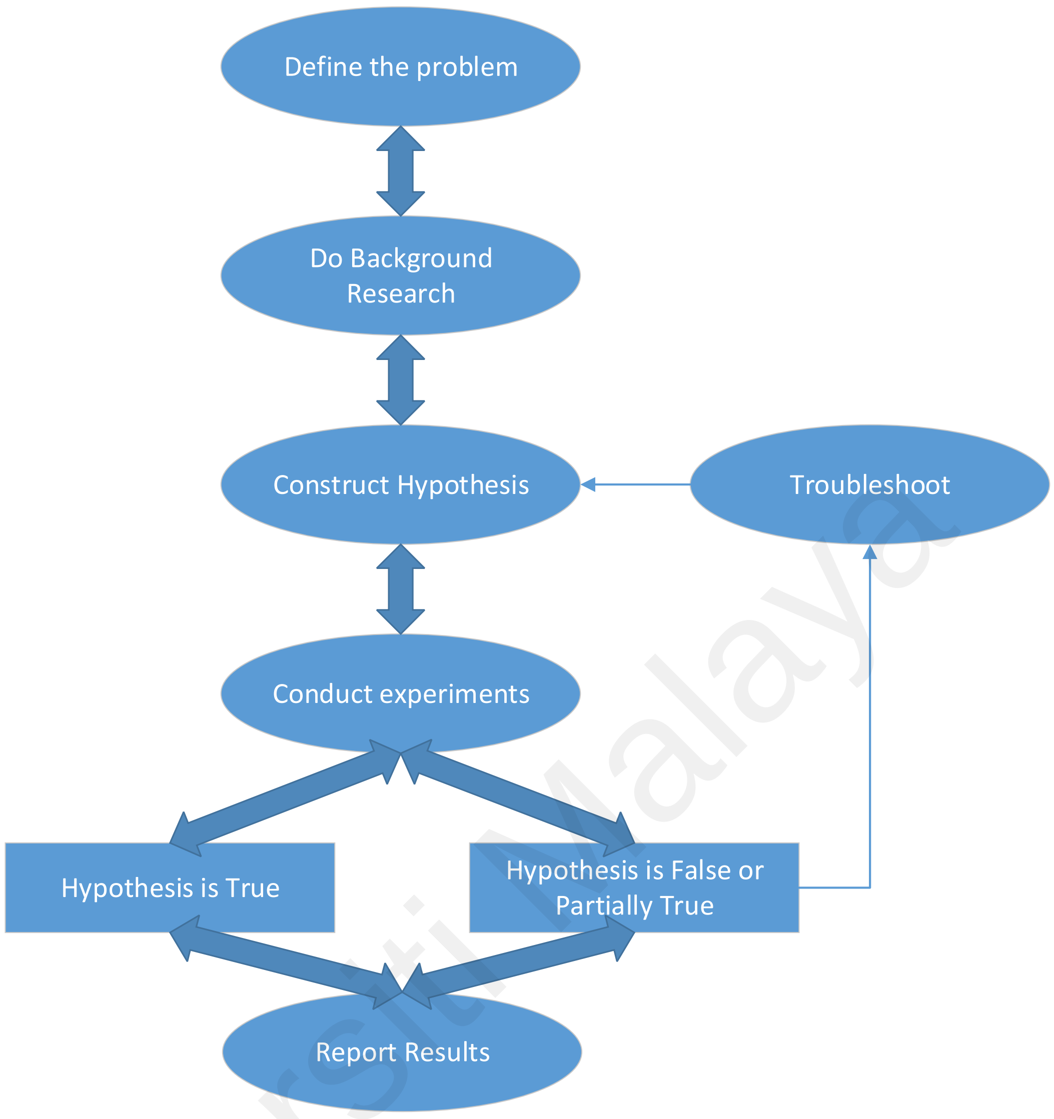


Figure 3.1 Research Methodology Flowchart

This section briefly explains the steps depicted in the research methodology flowchart.

a. Defining the problem:

Currently MapReduce implementation is based on the assumption that every compute node has the same capacity. What happen if the compute nodes have different capacity? The make-span will depend on the slowest compute node. How would this affect the task scheduler? In this stage of study, the need for this research will be strengthened and the aim and objectives of this research formed.

b. Background research

At this point of study, thorough study of current works in this area from academic journals, conference proceedings, white papers etc. will be carried out. This helped to identify good practice and widen the knowledge about this research topic. The literature survey will also paved way for better focus and allowed us to take note of the topic as something which worth further thought and investigation.

c. Hypothesis construction

A hypothesis is formed to provide a direction to further the research investigation. Attempts are made, based the background research and literature survey in the earlier step. The purpose of the hypothesis is not to arrive at the perfect answer to the question but with an effort to guess the possible answer of the question. At this stage, mechanisms/algorithms that are able to schedule MapReduce task more efficiently in a heterogeneous environment will be formulated.

d. Conduct Experiments/Simulations

After the formation of a hypothesis, the hypothesis will be tested through a set of experiments. To proof the feasibility of the proposed hypothesis and the concept, a prototype will be implemented as an extension to Hadoop (a MapReduce implementation) in a computer cluster environment. The hypothesis with the set of proposed mechanisms/algorithms will be implemented. After that, using the prototype the performance of the hypothesis is evaluated and contrasted against current works. Carefully designed and controlled experiments are crucial in the scientific method, as they are used to prove a hypothesis right or wrong, and to formulate scientific theories. The experiments must also be reproducible so that they can be tested/verified by future researchers. The prototype is critical to verify the claim advantages of the proposed mechanisms/ algorithms.

Due to the complexity of the hypothesis, the evaluation of the hypothesis is complemented with simulation so that more complex scenario could be provided to test the proposed hypothesis. Given the hardware limitation and time constraint, the prototype is unable to handle more complex scenario with greater number of compute nodes and with larger input data size. A simulator is built so that more complex scenario could be used to test the hypothesis. At this stage, mechanisms/algorithms that will schedule MapReduce task more efficiently in a heterogeneous environment will be evaluated.

e. Analyze the Data and Draw a Conclusion

As experiments are conducted using the prototype and simulator, several trials are conducted to ensure that the results are consistent. Experimental results are carefully recorded so that conclusion can be drawn regarding the strength of the hypothesis. At this phase, benchmarking with current algorithms will also be carried out. If the experimental results prove that the hypothesis is correct, the original question is answered and the experiment could be concluded; otherwise, new hypothesis is formulated or further refinement to the hypothesis is conducted and further experiments are required to test the refined or new hypothesis. This process goes on until a hypothesis can be proven correct by the experiments. At this stage, mechanisms/algorithms that will schedule MapReduce task more efficiently in a heterogeneous environment will be validated.

3.2 Objectives, Methods and Tools used

Table 3.1 Research objectives, Methods and Tools used

No	Research Objective	Methods	Tools Used
i.	To review and compare the current task-scheduler techniques in MapReduce.	Literature Search	-
ii.	To propose mechanism to estimate dynamically the capacity of the compute nodes using Hadoop heartbeat	Critical review of Literature Experiment	Hadoop prototype and computer cluster Java Simulator
iii.	To propose MapReduce task scheduler in Heterogeneous environment based on the dynamically estimated compute node capacity	Critical review of Literature experiment	Hadoop prototype and computer cluster Java Simulator
iv.	To strengthen the proposed MapReduce task scheduler in Heterogeneous environment based on historical run time record	Critical review of Literature experiment	Hadoop prototype and computer cluster Java Simulator
v.	To validate the proposed schedulers using prototype and simulations with respect to make-span and network traffic	Experiment	Hadoop prototype and computer cluster Java Simulator

In Table 3.1 how each objective of the current study is achieved via literature search, experiment and simulation is elaborated. Software and hardware tools used are also included in the table. As outlined above, to prove the concepts and feasibility of the proposed algorithm, a prototype is developed and implemented empirically using actual servers. Actual Hadoop framework will be used in our investigation. Using the prototype, the relationship and interaction between MapReduce, HDFS, job schedulers and task schedulers is critically analyzed. After that, components which can affect the task execution speed such as job queuing list in task scheduler, heartbeat, data replication and local task will be the focus. At this stage, relevant parameters will also be identified. Due to the limited resource available in the research lab, the challenges in the creation of heterogeneous and dynamic environment is resolved using virtualization technique. All compute-nodes are deployed as virtual machines. Heterogeneous computational capacity of compute-nodes are created by adjusting the CPU and RAM configuration of the virtual machines. The prototype is also useful to test the actual heartbeat process, blocks (i.e. tasks) process and the actual make-span and traffic load. In addition, to provide more variability in terms of data size, greater range of heterogeneous computation capacity, a Java simulator has been developed to analyze and measure the task schedulers' performance in terms of make-span, block movement and network traffic.

The prototype is implemented by modifying JobTracker.java and JobInProgress.java in Apache Hadoop. The modifications extend the heartbeat, and returns the current computing capacity. The return value and node status (to capture non-running local tasks) are used so that local tasks could be re-assign to the compute nodes in such a way that local tasks assigned to a node is proportional to the capacity of the node. In other words, more powerful nodes are assigned with more local tasks.

With the setting up of the Hadoop MapReduce prototype, current works in MapReduce task scheduling mechanism could be study and evaluated. The *wordcount* Hadoop application (an actual MapReduce application) with actual workload will be run in the prototype. Due to the hardware resources constraint, only four servers are used in the prototype. The prototype will reside within a cluster of virtual machines connected by a computer switch. Experimental results gathered will be compared in terms of performance metrics such as make-span and block movement (network traffic).

Since the prototype is limited by the number of compute node, data size, CPU computation variation and RAM size variation, more complicated scenario will be evaluated using the Java simulator. In this way greater variability in terms of data size and computation power could be tested and evaluated.

3.3 Chapter summary

This chapter has elaborated the research methodology adopted using a flowchart diagram. Details is also provided on how the research objectives could be achieved based on the diagram using relevant methods and tools.

CHAPTER 4: ADAPTIVE MAPREDUCE TASK SCHEDULER USING DYNAMIC CALIBRATION (AMTS-DC)

This chapter analyses task scheduling in MapReduce on Hadoop implementation and look into the effect of data locality on MapReduce in heterogeneous environment. The discussion will look into the basic operation on how in Hadoop, the original large data file, after being partitioned into smaller blocks and stored in the HDFS, are processed by Hadoop default and DDP. Details of the operation is examined within the period of the make-span to identify areas whereby task scheduling could be further enhanced.

4.1 Heterogeneous Environment

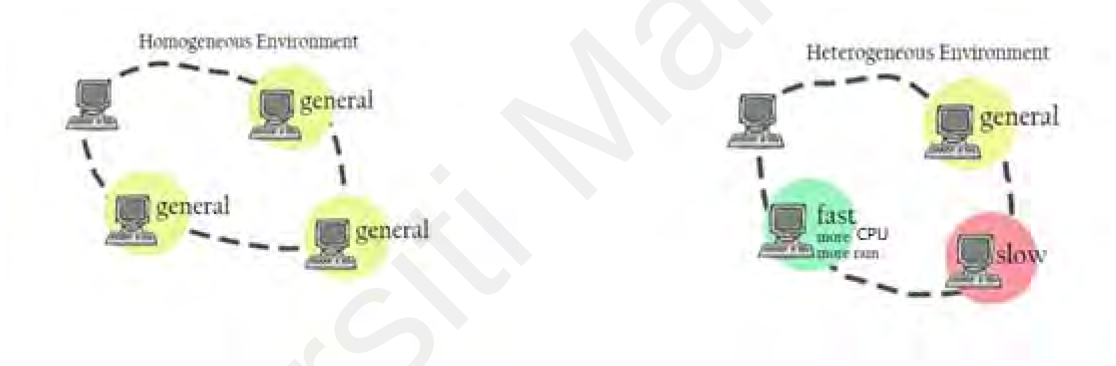


Figure 4.1 Homogeneous/Heterogeneous computation nodes

Figure 4.1 depicts the different between homogeneous and heterogeneous computing environment. A homogeneous computing environment is one which has homogeneous network and homogeneous machines with equal computing capacity; otherwise it is a heterogeneous computing environment.

4.2 Hadoop FIFO

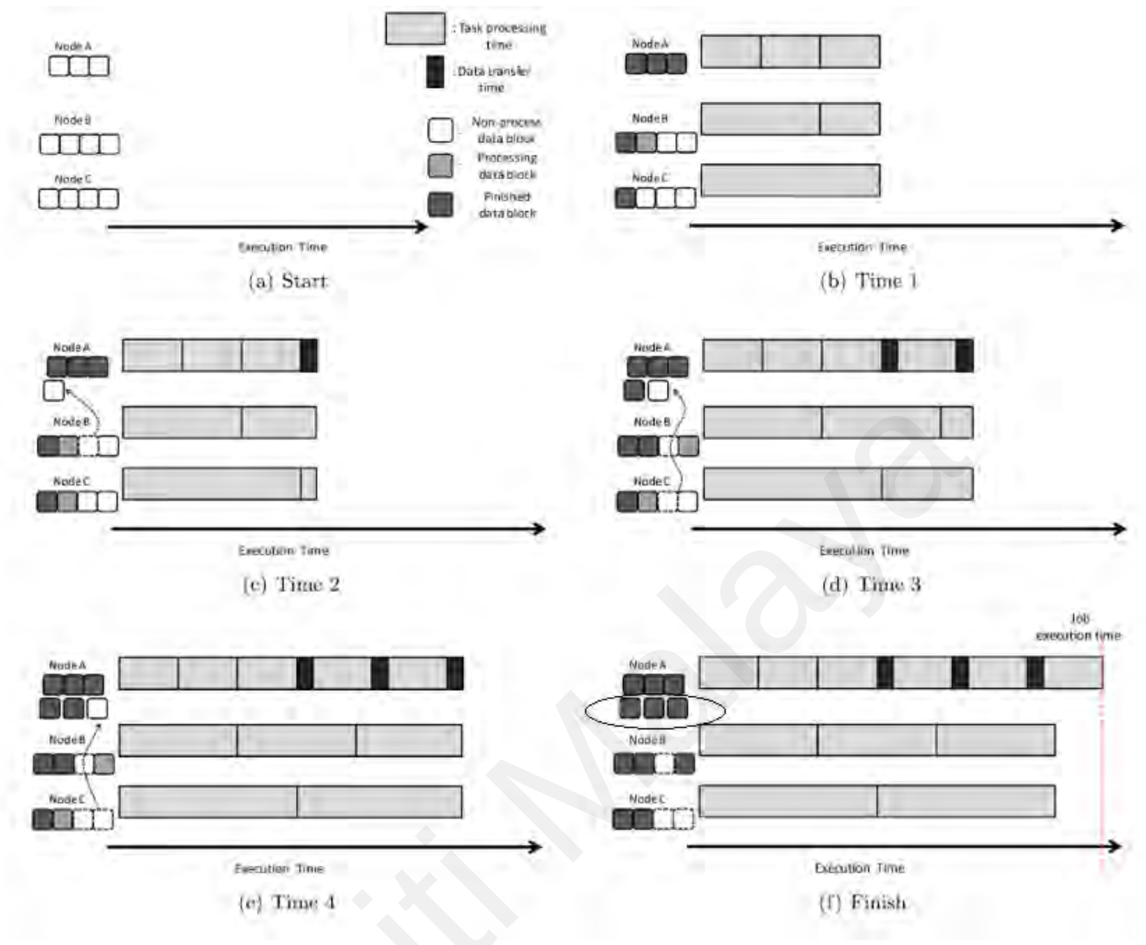


Figure 4.2 Hadoop default data allocation strategy

The basic design of Hadoop assumes a homogeneous environment. However, in real-world environment this may not be the case as it may consist of cluster and servers of various specification. Figure 4.2 illustrates a situation where compute nodes have different computing capacity. The relative computing capacity of nodes A, B and C are respectively 3, 2 and 1. Since node A is the fastest, it is able to complete its entire local block in a shorter time. Node A will process non-local blocks which are located in nodes B and C. From Figure 4.2 (f), 3 data blocks (the blocks which are surrounded by an eclipse) are transferred from the other nodes. This prolong the make-span.

4.3 Dynamic Data Placement (DDP)

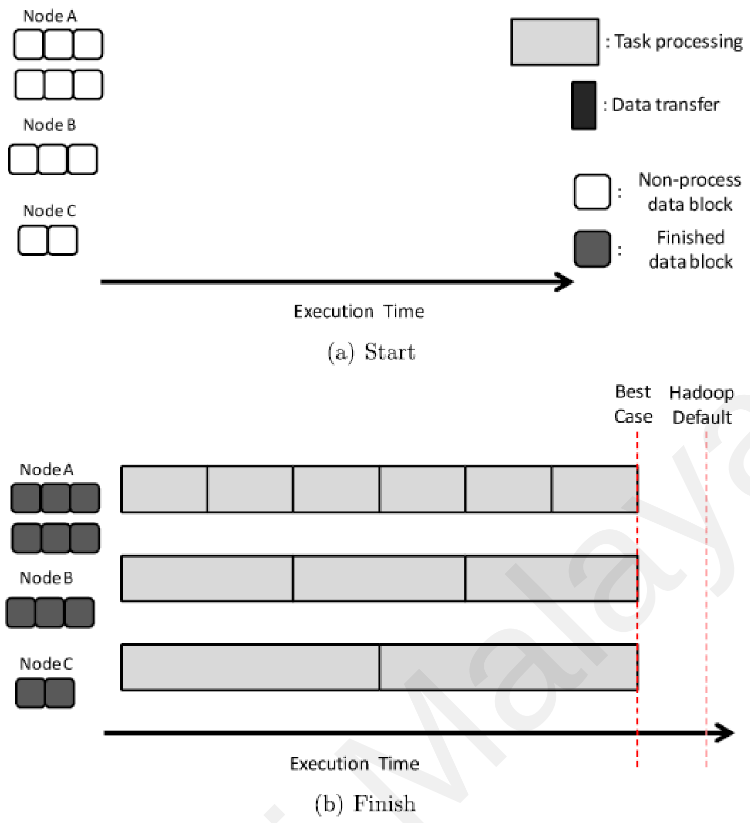


Figure 4.3 DDP data allocation

DDP (Lee et al, 2014) is a solution to resolve the issue of Hadoop default in heterogeneous environment. In DDP, data blocks are allocated in proportion to the computational capacity of each node. The computation capacity is obtained from historical record of previous run. In Figure 4.3., since the relative computing capacity of nodes A, B and C are respectively 3, 2 and 1, the number of blocks assigned to node A, B and C will also be in the ration of 3, 2 and 1. In this way, DDP requires no data block movement between compute nodes within the duration of job processing. This shorten the make-span.

Limitation of DDP

1) DDP relies heavily on the use of historical run time record. The relative computing capacity of compute-node within a cluster are historical record generated from previous

run. The record is a tuple denoting the <computation ratio, amount of data processed>. Historical data though useful, may not be accurate due to the following:

i. Changes in hardware configuration of the cluster: For instance the addition of RAM, upgrading of CPU, additional/removal of computing nodes within the cluster etc.
ii. Though with the same name, a job recorded in the historical data might not be the same with the current running job. E.g. Difference versions of *wordcount* MapReduce application. In addition, the argument value and argument types may also affect the execution time.
iii. Dynamic computation environment where additional process or application is running in the compute node

2) DDP does not support fault tolerance. Since only one set of data block is available for the entire cluster. On the other hand, MapReduce by default creates multiple copies or replication of data items and storing them at different compute/data nodes. This increases the availability so that if a node fails, data could be accessed from a different node.

4.4 JobTracker, TaskTracker and heartbeat

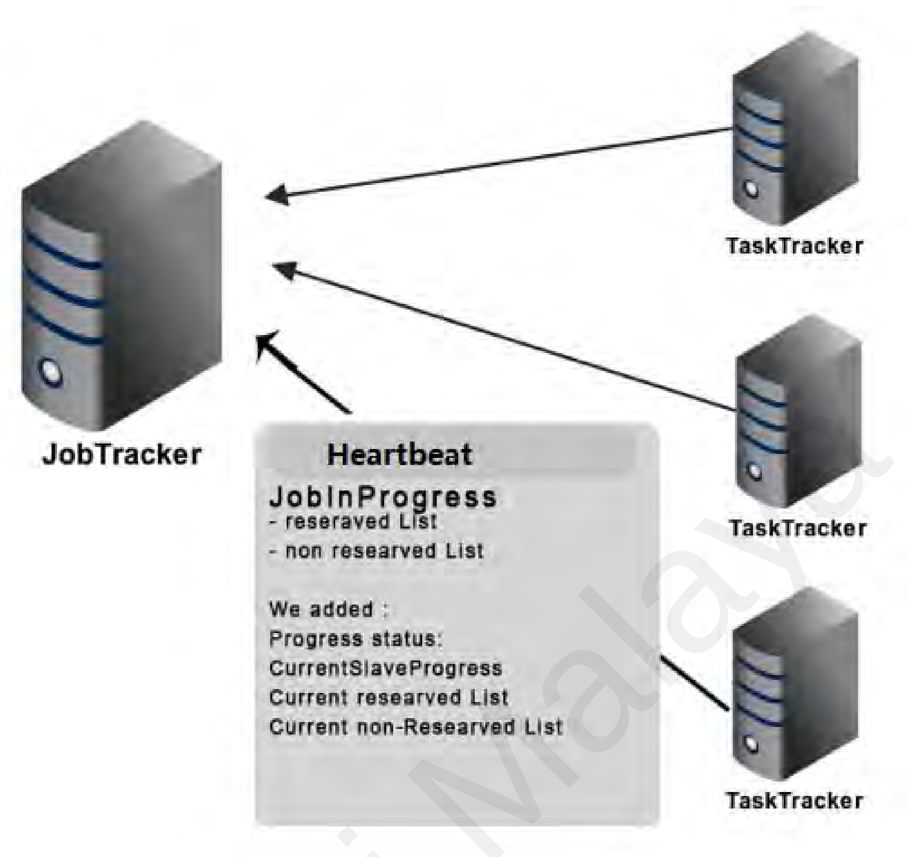


Figure 4.4 Heartbeat Mechanism in AMT-DC and the add-ons

Prior to our discussion on how to improve DDP, a discussion of the Hadoop heartbeat mechanism is required. Figure 4.4 shows the job execution. Here, the JobTracker will schedule all the jobs and distribute the tasks to the TaskTrackers. As depicted earlier in Figure 2.3 (Chapter 2), when a JobTracker accepts job request from a client, it will schedule the job to the TaskTrackers in the form of smaller tasks. In other words, a job is partitioned into tasks before the job can be executed at the compute-nodes. The TaskTracker will then perform the tasks and the results are returned to the JobTracker. The Job-Tracker and TaskTrackers communicates using heartbeat. Each TaskTracker is assigned with a set of task slots. The TaskTracker can then work on the tasks. When the task slot is empty, the TaskTracker will use the heartbeat to inform the JobTracker. If there are more tasks to be completed, the JobTracker assigns the tasks to the TaskTracker using heartbeat response. Via the regular heartbeat, the JobTracker is able to know the status of the tasks assigned. The default heartbeat interval of Hadoop is 3 second. Figure 4.4 shows the heartbeat mechanism in AMT-DC and the add-ons.

4.5 Proposed AMTS-DC Task Scheduler

Based on the discussion in 4.2 and 4.3, to improve MapReduce performance in heterogeneous environment, the proportion of local blocks assigned to the compute-nodes should be proportional to the computational capacity of the nodes. This is exactly how DDP is designed to out-perform the default Hadoop FIFO. Firstly, DDP assigned the data blocks without going through HDFS, i.e. by passing HDFS. The blocks re-assignment in DDP are actually direct block placement into the file system. This made the proposed solution less plausible as HDFS is an essential component of Hadoop. Secondly, in DDP, only one set of data block is available for the entire cluster. DDP lacks fault tolerance support. The failure of a data node renders the data blocks stored in the node inaccessible and the job will have to be aborted.

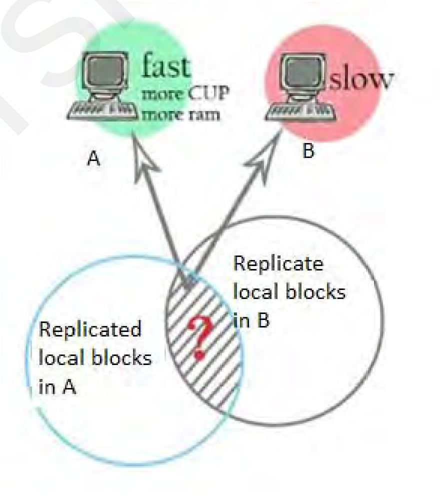


Figure 4.5 More local data blocks for faster node

The proposed Adaptive MapReduce Task Scheduler Using Dynamic Calibration (AMTS-DC) is designed without modifying the current HDFS implementation. In the initial stage, data files are replicated to the compute nodes by the instruction from the NameNode via HDFS. In the second stage, heartbeat mechanism in Hadoop is use to estimate the current relative computing capacity of the compute node instead of based on the historical data. As depicted in Figure 4.4, the Hadoop heartbeat is extended so that information of current slave progress, current reserve list and non-reserved list could be piggyback in the heartbeat and communicated to the JobTracker. In the third stage more local blocks will be reserved to compute nodes with greater capacity. This done by reserving more local tasks within the shaded area (depicted in Figure 4.5) to more powerful nodes.

4.5.1 AMTS-DC Algorithm

```
 //HDFS partitioned job data file into blocks
 //NameNode randomly distribute blocks to slave nodes
 Let k = int((TotalBlock*replicate)/n)
 For Each slave node S
    Initialize the reservedBlock and non-reservedBlock
 Ticks := 0     // Ticks is the time
 While there are still data blocks to be processed
   If Ticks equals Update_Computing_Ratio_Now
     updateBlockReservedByEachSlaveNode
    For each slave node S
        If reservedBlock of Sis not empty
           Process a block in Reserved Block
        Else If non-reservedBlock of S is not empty
           Process a block in Non-reserved Block
        Else Move and Process a block from other Slaves
    Make-span=Ticks
    Ticks=Ticks+1
  End While
  Return Make-span
End
```

Figure 4.6 Pseudocode of AMTS-DC

Figure 4.6 depicts the pseudocode of AMTS-DC, which provides a high overview of the AMTS-DC. Figure 4.6- 1 to Figure 4.6- 4 provides details of the proposed AMTS-DC algorithm.

```
   Variable:

1 TotalTask (a job partitioned into many tasks),
2 TotalBlock (data file is partitioned into many blocks; TotalBlock
   = TotalTask),
3  r (replication number)and n (number of slave),
4  HisRecExist (historical record of job and the relative node
   capacity exist)
5  RelCapOfSlave (Relative Computing Capacity Of Slaves)
6  SetOfComputeNode (S1,S2,..SN)
7  update_CR_Now (time to update the computation Ratio of each slave),
8  Block [S] (set of block assigned to slave S,
9  RBlock[S] (set of block assigned to slave S and reserved for slave
   S), NBlock[S] (set of block assigned to slave S but not reserved
   for slave S),
10 RoneB[S] (one block in RBlock[S]),
11 NoneB[S] (one block in NBlock[s]),
```

Figure 4.6- 1 AMTS-DC algorithm (Variables)

```
Proc Main()
1  Begin
2    //HDFS partitioned job data file into blocks {B1,B2,B3…}
3    //NameNode randomly distribute blocks to slave nodes
4    Let k = int((TotalBlock*replicate)/n)
5    For Each slave node S
6      RBlock [S] := ∅
7      NBlock [S] := ∅
8      RBlock[S] := RBlock[S] ∪ {Br,Br+1,…,Br+k} //r is a random
       num
9    // while there are still blocks at any slave unprocessed
10    Done := False
11    Initialize()
12    Ticks := 0     // Ticks is the time
13    While NOT Done
14      If Ticks equals Update_CR_Now
15        updateReservedBlock()
16      For each slave node S
17        If (not busy S)
18          If RBlock[S]is not empty
19          // Process a block in Reserved Block
20            RBlock[S]:=RBlock[S]\{RoneB[S]}
21          Else
22          If NBlock[S]is not empty
23          // Process a block in Non-reserved Block
24            NBlock[S]:=NBlock[S]\{NoneB[S]}
25          Else
26          // Process non reserved block from Slave T,.and T,.≠S
27            NBlock[T]:=NBlock[T]\{NoneB[T]}
28          End If
29          End If
30        End If
31      End For
32      Make-span=Ticks
33      Ticks=Ticks+1
34    End While
35    Return Make-span
36  End
```

Figure 4.6- 2 AMTS-DC algorithm (Main)

```
Proc updateReservedBlock():
1   Begin
2    //Receive task status of each slave via HeartBeat
3 compute RelCapOfSlave:={c[1],c[2],… ,c[n]} based on task status
4    For Each slave node S where S={1,2,..,n}
5      //Get remaining number of unprocessed blocks: remUB
6      Number of reserved block:=min(c[S]/Sum(cA,cB,….)*remUB
7     Number of non-reserved block:=remUB-Number of reserved block
8      Update the reserved blocks
9      Update the non-reserved blocks
10    End For
11 End
```

Figure 4.6- 3 AMTS-DC algorithm (updateReservedBlock)

```
Proc Initialize():
1  Begin
2    Set {cS1,cS2,… ,cSN} to {1,1,..1}
3  End
```

Figure 4.6- 4 AMTS-DC algorithm (Initialization)

4.5.2 Details analysis of Hadoop FIFO, DDP and AMTS-DC

In Hadoop when data file is written to HDFS, the file is split into data blocks. The default block size is 64MB. The parameter *dfs.replication* whose default value is 3 controls how many copies of a block can be produced. Each copy of a block is known as a replica. The default placement policy of the replica placement, i.e. where each replica is actually placed, is a random process. Considered for example a cluster with n worker nodes (each running TaskTracker and DataNode). The replicated data will be "evenly" distributed among the worker nodes if the number of block is large (i.e the number of blocks is greater than, says, 3 times the number of worker nodes).

As an illustration, consider 3 worker nodes (each running TaskTracker and DataNode) and with a replication set to 2 and there are 39 blocks with identity (ID): 1, 2, 3, 4 ... 39. The blocks are replicated to 1, 1, 2, 2, 3, 3, 4, 4 ... 39, 39.

Total replicated blocks=2*39=78

Each slave will be allocated 26 blocks as calculated from the equation below:

Total replicated block/(total number of slave)=78/3=26

One possible scenario is as follows with each slave being allocated 26 tasks. Note that the same replica will not appear twice within a worker node.

Slave 3: 15 16 23 10 20 4 33 32 18 30 24 11 12 38 27 26 3 35 9 36 17 6 31 2 39 29

Slave 2: 24 25 38 7 37 21 14 20 13 23 17 32 34 5 8 12 22 35 1 9 4 18 15 26 28 19

Slave 1: 6 2 29 3 27 37 33 34 28 7 21 30 16 8 10 25 13 5 11 36 1 39 19 14 22 31

Assume that the relative computing capacity of Slave 3: Slave 2: Slave 1 is 10:2:1 respectively, where 10 is the most powerful and 1 is the least. Therefore, the time to complete each block by slave 3, Slave 2 and Slave 1 will be 10/10: 10/2: 10/1. This is equivalent to the ration 10: 5: 1. To simplify our discussion, and without losing generality, the time required to complete a block by the fastest node is assume to be 1 second. Time taken to process a block by the slaves are as followed:

Slave 3: 1 second

Slave 2: 5 seconds

Slave 1: 10 seconds

Network cost: 6 seconds*

*Note that the network cost 6 seconds is obtained/estimated by measuring the time taken to move a data block from one data-node to another. This is an average value obtained from the prototype setup in Chapter 5. Here we assume a homogeneous network.

4.5.3 Case 1 Hadoop FIFO

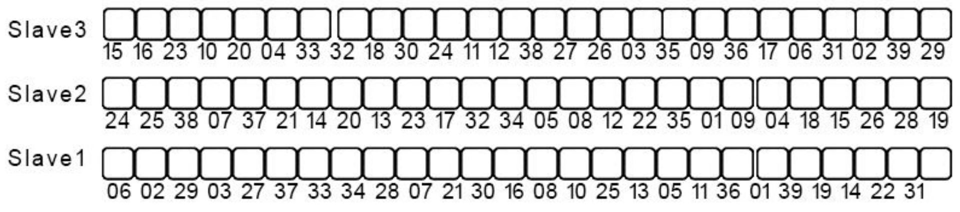


Figure 4.7 Hadoop original Block List

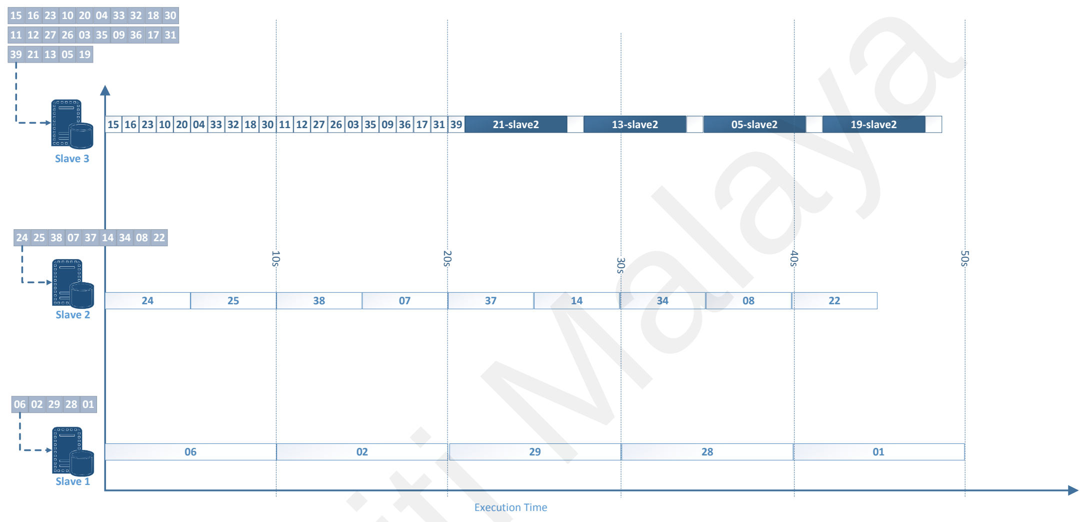


Figure 4.8 Hadoop: Block Processed vs Time

(0)3dq3:15 (0)2dq2:24 (0)1dq1:6 (1)3dq3:16 (2)3dq3:23 (3)3dq3:10 (4)3dq3:20 (5)3dq3:4 (5)2dq2:25 (6)3dq3:33 (7)3dq3:32 (8)3dq3:18 (9)3dq3:30 (10)3dq3:11 (10)2dq2:38 (10)1dq1:2 (11)3dq3:12 (12)3dq3:27 (13)3dq3:26 (14)3dq3:3 (15)3dq3:35 (15)2dq2:7 (16)3dq3:9 (17)3dq3:36 (18)3dq3:17 (19)3dq3:31 (20)3dq3:39 (20)2dq2:37 (20)1dq1:29 **X(21)3dq2:21** (25)2dq2:14 **X(28)3dq2:13** (30)2dq2:34 (30)1dq3:28 **X(35)3dq2:5** (35)2dq2:8 (40)2dq2:22 (40)1dq3:1 **X(42)3dq2:19**

Figure 4.9 Hadoop Block Processed Time

In this section, for simplicity of explanation, the time taken to process one data block by the fastest node is 1 second. From Figure 4.8 and Figure 4.9, it can be seen that at time t=20s, Slave 3 has completed all its local blocks and subsequently needs to move block 21, 13, 5 and 19 from slave 2. There are a total of 4 blocks movement as depicted by the 4 black boxes in Figure 4.8. The make-span can be calculated by adding the time to process a block (10 sec for slave 1) to the time the last block being processed by slave 1. Make-span is therefore equals to 50 seconds (40 + 10 = 50 seconds). The total data movement count is equal to the black boxes in Figure 4.8 or the sections preceded by bold **"X"** (eg **X(21)3dq2:21**) in Figure 4.9. Total data movement is 4.

Make-span=50 seconds

Data movement count = 4;

4.5.4 Case 2 DDP

Slave 3: 6 2 29 3 27 37 33 34 28 7 21 30 16 8 10 25 13 5 11 36 1 39 19 14 22 31 24 38 20
Slave 2: 23 17 32 12 35 9
Slave 1: 4 18 15 26

Figure 4.10 DDP original block List

Note that in DDP, since there is no block replication, only one set of block has been allocated.

(0)3dq3:1 (0)2dq2:9 (0)1dq1:26 (1)3dq3:24 (2)3dq3:32 (3)3dq3:11 (4)3dq3:28 (5)3dq3:29 (5)2dq2:5 (6)3dq3:15 (7)3dq3:35 (8)3dq3:31 (9)3dq3:16 (10)3dq3:3 (10)2dq2:8 (10)1dq1:20 (11)3dq3:33 (12)3dq3:25 (13)3dq3:39 (14)3dq3:4 (15)3dq3:7 (15)2dq2:30 (16)3dq3:19 (17)3dq3:21 (18)3dq3:34 (19)3dq3:13 (20)3dq3:37 (20)2dq2:23 (20)1dq1:2 (21)3dq3:27 (22)3dq3:17 (23)3dq3:10 (24)3dq3:6 (25)3dq3:22 (25)2dq2:18 (26)3dq3:38 (27)3dq3:36 (28)3dq3:12 (29)3dq3:14

Figure 4.11 DDP Block Processed Time

The case for DDP is straight forward as shown in Figure 4.10 and Figure 4.11. The last block processed by Slave 3 is at t=29 second. Therefore the make-span=29 + the time needed to process a block by Slave 3 = 29+1=30 seconds. Note that no data movement has been recorded in the entire make-span. This is because the number of blocks assigned to a compute node is proportional to the computational capacity of the node and all local nodes are processed and completed by the slave at almost the same time. DDP is very efficient and has the shortest make-span.

Make-span=30 seconds

Data movement count = 0;

4.5.5 Case 3 AMTS-DC

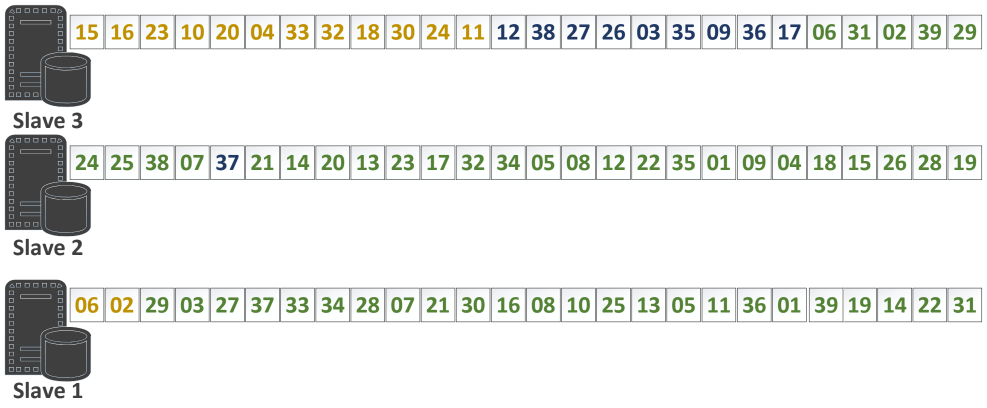


Figure 4.12 AMTS-DC Original Task List

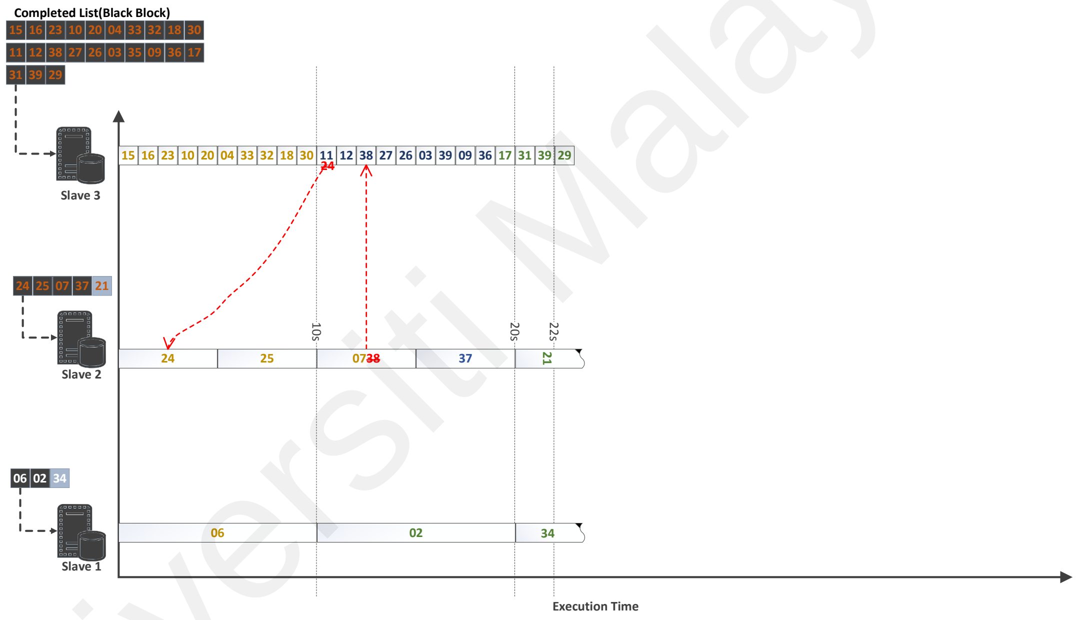


Figure 4.13 Block Processed vs Time (0 ~ 22)

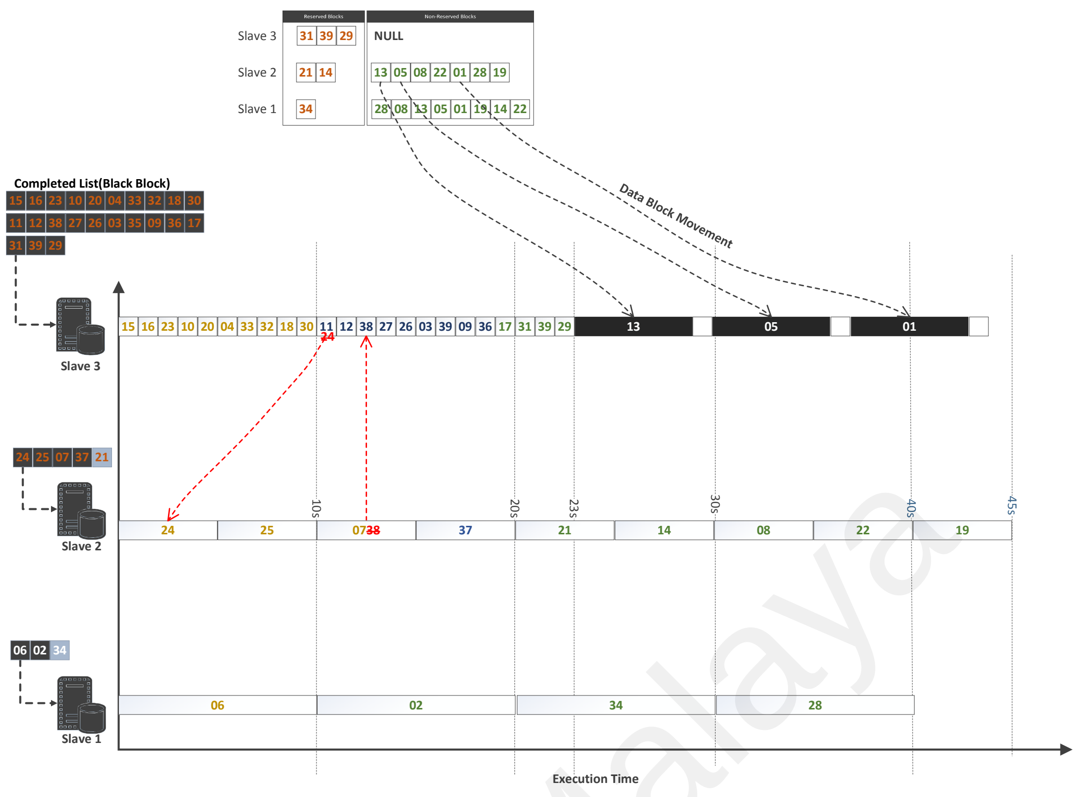


Figure 4.14 Block Processed vs Time (23 ~ 45 second)

(0)3dq3:15 (0)2dq2:24 (0)1dq1:6 (1)3dq3:16 (2)3dq3:23 (3)3dq3:10 (4)3dq3:20 (5)3dq3:4 (5)2dq2:25 (6)3dq3:33 (7)3dq3:32 (8)3dq3:18 (9)3dq3:30 (10)3dq3:11 (10)2dq2:7 (10)1dq1:2 (11)3dq3:12 (12)3dq3:38 (13)3dq3:27 (14)3dq3:26 (15)3dq3:3 (15)2dq2:37 (16)3dq3:35 (17)3dq3:9 (18)3dq3:36 (19)3dq3:17 (20)3dq3:31 (20)2dq2:21 (20)1dq1:34 (21)3dq3:39 (22)3dq3:29 X(23)S3dq2S:13 (25)2dq2:14 X(30)3dq2S:5 (30)2dq2S:8 (30)1dq1S:28 (35)2dq2S:22 X(37)3dq2S:1 (40)2dq2S:19

Figure 4.15 AMTS-DC Block Processed Time

Completed list (Total=26):

15 24 6 16 23 10 20 4 25 33 32 18 30 11 7 2 12 38 27 26 3 37 35 9 36 17

Uncompleted list (Total=13):

31 21 34 39 29 13 14 5 8 28 22 1 19

Figure 4.16 AMTS-DC Completed list and Uncompleted list at time=19 sec

Figure 4.12, 4.13 and 4.14 illustrate the working principle of the proposed AMTS-DC. They are complemented with Figure 4.14 and 4.15 to explain how local blocks can be reserved.

In Figure 4.15 and 4.16, the blocks 15 24 6 … 30 11 7 2 12 …9 36 17 (highlighted in yellow) have been completed before the capacity of slaves could be computed and returned by the heartbeat. These blocks are high-lighted in yellow. In Figure 4.15 the processing of the yellow blocks start at time = 0 seconds. When time equals 19 second, 26 blocks have been completed. Once the capacity of each slaves is known, the uncompleted blocks, block 31 21 34 39 29 13 14 5 8 28 22 1 19 (highlighted in green) will be marked and reserved in proportion based on the computation capacity of the slaves as indicated in Figure 4.17.

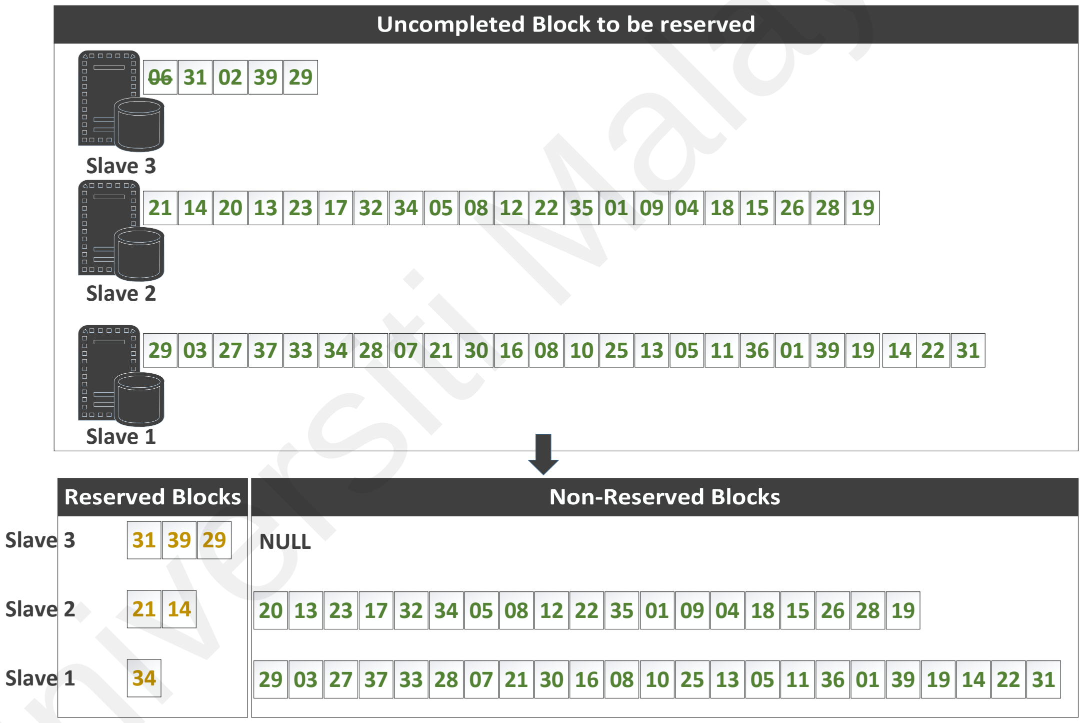


Figure 4.17 Blocks which are reserved after the heartbeat

Figure 4.13 and 4.14 depict the block processed at time period 0 to 22 second and 23 to 45 second. Figure 4.17 depicts the blocks which are being reserved after the heartbeat. The total blocks reserved by all slave is 6 (i.e. 3+2+1) as indicated as follows:

Slave 3 reserved: 31 39 29

Slave 2 reserved: 21, 14

Slave 1 reserved: 34

Blocks unreserved for each slave are as follows:

Slave 3 Unreserved: NULL*

Slave 2 Unreserved: 13, 5, 8, 28, 19

Slave 1 Unreserved: 3, 8, 1, 19, 22

* Note: Note referring to Figure 4.18, Slave 3 has unreserved blocks 6 and 2 (black font with green background). However, since Slave 1 had already processed both blocks 6 and 2 earlier, the total number of unreserved block for Slave 1 is thus NULL.

Slave 3: 15 16 23 10 20 4 33 32 18 30 24 11 12 38 27 26 3 35 9 36 17 6 31 2 39 29

Slave 2: 24 25 38 7 37 21 14 20 13 23 17 32 34 5 8 12 22 35 1 9 4 18 15 26 28 19

Slave 1: 6 2 29 3 27 37 33 34 28 7 21 30 16 8 10 25 13 5 11 36 1 39 19 14 22 31

Figure 4.18 AMTS-DC (Reserved and Unreserved Blocks)

(0)3dq3:15 (0)2dq2:24 (0)1dq1:6 (1)3dq3:16 (2)3dq3:23 (3)3dq3:10 (4)3dq3:20 (5)3dq3:4 (5)2dq2:25 (6)3dq3:33 (7)3dq3:32 (8)3dq3:18 (9)3dq3:30 (10)3dq3:11 (10)2dq2:7 (10)1dq1:2 (11)3dq3:12 (12)3dq3:38 (13)3dq3:27 (14)3dq3:26 (15)3dq3:3 (15)2dq2:37 (16)3dq3:35 (17)3dq3:9 (18)3dq3:36 (19)3dq3:17 (20)3dq3:31 (20)2dq2:21 (20)1dq1:34 (21)3dq3:39 (22)3dq3:29 X(23)S3dq2S:13 (25)2dq2:14 X(30)3dq2S:5 (30)2dq2S:8 (30)1dq1S:28 (35)2dq2S:22 X(37)3dq2S:1 (40)2dq2S:19

Figure 4.19 AMTS-DC Block Computation

Figure 4.18 is a snap-shot of Figure 4.12, after the yellow and blue blocks yellow in Figure 4.12 have been completed. Observed that in Figure 4.18, the number in white font within the green region are the reserved blocks. Number in black font within the green region are unreserved blocks. Block numbers in white font have the higher priority to be executed by the slave itself. For instance, blocks 31, 39 and 29 have higher priority of being executed by slave 3. Similarly blocks 21, 14 by Slave 2 and blocks 34 by Slave 1. In Slave 3, by reserving blocks 31, 39, 29 to itself, Slave 3 as the fastest node will have all the remaining local blocks reserved to it. The execution of these local blocks by Slave 3 will not incur additional network cost and hence shorten the make-span. Observe that the highlighted section in Figure 4.18 and Figure 4.19, there is a co-relation between sections of the same color. Observed in Figure 4.19 that data movement (i.e. copying of a block from one slave to another) is highlighted in red. More data movement implies longer make-span. Only 3 block-movements are recorded and the make-span of AMTS-DC is 40+5 seconds=45 seconds. The reservation mechanism in AMTS-DC managed to improve the amount of local blocks executed by the faster node. This is not the case for Hadoop FIFO. Hadoop FIFO has four block-movements and a make-span of 50 seconds.

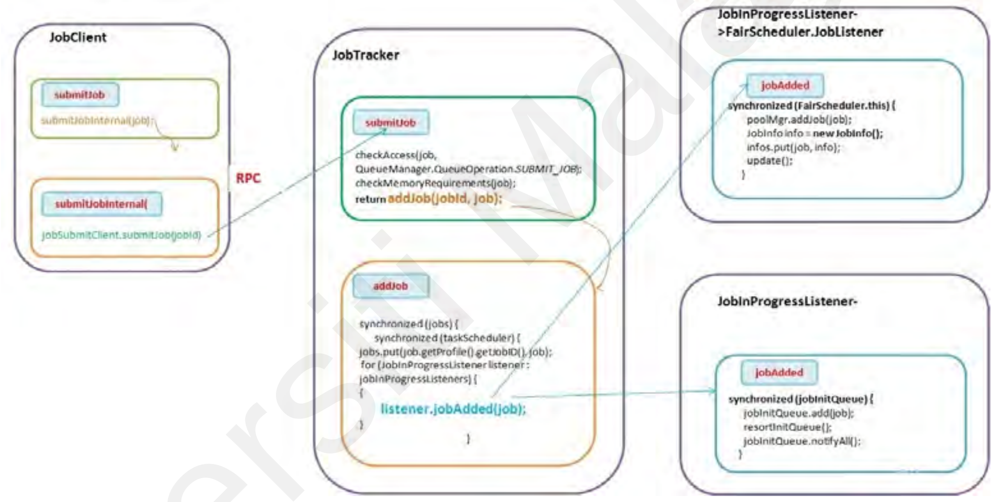
Summary

In this section the task scheduler of Hadoop FIFO and DDP are analyzed to identify areas where task scheduling could be further improved. Based on the finding a novel task scheduler namely, Adaptive MapReduce Task Scheduler Using Dynamic Calibration (AMTS-DC), is proposed.

CHAPTER 5: EXPERIMENTAL AND SIMULATION RESULTS

In order to evaluate the performance of the proposed Adaptive MapReduce task scheduler with dynamic calibration (AMTS-DC) mechanism proposed in chapter 4, experiments are conducted using prototype and simulator. This chapter presents detailed experimental setup, results and analysis of the performance of the proposed AMTS-DC against Hadoop FIFO and DDP. Experiments are carried by varying job size and the number of computer nodes. It is then followed by the discussion and analysis of the experimental results.

5.1 Prototype and Simulation Environment



```
submitJobInternal(job);
```

```
jobSubmitClient.submitJob(jobId)
```

```
checkAccess(job,
QueueManager.QueueOperation.SUBMIT_JOB);
checkMemoryRequirements(job);
return addJob(jobId, job);
```

```
synchronized (jobs) {
    synchronized (taskScheduler) {
jobs.put(job.getProfile().getJobID(), job);
 for (JobInProgressListener listener :
jobInProgressListeners) {
{
   listener.jobAdded(job);
}
                    }
```

```
synchronized (FairScheduler.this) {
    poolMgr.addJob(job);
    JobInfo info = new JobInfo();
    infos.put(job, info);
    update();
   }
```

```
synchronized (jobInitQueue) {
   jobInitQueue.add(job);
   resortInitQueue();
   jobInitQueue.notifyAll();
  }
```

Figure 5.1 Hadoop JobTracker (“Hadoop Job Submission Initialization”)

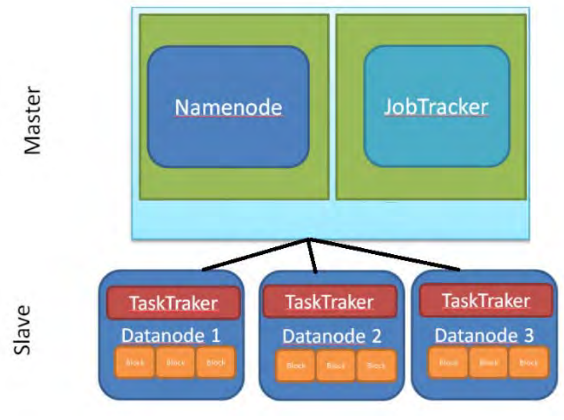


Figure 5.2 JobTracker, TaskTracker and DataNode

The prototype is developed using Hadoop version 1.2.1 framework. We use VMware ESXi server 5.5 and the compute/date nodes are virtual machines running Ubuntu 14.10 desktop. Figure 5.1 depicts the relationship between the JobClient and the JobTracker. The relationship between master and slave node is shown in Figure 5.2 and Figure 5.3 depicts the actual prototype cluster.

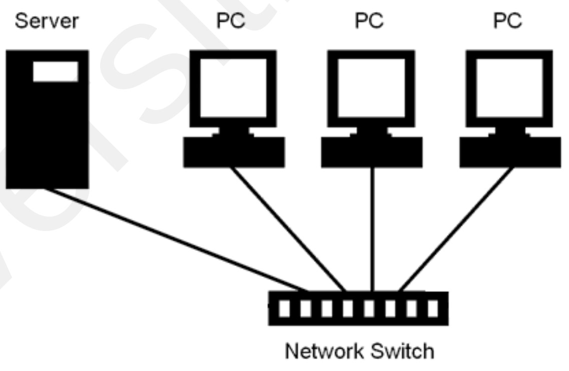


Figure 5.3 Prototype Cluster

Table 5.1 Hardware Configuration

	Machine	CPU	RAM	Disk
Host1	HP Compaq Elite 8300 SFF	4 CPUs * 3392 GHZ	12 GB	1 TB
Host2	HP EliteDesk 800 G1 TWR	4 CPUs * 3392 GHZ	16 GB	1 TB
Host3	HP Compaq Elite 8300 CMT	4 CPUs * 3392 GHZ	20 GB	1 TB
Host4	Intel Core Quad CPU Q9400	4 CPUs * 2659 GHZ	6 GB	1 TB

The experiment environment consists of a computer cluster containing one Manager (Jobtracker) and three workers (TaskTracker) as out-lined in Table 5.1.

Table 5.2 VM Configuration

	CPU(Hz)	RAM(Byte)	Network	Replication
Master	5120	3072	100 Mbps	2*
Slave 1	900	3072		
Slave 2	1024	3072		
Slave 3	5120	3072		

*Note: Replication for Hadoop FIFO and AMTS-DC

In Table 5.2, the CPU configuration of the Master node is set to the CPU speed of 5.12 GHz. In order to test the effect of local task and non-local task, the number of replication should be always less than the number of slaves. In this prototype, the number of slave is 3 and therefore the replication is set to 2.

A simulator, namely AMST-Sim, is developed to evaluate the proposed task scheduler. AMST-Simulator is needed since the prototype is unable to handle more complex scenario with greater number of compute nodes and with larger input data size.

5.2 Performance Evaluation Metrics.

In this section the performance of Hadoop default scheduler, DDP and the proposed AMTS-DC schedulers are evaluated using the *wordcount* Hadoop application with block size of 64 Mbyte. Data files feeding into the application are created by merging the text files downloaded from https://www.gutenberg.org.

Experiment 1: Static environment

All compute nodes are dedicated nodes and no other programs/processes are being run apart from the map-reduced jobs assigned to them.

Experiment 1a: Evaluating Hadoop FIFO scheduler

Using the hardware and the virtual machine configuration indicated in Table 5.1 and 5.2, the make-span of jobs with different data size is recorded

Experiment 1b: Evaluating DDP scheduler

Step1: Historical record of the relative computing speed, i.e. the ratio, of the compute-nodes are benchmarked by running map-reduce jobs and record the make-span of the job and the time taken by the node to complete the tasks assigned to the node.

Step2： Based on the ratio captured in the historical record, appropriate number of data blocks is assigned to each compute node based on this ratio. More powerful nodes get more blocks.

Experiment 1c: Evaluating AMTS-DC scheduler

Step1: Jobs assigned to the compute nodes are executed and heartbeat are transmitted between the Job Tracker and the Task Tracker. In Hadoop, heartbeats are send to the job tracker carrying with them information such as task status, task counters, and data read/write etc.

Step2： Based on the dynamic information captured from the heartbeat a ratio expressing the relative computing power of each node is computed. The local blocks within each compute node are re-assigned based on this ratio. More powerful nodes will get more local blocks reserved.

Experiment 2: Dynamic environment

Some of the compute nodes are non-dedicated nodes. Apart from running the map-reduced jobs assigned to them, other programs/processes as well might be running in the nodes.

5.3 Prototype Experimental Results and AMST-Sim Calibration

Experiment 1: Static environment

Table 5.3 Hadoop FIFO

	CPU (Hz)	data size (MB)	Number of Block allocated	Blocks completed	Average total Completion time (Make-span)
Slave1	900	900	40	10	888 sec
Slave2	1024	1024	40	13	
Slave3	5120	3092	40	37	
Total Block			120	60*	

* since data replication is set to 2, the number of block to be completed is half of the total block allocated, i.e., 120/2=60 blocks.

Table 5.4 DDP

	CPU(Hz)	Data size (MB)	Historical record ratio	Number of Block allocated	Blocks completed	Average total Completion time (Make-span)
Slave1	900	900MB	0.15	9	9	690sec
Slave2	1024	1024MB	0.18	11	11	
Slave3	5120	3092MB	0.67	40	40	
Total Block				60*	60	

Table 5.2, 5.3 and 5.4 provide details of the static environment experiment. In Table 5.4, the job to be processed by DDP is directly assigned to the compute nodes. Only one set of data and no data replication is involved.

Table 5.5 AMTS-DC

	CPU (Hz)	Memory (MB)	Blocks Allocated	Blocks completed	The total Completion time (Make-span)
Slave1	900	900MB	40	9	757 sec
Slave2	1024	1024MB	40	13	
Slave3	5120	3092MB	40	38	
Total Block			120	60	

In Table 5.5, note that since data replication is set to 2, the number of block to be completed is half of the total block allocated, i.e. 120/2=60 blocks. Block allocation is dynamically handled by HDFS

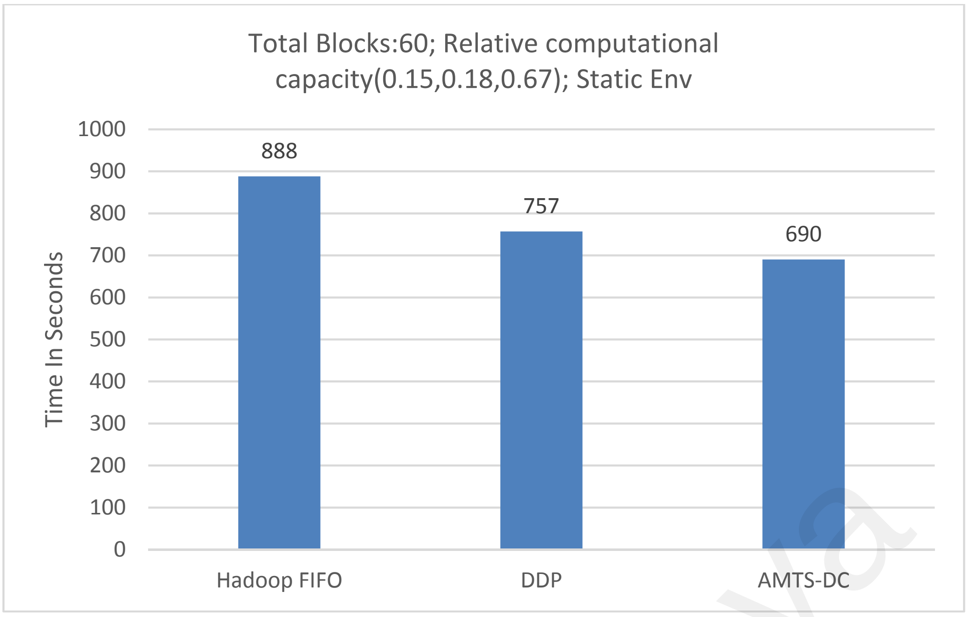


Figure 5.4- 1 Make-span of Hadoop FIFO, DDP and AMTS-DC (Static Env)

Figure 5.4-1 is derived from the experimental result obtained in Table 5.3, 5.4 and 5.5. The relative computational capacity (0.15, 0.18, 0.67) is obtained from the historical run time. The improvement of DDP make-span over Hadoop make-span is 22%. The improvement of AMTS-DC make-span over Hadoop make-span is 15%. In this experiment, DDP has the best performance with the shortest make-span. This is followed by AMTS-DC and Hadoop FIFO. DDP is able to outperform AMTS-DC because it is able to optimize the overall block-ratio (in this case all the 60 blocks) to be assigned to each compute node; while AMTS-DC on the other hand, only optimized part of the local blocks (in this case about 20 blocks) within each compute node.

AMTS-DC is able to outperform Hadoop FIFO because AMTS-DC is able to reserve some of the local tasks within the faster node. By doing so AMTS-DC is also able to cut down the number of data block to be transferred from one compute node to the other. In this experiment, Slave 3 of AMTS-DC and Hadoop FIFO only manage to execute 38 and 39 out of the 40 local tasks respectively. Some local tasks in Slave3 have been replicated in Slave1 or Slave2 and these tasks had been executed on the slower nodes (either Slave1 or Slave2) instead of by Slave3. DDP is able to complete all 40 local blocks using the fastest node (slave3) while AMTS-DC and Hadoop FIFO only managed to finish 38 and 37 blocks respectively using the fastest node.

In this experiment when the environment is static, DDP outperforms AMTS-DC. This may not be the case if the computing resources are dynamic as it will be shown in the next experiment.

Experiment 2: Dynamic Environment

In this experiment, the computation environment of Slave3 is dynamic. In other words, during the MapReduce job other processes are invoked and resulted in the reduction in the computation power of Slave3.

Table 5.6 Hadoop (dynamic environment)

Hadoop	CPU(Hz)	Data Size (MB)	Number of Block allocated	Status	Average total Completion time (Make-span)
Slave1	900	900	40	Static	950sec
Slave2	1024	1024	40	Static	
Slave3	5120	3092	40	dynamic	

Table 5.7 DDP (dynamic environment)

DDP	CPU(Hz)	Data Size (MB)	Historical record ratio*	Number of Block allocated	Status	Average total Completion time (Make-span)
Slave1	900	900MB	0.15	9	Static	1010sec
Slave2	1024	1024MB	0.18	11	Static	
Slave3	5120	3092MB	0.67	40	dynamic	

*Relative computational capacity ratio

Table 5.8 AMTS-DC (dynamic environment)

AMTS-DC	CPU(Hz)	Data Size (MB)	Number of Block allocated	Status	Average total Completion time (Make-span)
Slave1	900	900	40	Static	882 sec
Slave2	1024	1024	40	Static	
Slave3	5120	3092	40	dynamic	

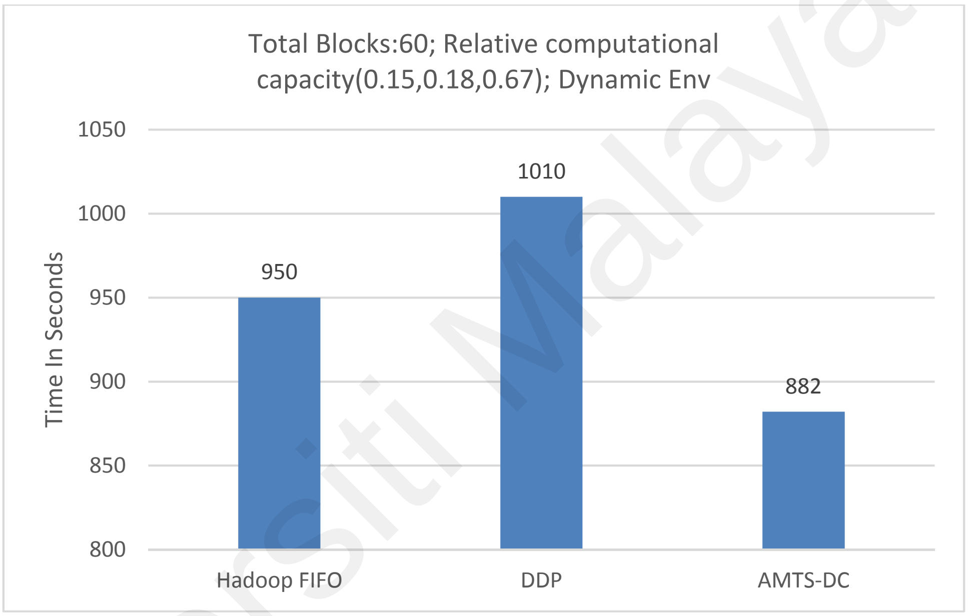


Figure 5.4- 2 Make-span of Hadoop FIFO, DDP and AMTS-DC (Dyn Env)

Figure 5.4-2 is derived from the results obtained in Table 5.6, 5.7 and 5.8. In this experiment, DDP has the worst performance with the longest make-span. This is followed by Hadoop FIFO and AMTS-DC. DDP is unable to perform although the block size to be allocated is proportion to the static computation power of each node. Since the status of Slave3 is dynamic, the computational power of Slave3 will vary from time to time. DDP should be able to perform well should the number of block to be allocated is proportion to the dynamic ratio of the computation power of each node instead of the static ratio as the case for AMTS-DC. AMTS-DC has the best performance in this experiment.

Since the prototype is unable to handle more complex scenario with greater number of compute nodes and with larger input data size. AMST-Sim is developed so that more complex scenario could be used to test the hypothesis. AMST-Sim is a discrete event simulator developed using Java programming language. The high-level pseudocode of the main method of AMST-Sim is as follows:

```
while (moreTask) {
        int n;
        for (int i = 1; i < numberOfQueue; i++) {
           if (ticks == fireNextQ[i]) {
              if (!q[i].isEmpty()) {
                 n = q[i].dequeue();
                 fireNextQ[i] += fire[i];  // process one data block}}}
        ticks++;
        if ( Every Queue is isEmpty) {
           moreTask = false;}}
```

Figure 5.4 AMST-Sim (Pseudocode)

In order to improve the accuracy of AMST-Sim, the simulator is calibrated using the output from the prototype. Measurements obtained from prototype experiment 1 is used to configure the simulator parameters such as make-span, network cost, value of relative computing capacity and so on. Table 5.9-1 and 5.9-2 provide details of the simulation configuration.

Table 5.9- 1 Nodes and Replication

Compute-Nodes (Slaves)	Replication	Task size (64MB/Block)
3	2	60Blocks

Table 5.9- 2 Relative computational capacity of nodes

Compute Node	Relative computation power
Slave1	1 (slowest)
Slave2	1.2

Slave3	4.8 (fastest)

Table 5.9- 3 Results obtained from Prototype Vs Results obtained from Simulation

Scheduler	Prototype Average Completion time (Make-span) seconds	Simulation Average Completion time (Make-span) seconds	Difference (%)
Hadoop FIFO	888	882	1%
AMTS-DC	757	749	1%
DDP	690	699	-1%

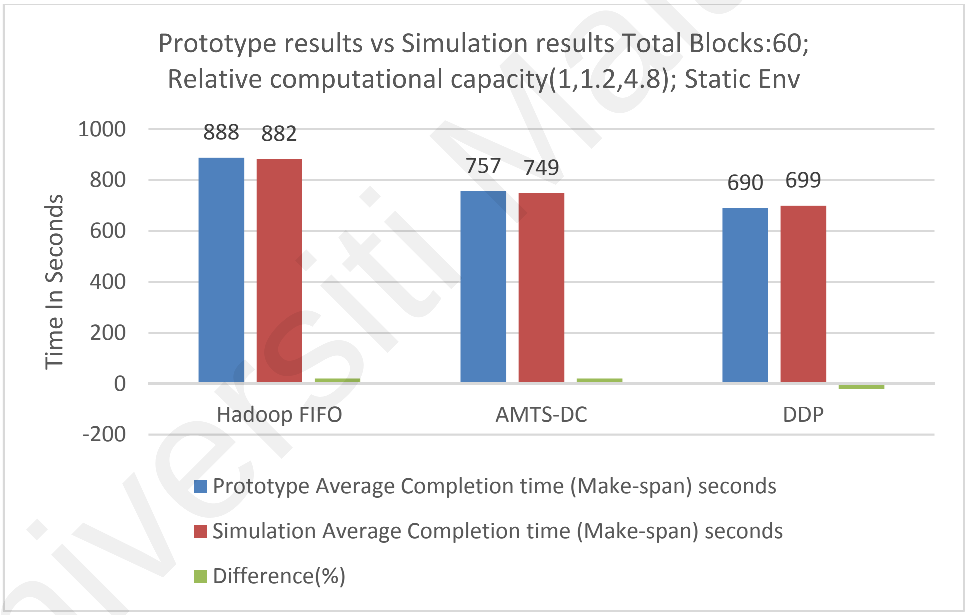


Figure 5.5 Comparison of results from Prototype and Simulation

As indicated in Table 5.9-3 and depicted in Figure 5.6, within the experiment error, the make-span obtained from the simulation is very close to that from the prototype with an overall error of -1% to 1%. For most cases, the make-span produced by the simulator is very close to the make-span obtained from the prototype experiments. Note that due to the used of a very small data size (only 60 blocks) in the calibration, the relative performance of the task schedulers may not reflects their relative strength.

5.4 Simulation results of Hadoop, DDP and AMTS-DC

Experiment 3

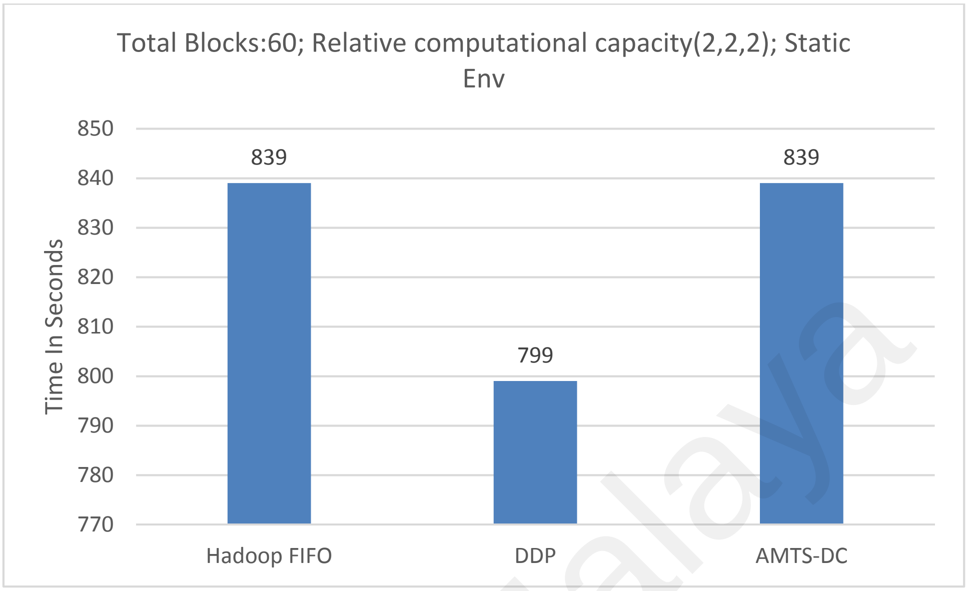


Figure 5.6 Make-span of Hadoop FIFO, DDP and AMTS-DC (Static Env)

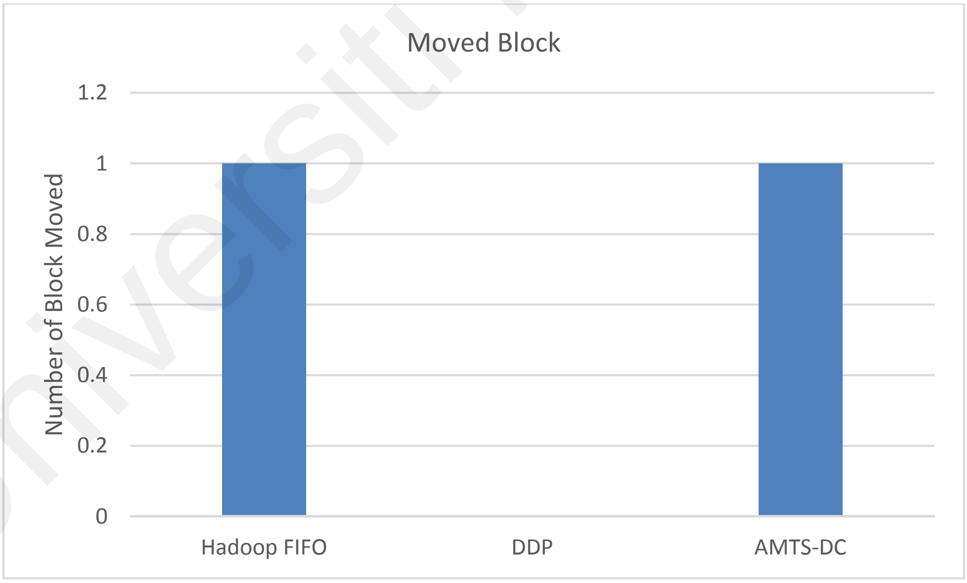


Figure 5.7 Block movement of Hadoop FIFO, DDP and AMTS-DC (Static Env)

In this experiment, all node are set to the same computational power of 2. As expected, as shown in Figure 5.7, all schedulers almost have similar make-span of around 800 (799 to 839) seconds. As shown in Figure 5.8, there is one block movement for Hadoop FIFO and AMTS-DC. This accounts for why both Hadoop FIFO and AMTS-DC has longer make-span of 839 seconds. There is no block movement for DDP. This is expected as DDP has optimized the data placement of all blocks.

Experiment 4 & 5

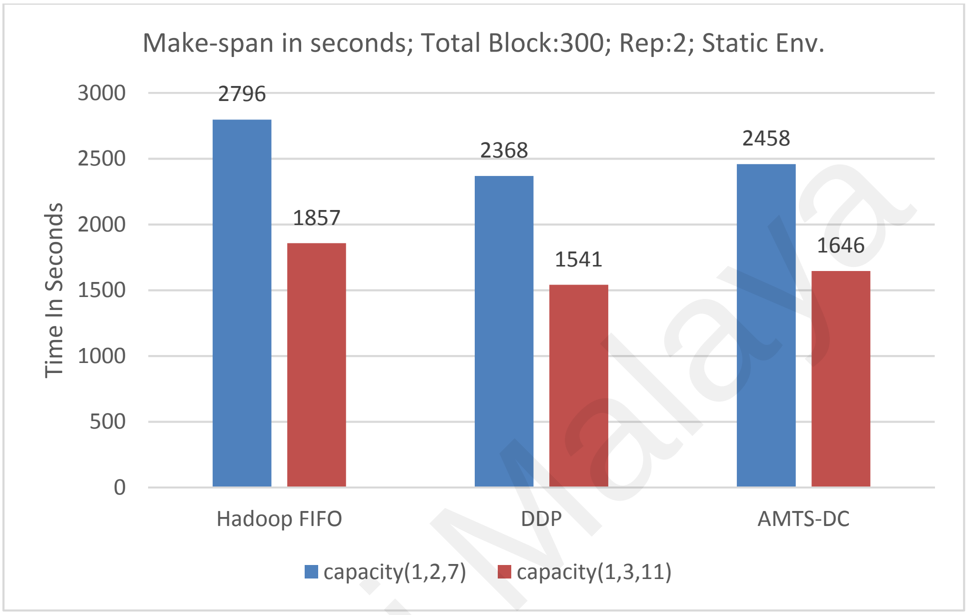


Figure 5.8 Make-span of Hadoop FIFO, DDP and AMTS-DC in different Capacity (Static Env)

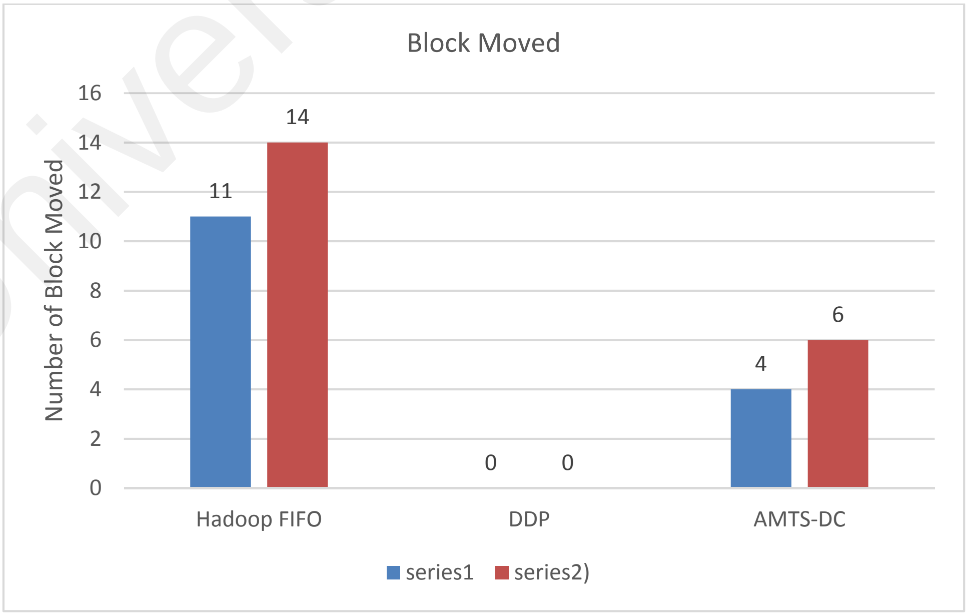


Figure 5.9 Block movement of Hadoop FIFO, DDP and AMTS-DC with capacity setting in Figure 4.8 (Static Env)

In these experiment, all nodes have different computation power. The computing power of slave 1, slave 2 and slave 3 are 1, 2, 7 respectively. It can be seen that Hadoop has the longest make-span and most data movement of 11. This is followed by AMTS-DC with data movement of 4. DDP remains the best with the shortest make-span and no data movement between the compute nodes. In experiment 5, the different in computational power among the nodes is even greater. The different in the computing power of slave 1, slave 2 and slave 3 are 1, 3, and 11 respectively. It can be seen that Hadoop again has the longest make-span of 1857 seconds and with the most data movement (14 blocks are moved). This is followed by AMTS-DC with shorter make-span of 1646 seconds (only 6 blocks are moved). DDP remains the best with the shortest make-span and no data movement between the compute nodes.

Experiment 6 & 7

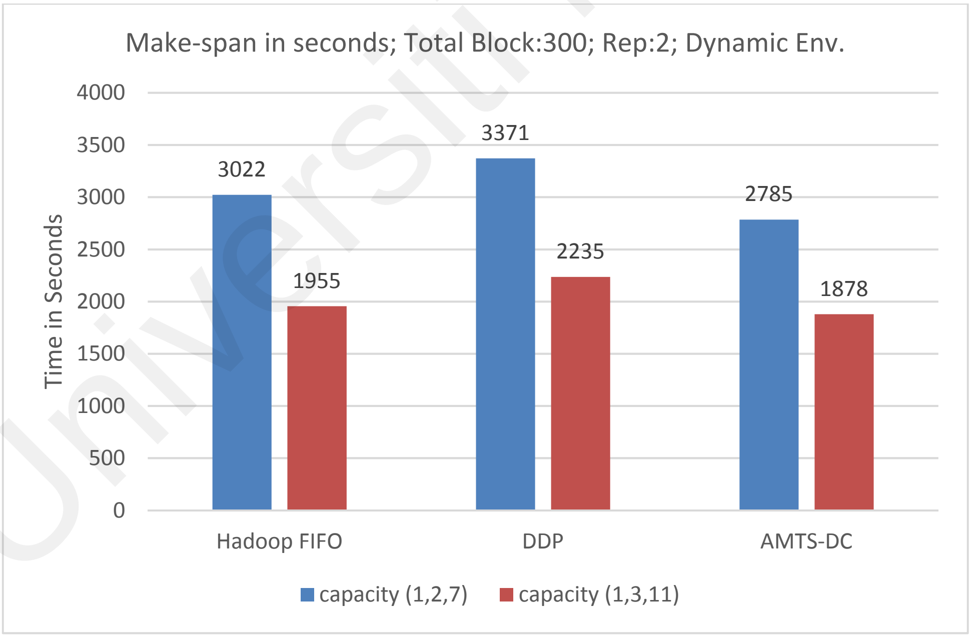


Figure 5.10 Make-span of Hadoop FIFO, DDP and AMTS-DC with Different Capacity (Dynamic Env)

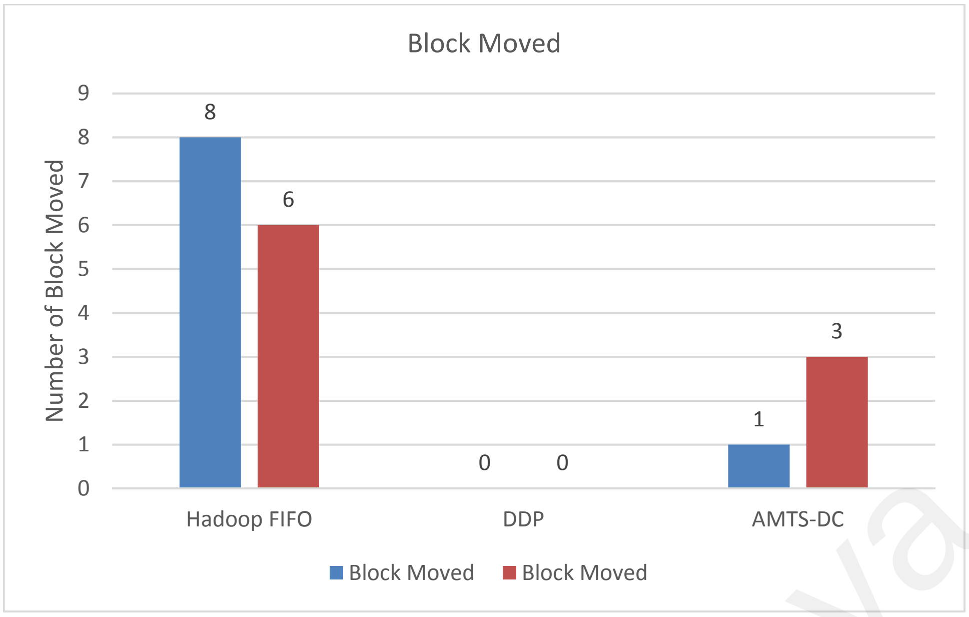


Figure 5.11 Block movement of Hadoop FIFO, DDP and AMTS-DC with Different Capacity (Dynamic Env)

Experiment 6 is similar to experiment 4 except that the computational power is dynamic. The speed of the fastest node is changed in the middle of computation. The speed is reduced from 7 to 6, i.e. by a small factor of 1/7. The make-span for all schedulers has increased. But it can be seen that now DPP Hadoop has the worst performance with the longest make-span of 3371 seconds and with no data movement (data are not moved in DDP, the replication number for DDP is 1). This means for DDP, the remaining tasks has to be completed by the now “slower” fastest node (slave 3) alone. This performance is 10% worse than that of Hadoop FIFO. This extended the DDP make-span. This is followed by Hadoop. The performance of the proposed AMTS-DC is the best with the shortest make-span and only one data movement between the compute nodes.

Experiment 7 is similar to experiment 5 except that the computational power is dynamic. The speed of the fastest node is reduced in the middle of the computation. The speed is reduced from 11 to 10, i.e. by a smaller factor of 1/11. It can be seen that now DPP again has the longest make-span of 2235 seconds and with no data movement (data are not moved in DDP, since the replication number for DDP is 1). This means for DDP, the remaining tasks has to be completed by the now “slower” fastest node (slave 3) alone. This extended the DDP make-span. This is followed by Hadoop. The performance of AMTS-DC is the best with the shortest make-span and only 3 data movement between the compute nodes.

Summary

In this section the performance of Hadoop FIFO, DDP and AMTS-DC are compared and analyzed. In homogeneous environment all the task schedulers (i.e. Hadoop FIFO, DDP and AMTS-DC), within experimental error, has the equal performance. In static heterogeneous environment, the performance of the scheduler, in descending order, are DDP, the proposed AMTS-DC and Hadoop FIFO. Lastly in dynamic heterogeneous environment where computing capacity of node varies with time, the proposed AMTS-DC has the best performance followed by Hadoop FIFO. DDP tailed the list as it is unable to adapt to dynamic changing environment.

CHAPTER 6: ENHANCED ADAPTIVE MAPREDUCE TASK SCHEDULER USING DYNAMIC CALIBRATION (EAMTS-DC)

This chapter discusses the limitations faced by the AMTS-DC mechanism proposed in chapter 4 and the motivation to enhance AMTS-DC.

6.1 Limitation of AMTS-DC

For ease of discussion, Figure 4.12, 4.15 and 4.16 are repeated here as Figure 6.1-1, 6.1-2 and 6.1-3.

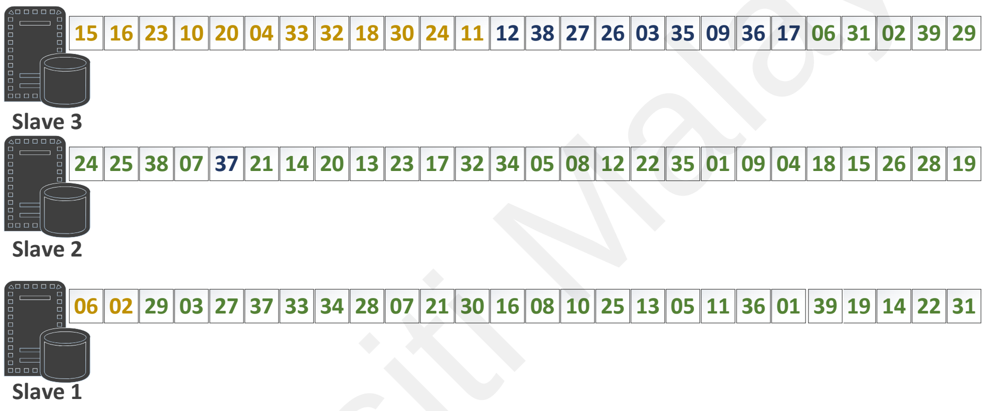


Figure 6.1- 1 AMTS-DC Original Task List

(0)3dq3:15 (0)2dq2:24 (0)1dq1:6 (1)3dq3:16 (2)3dq3:23 (3)3dq3:10 (4)3dq3:20 (5)3dq3:4 (5)2dq2:25 (6)3dq3:33 (7)3dq3:32 (8)3dq3:18 (9)3dq3:30 (10)3dq3:11 (10)2dq2:7 (10)1dq1:2 (11)3dq3:12 (12)3dq3:38 (13)3dq3:27 (14)3dq3:26 (15)3dq3:3 (15)2dq2:37 (16)3dq3:35 (17)3dq3:9 (18)3dq3:36 (19)3dq3:17 (20)3dq3:31 (20)2dq2:21 (20)1dq1:34 (21)3dq3:39 (22)3dq3:29 X(23)S3dq2S:13 (25)2dq2:14 X(30)3dq2S:5 (30)2dq2S:8 (30)1dq1S:28 (35)2dq2S:22 X(37)3dq2S:1 (40)2dq2S:19

Figure 6.1- 2 AMTS-DC Block Processed Time

Completed list (Total=26):

15 24 6 16 23 10 20 4 25 33 32 18 30 11 7 2 12 38 27 26 3 37 35 9 36 17

Uncompleted list (Total=13):

31 21 34 39 29 13 14 5 8 28 22 1 19

Figure 6.1- 3 AMTS-DC Completed list and Uncompleted list at time=19 sec

Figure 6.1- 1, 6.1- 2 and 6.1- 3, indicate that some local blocks have to be processed before any local blocks can be reserved. After the value relative computation capacity of the nodes are returned via the heartbeat, only 13 blocks (these blocks are highlighted in green) are left to be processed as depicted in Figure 6.1-3.

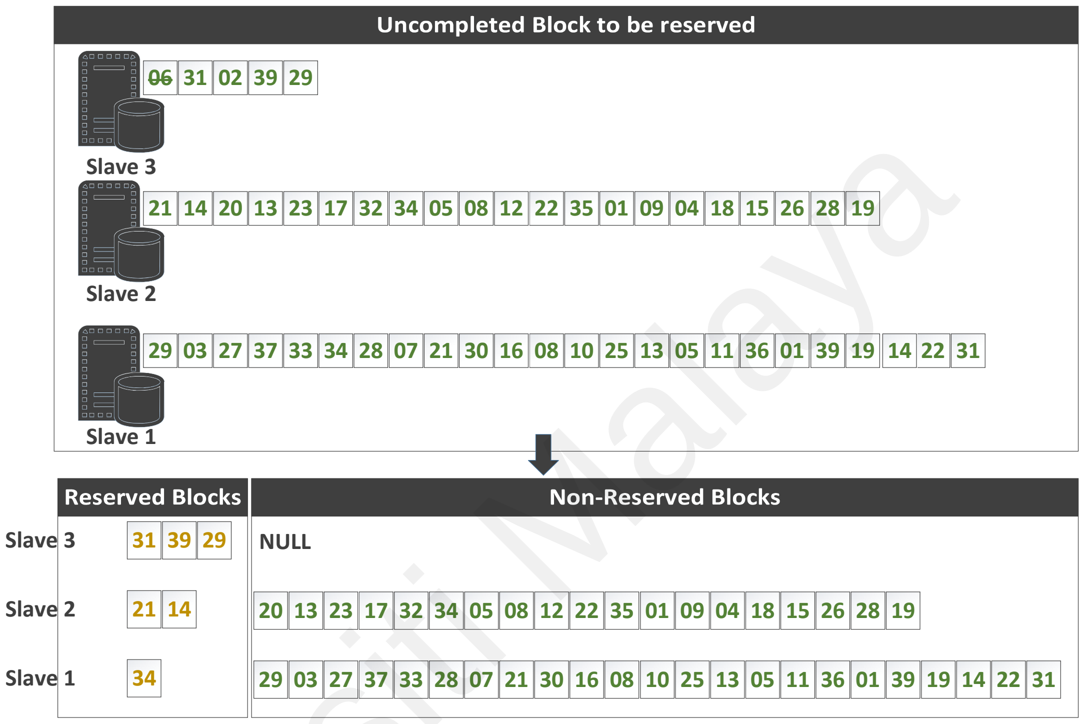


Figure 6.1- 4 Blocks that are reserved by AMTS-DC

The uncompleted list are the candidates which could be reserved are depicted in Figure 6.1-4. It can be seen that the number of local block that could be reserved is very limited. In this case the total number of local block which can be reserved is only 6 (brown color in Figure 6.1-4).

6.2 Motivation to enhance AMT-DC

Supposing that the relative capacity of the compute nodes are known prior to the processing of any local blocks, then all local blocks are potential candidates that could be reserved by the compute node. A possible scenario is depicted in Figure 6.2 where the brown blocks are reserved blocks.

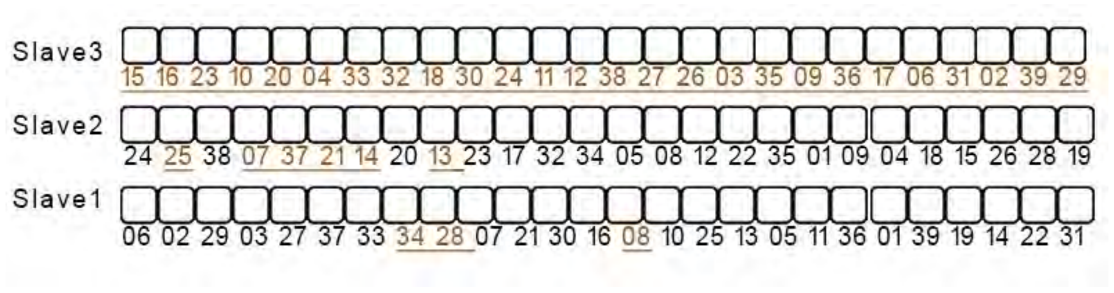


Figure 6.1 More blocks could be reserved if historical run record is used

6.3 Proposed Enhanced AMTS-DC (EAMTS-DC)

Supposing historical record of the relative computing capacity of the compute-nodes are known. The historical value can be used at the initialization stage, prior to the processing of the blocks.

```
Proc Initialize():
Begin
  If Not HisRecExist  // No history, set all to 1
    Set {cS1,cS2,… ,cSN} to {1,1,..1}
  Else
   //Obtain from history the value of c[1],c[2],… ,c[N]
    RelCapOfSlave:={c[1],c[2],… ,c[N]}
    For Each slave node S where S:={1,2,..,n}
      //Get remaining unprocessed local blocks: remUB
         Number of reserved block :=min(c[S]/Sum(c[1],c[2],… ,c[N])*remUB
       Number of non-reserved block:=remUB-Number of reserved block
      Update reserved blocks
      Update non-reserved blocks
     End For
   End IF
 End
```

Figure 6.3- 1 EAMTS-DC Algorithm (Initialization)

Figure 6.3-1 depicts the Initial Data Allocation algorithm of EAMTS-DC. The rest of the EAMTS-DC is similar to the AMTS-DC Algorithm depicted in Figure 4.6-2, 4.6-3 and 4.6-4. For any job, if the historical record of the run time capacity of the nodes are known, then the record will be used at the initial stage of the job being executed, otherwise a general weightage of 1 will be used. A possible scenario is depicted in Figure 6.2 above where the brown blocks are reserved blocks.

Details analysis of EAMTS-DC

A more detail analysis of EAMTS-DC is provided using the example below.

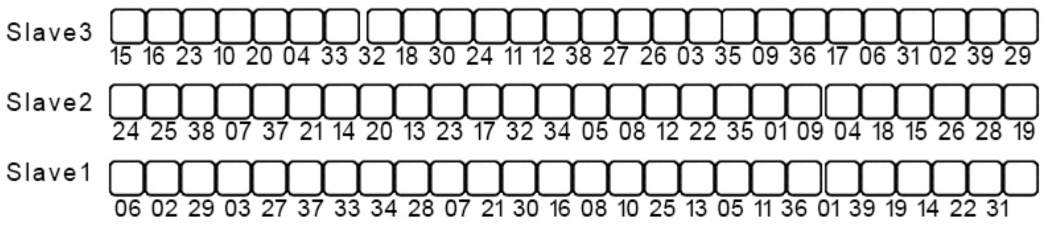


Figure 6.3- 2 EAMTS-DC Original Task List

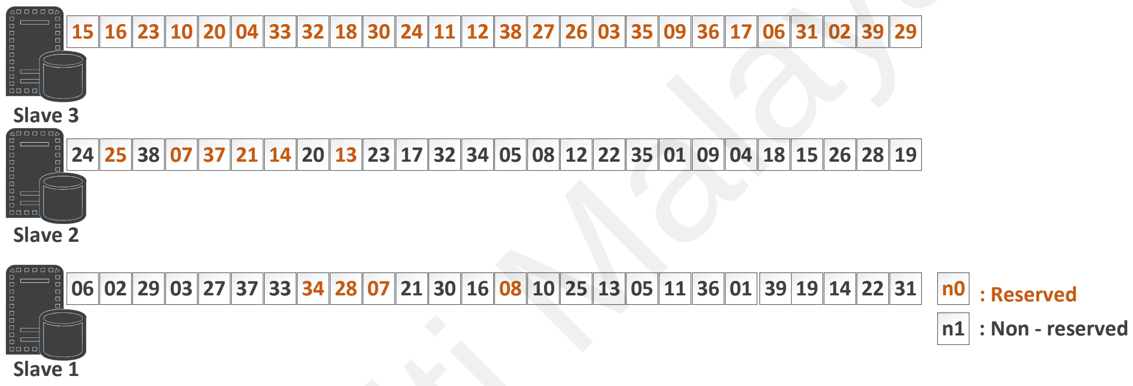


Figure 6.3- 3 EAMTS-DC reserved list

Since the reservation of EAMTS-DC starts from the very beginning by assigning proportional amount of local task to each slave, the number of local task reserved will be higher for each slave. By applying the initial data allocation algorithm to the data blocks in Figure 6.3-2, we get the reserved blocks which are indicated in brown in Figure 6.3-3. The number of block reserved are 26, 6 and 4 for Slave 1, Slave 2 and Slave 3 respectively. 35 blocks out of a total of 39 blocks are reserved. DDP as a contrast, are able to "reserve" all the 39 blocks. In theory the maximum number of block which could be reserved by Slave 3 is equal to (capacity of Slave 3)/ (total capacity)*(total blocks)=10/(1+2+10)*total blocks= 10/13*39=30 blocks. If Slave 3 is able to reserve 30 blocks, the total number of block that could be reserved by EAMTS-DC will be 39 (i.e. 30+6+3). However, since the number of local task for Slave 3 is only 26, only 26 blocks could be reserved for Slave 3. DDP is able to out-perform EAMTS-DC in static environment since it could reserve the maximum number of blocks (i.e. all the blocks).

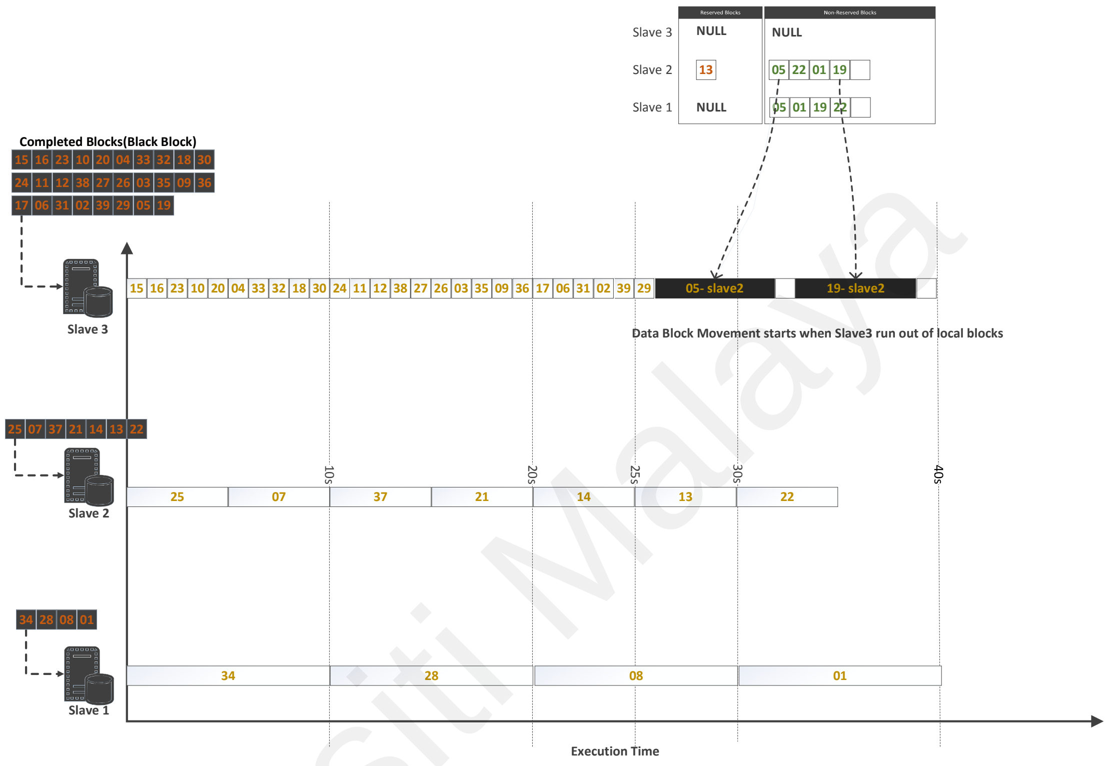


Figure 6.3- 4 EAMTS-DC- Reserved block list at time 25 second

Figure 6.3- 4 shows the situation at time 25 second. At this time Slave 3 which is the fastest node, runs out of data block. Data block 05 and 19 are then moved from Slave 2 to Slave 3. Slave 3 will then process data block 05 and 19 for processing.

(0)3dq3:15 (0)2dq3:25 (0)1dq1:34 (1)3dq3:16 (2)3dq3:23 (3)3dq3:10 (4)3dq3:20 (5)3dq3:4 (5)2dq3:7 (6)3dq3:33 (7)3dq3:32 (8)3dq3:18 (9)3dq3:30 (10)3dq3:24 (10)2dq3:37 (10)1dq1:28 (11)3dq3:11 (12)3dq3:12 (13)3dq3:38 (14)3dq3:27 (15)3dq3:26 (15)2dq3:21 (16)3dq3:3 (17)3dq3:35 (18)3dq3:9 (19)3dq3:36 (20)3dq3:17 (20)2dq3:14 (20)1dq1:8 (21)3dq3:6 (22)3dq3:31 (23)3dq3:2 (24)3dq3:39 (25)3dq3:29 (25)2dq3:13 X(26)3dq2S:5 2dq2:22 (30)1dq1:1 X(33)3dq2S:19

Figure 6.2 EAMTS-DC Block Computation

As highlighted in red in Figure 6.4, only two block-movements are recorded. The number of block-movement by DDP, EAMTS-DC, AMTS-DC and Hadoop are 0, 2, 3 and 4 respectively. This corresponding to the efficiency of the task schedulers. DDP is more efficient than EAMTS-DC. EAMTS-DC is more efficient than AMTS-DC and Hadoop the least.

6.4 Evaluation of AMTS-DC and EAMTS-DC

This section presents detailed simulation results and analysis of the performance of the proposed AMTS-DC against EAMTS-DC. Experiments are carried out by varying job size, the capacity of compute nodes and the number of computer nodes.

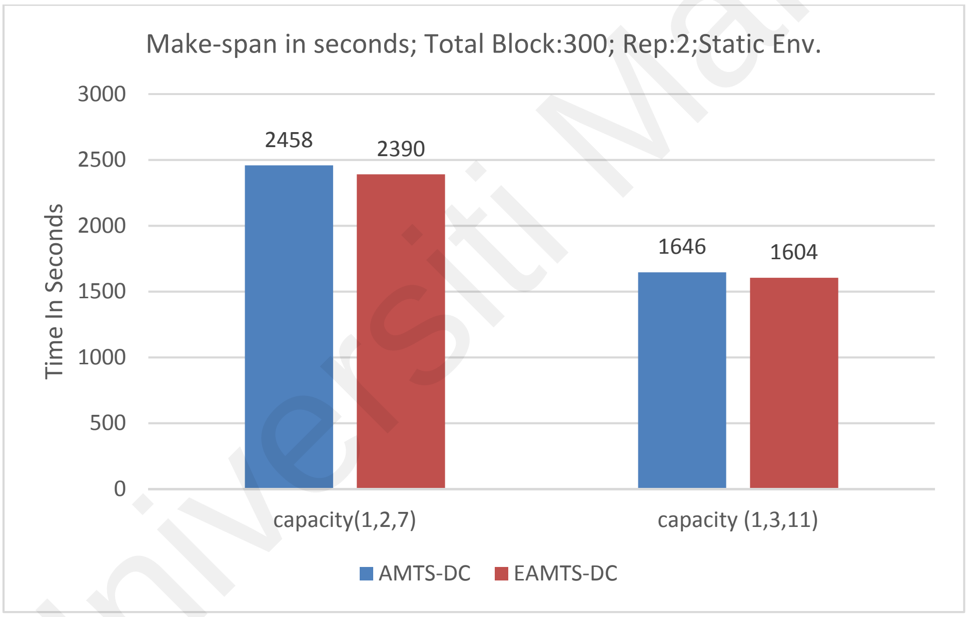


Figure 6.3 Make-span of AMTS-DC and EAMTS-DC with different Capacity (Static Env)

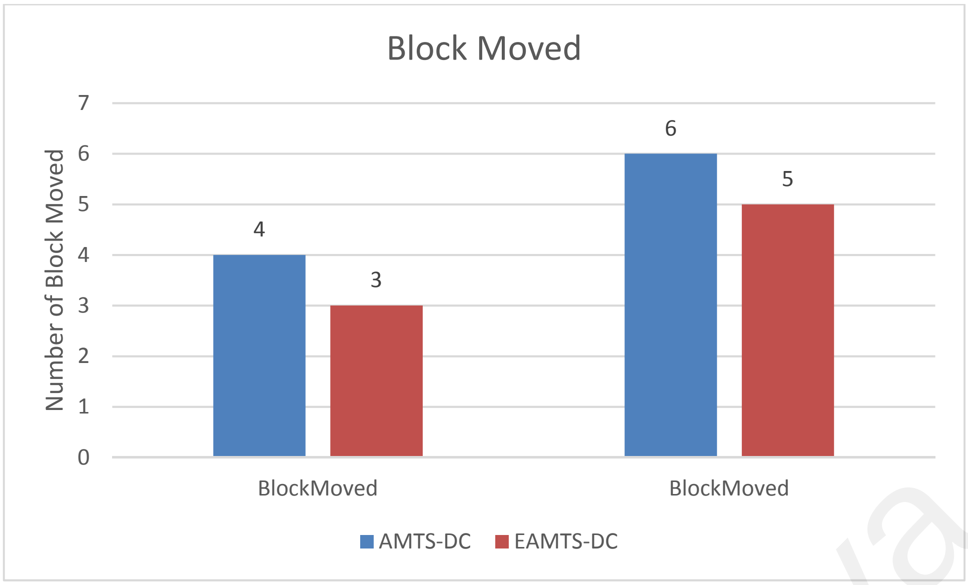


Figure 6.4 Block Moved of AMTS-DC and EAMTS-DC with different Capacity (Static Env)

In Figure 6.5 and Figure 6.6, it can be seen that EAMTS-DC has a shorter make-span and data block movement compared to AMTS-DC. This is because EAMTS-DC uses historical data from the beginning while AMTS-DC need time to "warm-up" and wait for the heartbeat to return the ratio of the computing power of the compute-nodes.

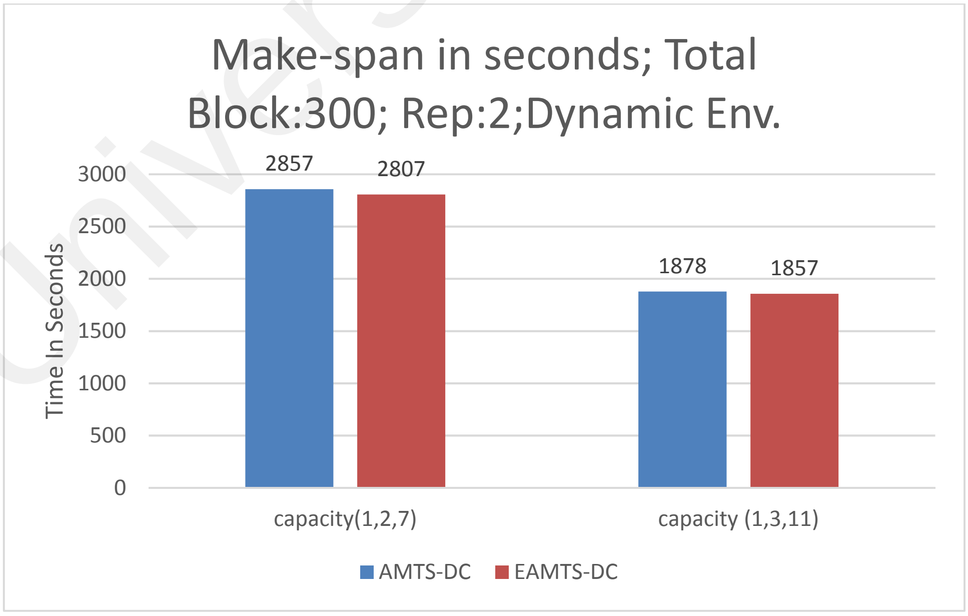


Figure 6.5 Make-span of AMTS-DC and EAMTS-DC with different Capacity (Dynamic Env)

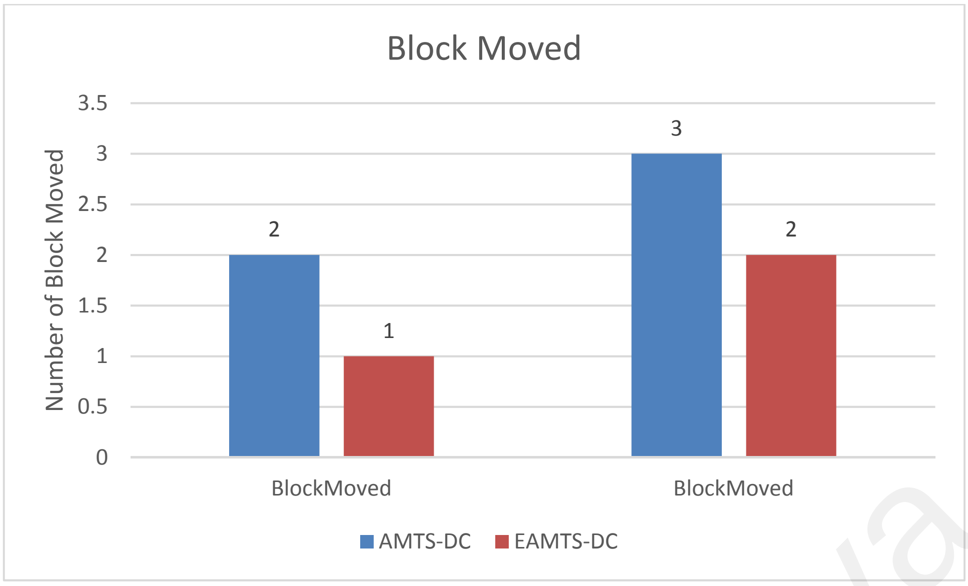


Figure 6.6 Block Moved of AMTS-DC and EAMTS-DC with different Capacity (Dynamic env)

In Figure 6.7 and 6.8, it can be seen that EAMTS-DC again has both shorter make-span and lesser data block movement compared to AMTS-DC.

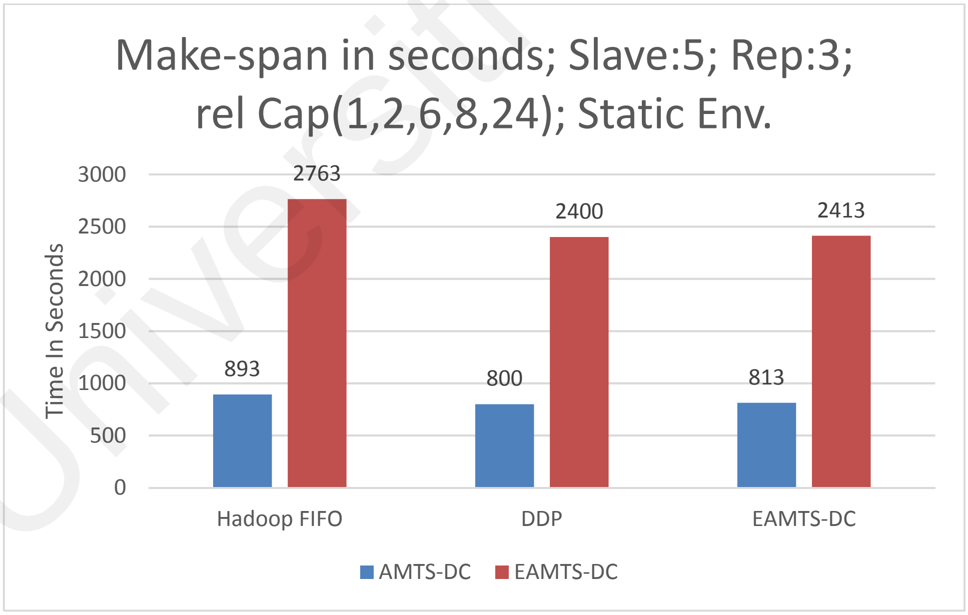


Figure 6.7 Make-span of Hadoop FIFO, DDP and AMTS-DC in 5 Slaves cluster and replication of 3 (Static Env)

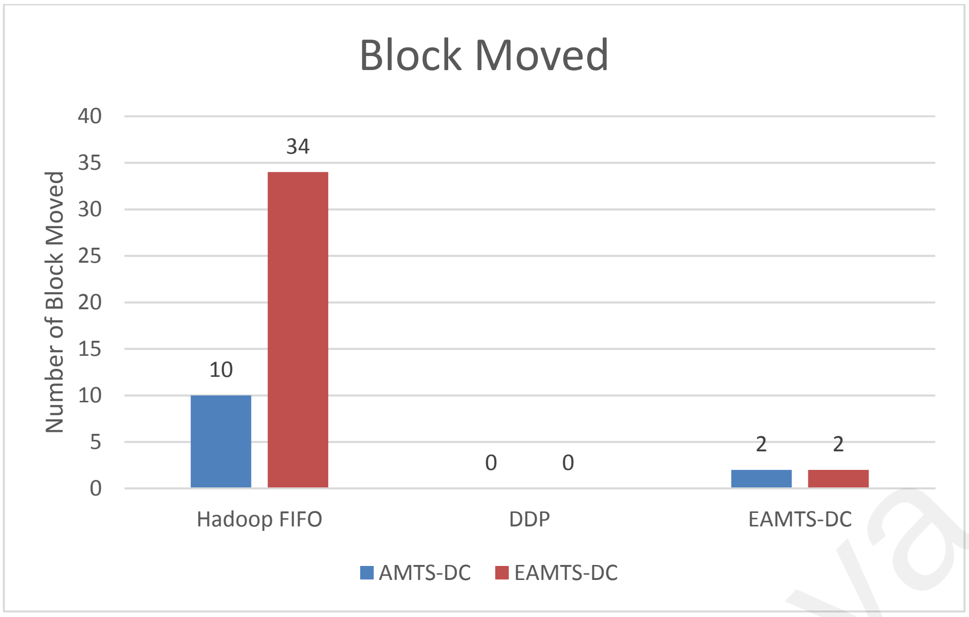


Figure 6.8 Block movement of Hadoop FIFO, DDP and AMTS-DC in 5 slaves cluster and replication of 3 (Static Env)

In Figure 6.9 and 6.10, the setup is 5 slaves with 3 replications. Since it is a static environment, DDP has the best overall performance in terms of make-span and data movement. EAMTS-DC is in the middle. The make-span and data movement of Hadoop FIFO are the poorest. The make-span and data movement of Hadoop FIFO are 2763 seconds and 34 respectively.

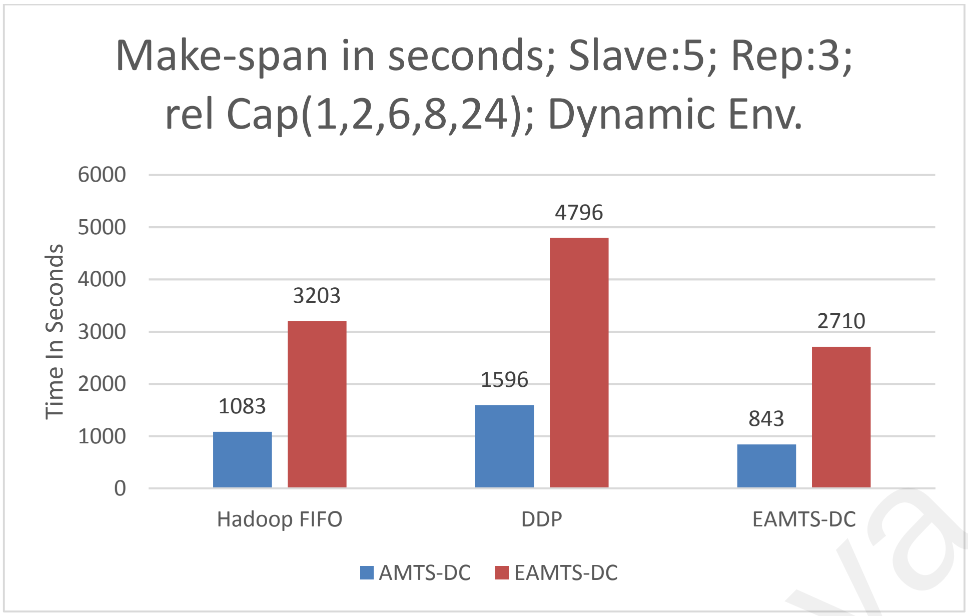


Figure 6.9 Make-span of Hadoop FIFO, DDP and EAMTS-DC in 5 Slaves cluster and replication of 3 (Dynamic Env)

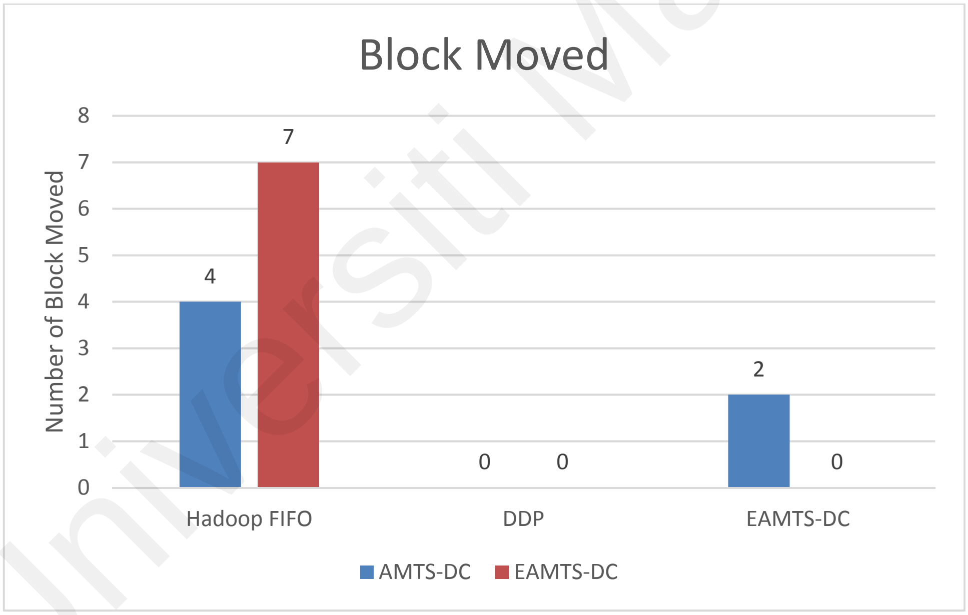


Figure 6.10 Block movement of Hadoop FIFO, DDP and EAMTS-DC in 5 slaves cluster and replication of 3 (Dynamic Env)

In Figure 6.11 and Figure 6.12, the setup is 5 slaves with replication set to 3. Since it is a dynamic environment, DDP has the worst overall make-span performance with a value of 1596 seconds and 4796 seconds in experiment 14 and 15 respectively (no data movement for DDP as the replication for DDP is 1). The proposed EAMTS-DC has the best overall performance with a make-span of make-span of 843 seconds and 2710 seconds in experiment 14 and 15 respectively. EAMTS-DC can do very well because it is able to reduce data movement to minimum. The data movements are 2 blocks and 0 block for experiment 14 and 15 respectively.

Summary

In this section AMTS-DC is analyzed to identify areas where task scheduling could be further improved. Based on the finding, an enhanced version of AMTS-DC namely, Enhanced Adaptive MapReduce Task Scheduler Using Dynamic Calibration (EAMTS-DC), is proposed by augmenting AMTS-DC with historical run time record. Experimental results show that EAMTS-DC has better performance than AMTS-DC in terms of make-span and data transfer. The access to historical run time relative computing capacity allows EAMTS-DC to start reservation early and thus more local data blocks can be reserved. EAMTS-DC effectively reduced data movement and the make-span.

CHAPTER 7: CONCLUSION

7.1 Achievement

MapReduce is a software framework that allows for easy deployment of parallel applications relating to large amount of data set using large computing cluster. Literature survey conducted enables us to define the problems that exists in heterogeneous environment MapReduce task schedulers. The default MapReduce implementation in Hadoop is based on the assumption of homogeneous environment in which every compute node has the same capacity. However, in a heterogeneous environment with compute node of varying capacity, such assumption will actually hinder MapReduce performance.

To address this problem, many works have been proposed. Some works demonstrated that the time taken to complete a MapReduce job could be reduced drastically if data files to be processed are located near to the compute nodes and the data files allocated is proportional to the computing capacity of the node (Lee et al., 2014)

The strength of these approaches lies in the use of historical data. Historical records of all MapReduce jobs are recorded and the proportion of data files to be allocated is calculated based on the historical record. These approaches work well and are very efficient should the environment remains static and unchanged. However, when the environment becomes more dynamic and non-static, the performance of these approaches diminish quickly.

Major contributions of the thesis are as follows:

i. AMTS-DC Task Scheduler algorithm

ii. EAMTS-DC Task Scheduler algorithm

iii. A prototype of AMTS-DC and EAMTS-DC is developed using Hadoop to show the feasibility of the proposed mechanism by extending Hadoop.

AMTS-DC utilizes the heartbeat to dynamically estimate the computing capacity of the compute nodes and rescheduled local task to efficiently process a MapReduce job. AMTS-DC has been implemented in Hadoop and in a simulator (AMST-Sim). Experiments were conducted for AMTS-DC to find out the performance of AMTS-DC in terms of make-span, data movement (data transmission during file transfer from one compute node to the other) by varying the job size and the relative computing capacity among the compute nodes.

It has been observed that for static heterogeneous environment, DDP outperformed Hadoop FIFO and AMTS-DC by having shorter make-span and no data movement between compute nodes. However, as the heterogeneous environment becomes more dynamic, both Hadoop FIFO and AMTS-DC are able to adapt and their performance becomes better. DDP which is designed for static environment is unable to adapt to the dynamic situation and lagged behind and ended with the worse performance. The proposed AMTS-DC has the best performance.

The task locality of AMTS-DC has been further optimized to EAMTS-DC by incorporating historical information of relative computing capacity of the compute-node during the initial stage of the job. EAMTS-DC is evaluated in terms of make-span and data transfer. Experiment results show that in dynamic heterogeneous environment, both make-span and data transfer have been reduced in EAMTS-DC when compared to AMTS-DC.

7.2 Future Work

The following research directions can be focused in future to further enhance the performance of the proposed AMTS-DC and EAMTS-DC. The Inclusion of a prediction model to predict the remote blocks to be processed and these blocks could be pre-fetch to reduce network communication time in terms of data block movement between blocks. Soft computing framework such as fuzzy logics could be used in the modelling and inferencing of the computation capacity of the compute nodes and the dynamic state of the compute-node.

REFERENCES

Abad, C. L., Lu, Y., & Campbell, R. H. (2011). *DARE: Adaptive Data Replication for Efficient Cluster Scheduling.* 2011 IEEE International Conference on Cluster Computing.

Anjos, J. C. S., Carrera, I., Kolberg, W., Tibola, A. L., Arantes, L. B., & Geyer, C. R. (2015). MRA++: Scheduling and data placement on MapReduce for heterogeneous environments.*Future Generation Computer Systems, 42*, 22-35.

Apache Hadoop (2017). Available from: <http://hadoop.apache.org.>[10 August 2017].

Brinkmann, A., Effert, S., & Heide, F. M. a. d. (2007). *Dynamic and Redundant Data Placement.* 27th International Conference on Distributed Computing Systems (ICDCS '07).

Cheng, D., Rao, J., Guo, Y., & Zhou, X. (2014). *Improving MapReduce performance in heterogeneous environments with adaptive task tuning*. Proceedings of the 15th International Middleware Conference, Bordeaux, France.

Chen, Q., Zhang, D., Guo, M., Deng, Q., & Guo, S. (2010). SAMR: A Self-adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment. 2010 10th IEEE International Conference on Computer and Information Technology.

Dai, X., & Bensaou, B. (2016). Scheduling for response time in Hadoop MapReduce.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters.*Commun. ACM, 51*(1), 107-113

Divya, M., & Annappa, B. (2015). *Workload characteristics and resource aware Hadoop scheduler.* 2015 IEEE 2nd International Conference on Recent Trends in Information Systems (ReTIS).

Elkholy, A. M., & Sallam, E. A. H. (2014). *Self adaptive Hadoop scheduler for heterogeneous resources.* 2014 9th International Conference on Computer Engineering & Systems (ICCES).

Gu, R., Yang, X., Yan, J., Sun, Y., Wang, B., Yuan, C., & Huang, Y. (2014). SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters. *Journal of Parallel and Distributed Computing, 74*(3), 2166-2179.

He, C., Lu, Y., & Swanson, D. (2011). *Matchmaking: A New MapReduce Scheduling Technique*. Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science.

Jiong, X., Shu, Y., Xiaojun, R., Zhiyang, D., Yun, T., Majors, J., Xiao, Q. (2010,). *Improving MapReduce performance through data placement in heterogeneous Hadoop clusters.* 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW).

Lee, C.-W., Hsieh, K.-Y., Hsieh, S.-Y., & Hsiao, H.-C. (2014). A Dynamic Data Placement Strategy for Hadoop in Heterogeneous Environments. *Big Data Research, 1*, 14-22.

Lee, L.-W., Scheuermann, P., & Vingralek, R. (2000). File Assignment in Parallel I/O Systems with Minimal Variance of Service Time. *IEEE Trans. Comput., 49*(2), 127-140.

Madathil, D. K., Thota, R. B., Paul, P., & Tao, X. (2008). *A static data placement strategy towards perfect load-balancing for distributed storage clusters.* 2008 IEEE International Symposium on Parallel and Distributed Processing.

Mao, H., Hu, S., Zhang, Z., Xiao, L., & Ruan, L. (2011). A Load-Driven Task Scheduler with Adaptive DSC for MapReduce.

Polo, J., Carrera, D., Becerra, Y., Steinder, M., & Whalley, I. (2010). Performance-driven task co-scheduling for MapReduce environments.

Qutaibah, A., Alqudah, O., Jararweh, Y., & Yaseen, Q. (2014). Multi-threading based Map Reduce tasks scheduling.

Selvarani, S., & Sadhasivam, S. (2010). Improved cost-based algorithm for task scheduling in cloud computing.

Tang, Z., Zhou, J., Li, K., & Li, R. (2013). A MapReduce task scheduling algorithm for deadline constraints (Vol. 16).

Tom White.(2009). Hadoop: The Definitive Guide, O'Reilly.

Uysal, M. Ulus, T, (2007). A Threshold Based Dynamic Data Allocation Algorithm-A Markov Chain Model Approach. Journal of Applied Sciences, 7: 165-174.

Verma, A., Cherkasova, L., & Campbell, R. H. (2012). *Two Sides of a Coin: Optimizing the Schedule of MapReduce Jobs to Minimize Their Make-span and Improve Cluster Performance.* 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems.

Wang, W., Zhu, K., Ying, L., Tan, J., & Zhang, L. (2016). MapTask scheduling in mapreduce with data locality: throughput and heavy-traffic optimality. *IEEE/ACM Trans. Netw., 24*(1), 190-203.

Xu, Y., Cai, W. (2015). *Hadoop Job Scheduling with Dynamic Task Splitting.* 2015 International Conference on Cloud Computing Research and Innovation (ICCCRI).

Xu, Y., Li, K., Hu, J., & Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences, 270*, 255-287.

Yao, Y., Tai, J., Sheng, B., & Mi, N. (2013). *Scheduling heterogeneous MapReduce jobs for efficiency improvement in enterprise clusters.* 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013).

Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., & Stoica, I. (2008). *Improving MapReduce performance in heterogeneous environments*. Proceedings of the 8th USENIX conference on Operating systems design and implementation, San Diego, California.