DYNAMIC LOAD BALANCING POLICIES FOR CLUSTERED DISTRIBUTED SYSTEM

JAY LIM WEI YIK

MASTER OF SCIENCE
(INFORMATION TECHNOLOGY)

MULTIMEDIA UNIVERSITY

MAY 2014

DYNAMIC LOAD BALANCING POLICIES FOR CLUSTERED DISTRIBUTED SYSTEM

BY

JAY LIM WEI YIK

B.Sc. (Hons), Multimedia University, Malaysia

THESIS SUBMITTED IN FULFILMENT OF THE

REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE (INFORMATION TECHNOLOGY)

(by Research)

in the

Faculty of Computing and Informatics

MULTIMEDIA UNIVERSITY
MALAYSIA

May 2014

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in this Thesis has been submitted in support of any application for any other degree or qualification on this or any other university or institution of learning.

_____

Jay Lim Wei Yik

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor, Dr. Poo Kuan Hoong, whose area is in distributed system, has been guiding me throughout this research. He continually and convincingly conveyed a spirit of adventure in regard to research. Without his guidance and persistent help this thesis would not have been possible.

Also, thank you to my co-supervisor, Dr. Yeoh Eng-Thiam, who has shared his experience and provided to me his inputs in this research.

Thank you to those who has provided their feedbacks on my research; especially those that are in the research committee and to those that have evaluated my publications.

Finally, thanks to my parents for the moral support, motivation, and the amazing chances they've given me over the years.

ABSTRACT

In parallel distributed computing system, lightly and overloaded nodes can cause load imbalancing and affect the total time needed to complete a task to increase. Besides that, since the distributed system shared by multiple users with their own computing task, load imbalance could bring impact on to other computing tasks. Lightly loaded nodes which are capable of compute more jobs, might finish their task faster and remain idle whilst heavily loaded nodes are still racing against the clock to complete their computing tasks. As a result, the utilisation of distributed system is not optimised. In order to solve this, load balancing algorithm is employed to balance the loads of each nodes. A load balancing algorithm can be further categorised as static or dynamic load balancing. A static load balancing algorithm formulates the job distribution decision before the execution of the program; during the compilation time. Whilst, dynamic load balancing algorithm distribute jobs during the execution of the program. In other words, static approach is more effective in a homogeneous environment because each node knows the structure of the system. On the contrary, a heterogeneous environment is more suitable to use dynamic approach because the structure of the system and nodes are not known until the execution of the program. Nodes in this approach use the current system state information during the execution of the program to formulate the job distribution decision. At any point of time, the decision regarding a job distribution might change due to the variation of the system. Furthermore, dynamic load balancing can be implemented in a centralised or decentralised model. The centralised model is where one node will formulate the job distribution decision, and decentralised model incorporates at least two nodes to formulate the decision. In dynamic load balancing, load information exchange between nodes is an important factor. Especially in decentralised model, where at least two nodes are taking part in the formulation of a job distribution decision. In decentralised model, an overloaded node that distribute jobs to lightly loaded node is known as sender initiated load balancing. A receiver initiated load balancing is where lightly loaded node request jobs from an overloaded node. Both sender and receiver initiated load balancing need an up-to-date load information in order to optimised the formulation of job distribution. However, exchanging load

information frequently can lead to an increase of communication messages. Besides the exchange of load information, the formation of the distributed system is also an important factor for load balancing. In a decentralised system, connecting each node to all other nodes in the system will increase the communication messages because each node needs to exchange load information with all other nodes in the system; so that the job distribution decision can be formulated. There are numerous load balancing methods in the literature emphasize on the process of load information exchange towards load balancing. Mutual information feedback is one of the method discussed in the literature, where the exchange of load information take place during the transfer of a job from one node to another. A load balancing algorithm namely SIDDLB, using this information exchange method together with the formation of each node connected to the nodes which are relatively more computing power. This thesis enhanced the SIDDLB approach and conducted a series of comparitive performance test with SIDDLB approach. In order to validate the system utilisation of the load balancing algorithm, overall response time of each computational task is measured. The simulation results show that the proposed method showing an overall improvement over the SIDDLB approach.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Parallel computing is an ideal solution for computational intensive application as it breaks up a large task into multiple smaller task called jobs. After that, each job can be executed concurrently in a multiple processors environment. The advantage of parallel computing is to speed up the computation time of a task. A method to achieve parallel computing in a distributed system which consists of geographically scattered interconnected heterogeneous computing nodes that contribute their computing resources to provide seamless access to high performance computing resources. An example of such system is grid computing which combines heterogeneous computing resources from multiple administrative domains to achieve a common goal (Foster, Kesselman, & Tuecke, 2001). Heterogeneous nodes are differ in terms of computing resources, such as computing power, memory storage, and network bandwidth. Which means, each node is not having the same computing capability. Due to the heterogeneity of nodes, higher computing capability nodes, that generally have more computing resources, may complete their assigned jobs faster to those lower computing capability nodes. So, by distributing jobs evenly to each node may increase the overall of completion time or response time of a task. This is also refer as load imbalance because there are nodes in the system are under utilising or idle whilst some nodes still have jobs to be executed. By applying load balancing algorithm, it can distribute jobs from heavily loaded nodes to lightly loaded nodes so that the resources in the distributed system are always fully utilising and also to speed up a computational application (Zhou, 1988). A heavily loaded node is a node that has exceeds its maximum computing capability, where as a lightly loaded node is below its average computing capability. In short, the goal of a load balancing algorithm is to maximise the system resources and minimise the overall response time of a task.

1.1 Background

Load balancing algorithm can be categorised into static or dynamic load balancing (Casavant & Kuhl, 1988). Both types of algorithm aimed to optimise the utilisation of distributed system by distributing jobs across nodes. A static load balancing algorithm tends to formulates job distribution decision before the execution of a program, that is the information needed to formulate the decision is already known during the compilation time. On the other hand, dynamic load balancing algorithm formulates job distribution decision at the runtime of a program with the use of the current system state information. A static approach works at its most effective when it is a homogeneous environment. Only in homogeneous environment, each node is having the same property and the structure of the distributed system is already known. In a heterogeneous environment, where nodes are distinguished by their computing capabilities, it is more difficult to determine the load level of each node at all time. The algorithm that fits such environment is the dynamic load balancing algorithm because it formulates job distribution at the runtime and it uses the current state information to formulate the decision.

Dynamic load balancing algorithm can be implemented in a centralised or decentralised model. Both models describe how load balancers are formed. The centralised model, is which there is only one node that will be the load balancer to decide on the formulation of the job distribution and thus, the other nodes in the distributed system will have to update their load information to this node. As for the decentralised model, there are at least two nodes will be the load balancers and they work together to formulate the job distribution decision. One common behaviour from these models is that both require each node to send their load information to the load balancers so that the load balancers can workout with the job distribution decision based on their current load in order to maximise the system utilisation. The difference between the two models comes in when the scalability of a system increases. Which means, the size of the distributed system is increasing. The centralised model will be encountering communication bottleneck at the load balancer because all nodes will have to send their load information to the load balancer. The communication bottleneck

will cause a single point of failure or a less optimised job distribution decisions due to the late arrival of the load information. The decentralised model more likely to scale up since there are at least two load balancers in the distributed system to workout with the job distribution decision. In case the one of the load balancer is down, there is another load balancer.

Load balancers in decentralised model can be categorised as sender initiated or receiver initiated. A bidding algorithm proposed by Stankovic and Sidhu (1984), initiates a load transfer from a heavily (overloaded) node to lightly loaded node. This is categorised as the sender initiated approach. The overloaded node sends a job bidding request to other nodes in the system and wait for their reply. After all other nodes replied, the node with the highest bid will receive job from the overloaded node. On the other hand, a receiver initiated method is whereby nodes seeking for jobs instead of sending request to offload jobs. For example, the drafting algorithm proposed by Ni, Xu, and Gendreau (1985) work in such a way that each node sends out a draft of a job model to search for more jobs. The drafted job model describes the job that a node can handle. If there is a similar job found according to the drafted job model, that job will be transferred to the requester. Casavant and Kuhl described that a sender initiated approach is when an overloaded node that will initiate the job transfer request, and a receiver initiated approach is lightly loaded node seeking for more jobs to execute. Despite the different approach, both methods often need the load information of other nodes in order to formulate a job distribution decision. Hence, the load information exchange between nodes is an important factor which could affect the job distribution decision.

As far as the exchange of load information is a crucial element in determining the job distribution decision, Acker and Kulkarni (2007) proposed an algorithm to adaptively keep track of nodes' load information. The algorithm stores a list of nodes containing their load information and the list is always updated through the exchange of load information periodically. This has given an advantage to the algorithm to be able to keep track of node joining and leaving the system. However, as there are more

nodes joining into the system, managing the list has became another problem and also, the increase of the amount of communication messages. Lu, Subrata, and Zomaya (2006) introduced a technique called mutual information feedback to exchange load information between nodes which can minimise the overwhelming of communication messages.

Apart from the exchange of load information, the construction of a neighbour list is also an important factor in decentralised approach. It can faster in determining the possible node to offload the workload during the event of an overloading node. Nandagopal, Gokulnath, and Uthariaraj (2010, 2011) have proposed a sender initiated algorithm called as *Sender Initiated Decentralised Dynamic Load Balancing* (*SIDDLB*), which forms the neighbour list based on the *a*) computing power, and *b*) network delay. As a result, the neighbour list will only contain those nodes that are nearby and higher computing power. So, whenever a node is overloaded, a job is always transferred to a higher computing powered node.

1.2 Research Problem

The challenges of a dynamic load balancing algorithm is the load information, that describes the *a*) construction of neighbour list, and *b*) load information exchange method, holds the key to optimise a job distributed decision in a decentralised environment. The construction of a neighbour list can be as simple as storing all nodes or selectively stores nodes in the system. The neighbour list construction approach used in *SIDDLB* as described in section 1.1, selects only nodes which are relatively higher in computing power with minimum network delay. Assuming that the network delay is negligible, this leaves the node that posses the highest computing power in the system would have an empty neighbour list. Furthermore, when this node is overloaded whilst a job arrives, it has no other options but to accept the job since it has no neighbours to offload the job to. In other words, there are no other nodes in the distributed system which are more capable than this node. The challenges and motivation of this research are:

1. The neighbour list construction in *SIDDLB* approach is designed to accommodate feasible nodes. And this has result in some nodes with an empty neighbour list, which could affect the utilisation of resources in a distributed system.

2. The information exchange protocol used is *mutual information feedback* which highly depends on a job transfer to update the load information of both sender and receiver. If a node continuously execute jobs locally, that is the node is executing job by itself, then the neighbour list would not be updated.

3. Again, with regards to the highest computing powered node, it should not be leaving the neighbour list empty. In some events where the lower computing capability nodes are also the lightly loaded nodes and they are capable of executing the jobs rather letting them idle. They could ease the higher computing powered nodes by offloading some jobs from the higher computing powered nodes.

1.3 Research Objectives

This research focuses on the study of a dynamic load balancing algorithm in a distributed system. Specifically, the study of the load information policy. The following are the objectives of this research:

1. To analyse and design of a decentralised dynamic load balancing algorithm that focuses on the neighbour list.

2. To develop a load information exchange mechanism in order to keep the neighbour list up-to-date.

1.4 Research Outline

In this research work, the work is organised by first investigate the importance of a parallel computing in distributed system, which lead to the study of the significant impact of workload towards the distributed system. Then, follow by the study of decentralised dynamic load balancing algorithm. In chapter 2, the structure of decentralised system and the related work of load balancing algorithm are described.

In chapter 3, the research tools, methods and simulation model used are described. This includes the simulation study of the workload impact in a distributed system and also the proposed load balancing algorithm *Heuristic Neighbour Selection Algorithm* (*HNSA*) are described in detail.

Chapter 4 will discuss the results from the impact study. Besides that, the proposed *HNSA* algorithm in 3 is also simulated and discussed.

Finally, chapter 5 summarises the whole thesis and the research outcomes. Besides that, the future work is also described to improve on the proposed *HNSA* algorithm.

CHAPTER 2

LITERATURE REVIEW

The increasing usage of computing intensive application and ubiquitous computing have led to the use of distributed parallel computing such as grid system. As explained by Iosup, Dumitrescu, Epema, Li, and Wolters (2006) on *How are Real Grids Used?*, that a grid system is made up of multiple clusters from which each cluster consists of computers connected to each other locally to solve a common problem. Clusters are often heterogeneous and geographically dispersed. The heterogeneity of clusters can result in load imbalance whereby a cluster can be in the state of idle, heavily loaded or lightly loaded with jobs. Load imbalance decreases the utilisation of resources in the system with some clusters sitting idle while others still processing jobs. A research study by Groot, Goda, and Kitsuregawa (2010) on the data intensive distributed computing such as *Hadoop* supported this point. Besides the grid which coupled clusters to work on a common goal where load balancing issue concerns, another type of distributed system described as peer-to-peer computing which allows computing resources to join and leave the system freely. For example, the SETI@Home project (Anderson, Cobb, Korpela, Lebofsky, & Werthimer, 2002) which focuses on using public computing resources to analyse the radio signals to search for extra terrestrial intelligence. Any personal computer around the world can participate in this system to assist in analysing the radio signals. Another commonly use of load balancing is in the web that can overcome flash crowd as well as those request that requires larger time scale to serve (Ranjan & Knightly, 2008). Kalantari and Akbari (2008) employed prediction method into the load balancing algorithm as well as considering the deadlines of the tasks into account. Mamat, Lu, Deogun, and Goddard (2012) focused on using real-time divisible load technique towards load balancing in a clustered environment for applications that involve in a huge data to be computed.

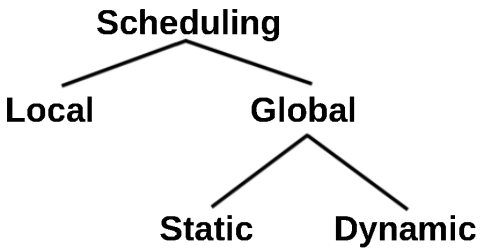

Figure 2.1: Classification of Scheduling

Scheduling algorithm can be classified into local scheduling and global scheduling (Casavant & Kuhl, 1988; Waraich, 2008; Mukhopadhyay, Ghosh, & Mukherjee, 2010). A local scheduling refers to the scheduling of task at the operating system level. On the other hand, a distributed system level of scheduling is known as global scheduling which formulates the decision of which is to execute a task in order to achieve minimum execution time. Moreover, a global scheduling can be either implemented in a centralised model or decentralised model. A centralised model consists of only one node to formulate the scheduling decision, whereby the decentralised model has more than one node, assisting each other to decide at the scheduling process. In global scheduling process, despite it is implemented in a centralised or decentralised model, a job scheduler has to know about the load status of other nodes prior to formulate job distribution decision. In concerning about the method to obtain load status, this can be further categorised into static and dynamic load balancing. A static load balancing is which the load information required to formulate the decision is known before the execution of the task and can be only assigned once. On the contrary, dynamic load balancing algorithm formulates the decision during the runtime of the system which uses the current state information of the system and able to reschedule task to another feasible node if necessary. Fig. 2.1 and Fig. 2.2 illustrate the centralised and decentralised model, and the classification of scheduling algorithm.

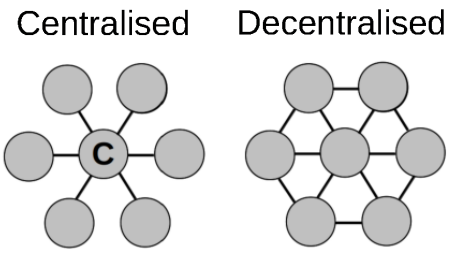

Figure 2.2: Centralised and Decentralised Distributed System Model

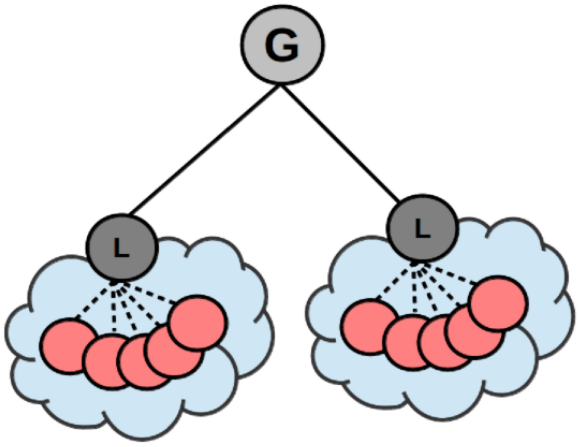

Figure 2.3: Tree-based Distributed System Model

2.1 Centralised Model

A tree-based task scheduling model illustrated in Fig. 2.3 has a centralised load balancing control (Ahn, Youn, Jeon, & Lee, 2007; Barazandeh, Mortazavi, & Rahmani, 2009; Barazandeh & Mortazavi, 2009; Lin & Shen, 2010). Ahn et al. (2007) uses fuzzy logic controller in the load balance to overcome sudden burst of tasks within a short time period. Barazandeh et al. (2009); Barazandeh and Mortazavi (2009) proposed a hierarchical structured load balancing that has both the simplicity of static and adaptiveness of dynamic methods. Lin and Shen (2010) described that there are 3

types of nodes in this model: i) global load balancer, which manages the distributed system resources and handle arrival jobs, ii) local load balancer, which is the node that is connected directly under the global load balancer and manages a cluster of computing nodes, and iii) P2P computing node, which connects to its own respective local load balancer and execute all the tasks requested by its local load balancer. Besides that, this approach is easy to implement because it has a global load balancer which is collecting the load information of its local load balancers and can apply to many parallel programming model such as divide and conquer. However, when it comes to the scalability of the system, the global load balancer and local load balancers will experience communication bottleneck and over a prolonged period of time, it could fail the whole system.

2.2 Decentralised Model

In a decentralised model, nodes often have to keep track of who their neighbours are and constantly have to update their load information in order to perform a job distribution decision. Unlike in the centralised approach which the knowledge about all the nodes in the distributed system only resides at one load balancer, decentralised approach keeps the neighbour list in more than one node and thus, it is important to maintain those load information in the neighbour list up-to-date.

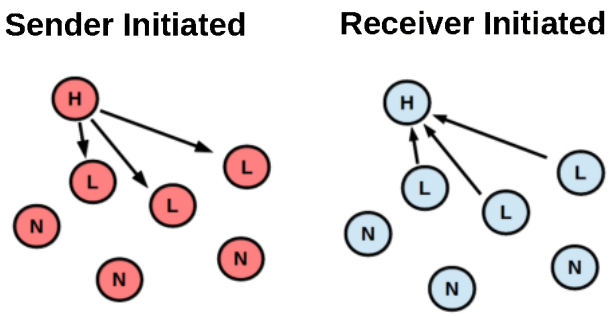

Figure 2.4: A Sender and Receiver Initiated Load Balancing Model

In a decentralised model, any nodes are able to trigger the load balancing algorithm since there is no longer a centralised load balancer to keep track of all nodes. In particular, the load balancing algorithm can be triggered by an overloaded or lightly loaded node, which is known as sender or receiver initiated respectively. Fig. 2.4 illustrates sender and receiver initiated load balancing model. A node with label *H* represents an overloaded node, *L* represents a lightly loaded node, and *N* represents the node is neither in both state. Besides that, a node node with an arrow pointing outwards is the node which initiate the load balancing algorithm. Stankovic and Sidhu (1984) presented a bidding algorithm which is an example of sender initiated load balancing algorithm. During the load balancing process, an overloaded node sends out a bidding request to other nodes to bid for its jobs. On the receiver side, upon receiving the bidding request, it replies back to the sender with a bid placement. After collecting all of the bids from the system, the sender then will select the winner and transfer some jobs to it.

As for the receiver initiated load balancing algorithm, Ni et al. (1985) proposed a drafting algorithm which the lightly loaded nodes are responsible for the balancing of the system. Once a node has become lightly loaded, it will first check the load of all other nodes in the system. If there are no overloaded nodes, then the system is considered to be balanced. Otherwise, the lightly loaded nodes have to send draft request to those overloaded nodes in order to draft the age of their jobs. Upon receiving the draft requests from the overloaded nodes, then the lightly loaded nodes request the overloaded nodes to send the oldest job. It is noted that the nodes in either sender or receiver initiated model are require to know their neighbours in other to perform load balancing and the methods to update the load information which is known as the information exchange process. Hence, the construction of neighbour list, as well as the information exchange process is an important factor in rescheduling jobs to other nodes.

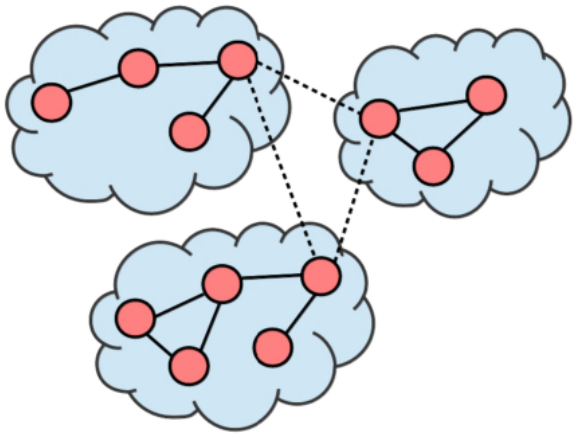

Figure 2.5: A Hierarchical Distributed System Model

2.2.1 Neighbour List

Gupta and Gopinath (1990) presented a two levels load balancing in a hierarchical distributed system. The neighbour list construction proposed by Gupta and Gopinath is to partition all the nodes into a desirable group size based on the load of the network communication link to another node. If a node frequently communicates to another node through a heavy loaded link, both nodes will be separated into different groups. This process will continue to separate all the nodes until the desirable size of a group has reached. Once the limit of a group has reached, a new group will be created. Eventually, the distributed system forms a set of groups. For example, in Fig. 2.5 illustrates the distributed system model with the maximum desirable size of a group is 5. Moreover, since the links between each cluster are consider to be heavily loaded communication link, each group will balance the load among themselves in order to minimise the communication to another partition. The inter group load balancing is done only if the intra group load balancing fails. In summary, this approach presented two types of neighbour list. The first neighbour list which made up of nodes within a group which given the higher priority for load balancing and the second list consists of nodes from another group which will be used for load balancing if the initial list has fail to balance. Besides that, the construction of neighbour list is depends on the load of network link and thus, this approach is best for nodes with homogeneous computing

power. Antonis, Garofalakis, Mourtos, and Spirakis (2004) presented a virtual binary tree structure over the actual network and demonstrated a low communication message using difference-initiated technique for the load balancing. More recently, Stavrinides and Karatza (2009, 2010) proposed a similar approach, which is using hierarchical based distributed system, in a homogeneous distributed real-time system; such as video-on-demand. Besides that, due to the structure of the system, it also tends to encounter network congestion. Furthermore, when the deadline of the job is a factor, a real-time system has to trade off the precision of the computations for time, hence, it allows imprecise computations; e.g. lowering the quality of the video. Grande and Boukerche (2011) uses hierarchical approach to evenly distribute the load for a large-scale HLA simulations in a heterogenous environment. The tree structure able to detect imbalances and repartition the distributed load.

Acker and Kulkarni (2007) proposed a control protocol to dynamically construct the neighbour list for a node. This has given the algorithm the ability to accommodate nodes joining and leaving the system at any point of time. Besides, Acker and Kulkarni assume that the system has to be able to support multicast messages. For example, a node sends a multicast message to the other nodes in the distributed system periodically to notify other nodes about its load information or insert it into their neighbour list if it does not exists. This action is being performed periodically by other nodes as well. For example, when a node A receives a multicast message from another node B, A will first check if B is its neighbour. If B is not found in A, then A will insert B into its neighbour list. Otherwise, the load information regarding the sender, which is B, will be updated to the neighbour list of A. Again, this process is repeated periodically and whilst during the load balancing process, nodes are only require to balance the load among their neighbour list.

Riakiotakis, Ciorba, Andronikos, and Papakonstantinou (2011) proposed a decentralised approach by forming a virtual ring of nodes; each node only needs to connect to 2 nodes. The computation of the jobs is pipelined from one node and pass on to the next node until it reaches back to the first node. In this virtual ring, node

only exchange load information between the 2 connected nodes and thus, using the up-to-date load information to chunk the job size. So, a lightly loaded node might take a bigger chunk as compared to the heavily loaded nodes. This approach is able to handle application with dependent tasks.

Several studies by (Lu et al., 2006; Lu & Zomaya, 2007; Lu, Subrata, & Zomaya, 2007) have presented a distributed system model which the construction of neighbour list based on the nodes that are relatively near to each other in terms of network delay. Lu et al. have determined that the acceptable network delay between a pair of nodes should not be not more than 1.5 times from the nearest node. For example, assume that there are N total number of nodes in the system, for each node n_i, sort all the nodes by transfer delay in an ascending order and insert those nodes which their transfer delay is less than 1.5 times from n_i. By this way, the load balancing algorithm can assure that during an event of load imbalance, the time taken for a job to transfer from an overloaded to lightly loaded node can be minimised. For instance, a node A considers node B as neighbour as long as the network delay is within 1.5 times from the nearest node. As a result, nodes will not have an identical set of neighbours and balance the load among their neighbours.

Apart from this, a similar approach applied by Nandagopal et al. (2010, 2011) which is to form the neighbour list based on both computing power and network delay of a node. Intuitively, instead of having the nodes that are relatively near to be its neighbour, this approach forms a set of nodes that has higher computing powered. With regards to this, when a node is overloaded, the load balancing algorithm will make sure to transfer a job to another node which has higher computing power. Besides that, this algorithm also assumes that if a job is transferred to the highest computing powered node, it means that there should not be any other nodes that are capable to execute the job. As a result, this approach can achieve load balancing with the assumption that the highest computing powered node will not be overloaded.

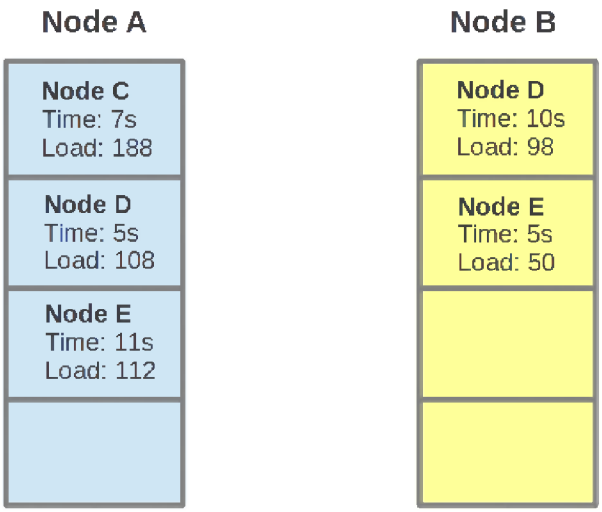

Figure 2.6: Neighbour List for Node *A* and Node *B*

2.2.2 Information Exchange

With regards to the neighbour list proposed by Acker and Kulkarni (2007), the method they used was periodic information exchange. However, the load information update interval which depends on the total number of neighbours, varies from node to node. For example, assuming that T_i is the total number of neighbours for node i and k is a default small fixed interval. The update interval is said to be $T_i \times k$ and thus, the update interval increase as the total number of neighbours increase. As a result, the overall communication message can be minimised. However, the increasing number of neighbours for a node may affect the validity of the load information. In order to minimise this issue, instead of updating all the neighbours' load information, partial information exchange method such as mutual information feedback approach can be used (Lu et al., 2006; Lu & Zomaya, 2007; Nandagopal et al., 2010, 2011).

Lu et al. (2006) mentioned that mutual information feedback can minimise the overall communication messages as compared to the traditional approach that requests load information periodically. The approach transfers load information along with a job transfer in order to minimise the network traffic. Moreover, since in a decentralised model, each node maintains its own neighbour list and thus, some of the

load information about the neighbour will be shared to the receiver too. Take Fig. 2.6 as an example of neighbour list for node *A* and *B*, when node *A* transfers job to node *B*, the information exchange process will piggyback the load information of *A* and some nodes from its neighbour list, onto the job transfer message. In response to node *A*, node *B* in return will reply with a job acknowledgement message and piggyback some neighbours onto the message. The criteria for updating the neighbour list upon receiving the message from the sender are: 1) whether if such node exists in the neighbour list of the receiver, and 2) comparing the timestamps of the load information. From Fig. 2.6, assuming that all information of the neighbours in node *A* are shared to node *B*, and as a result, node *B* will ignore the updates of node *C* and *D*. This is because node *C* is not the neighbour of node *B* and the load information of node *D* is up-to-date as compared to the one node *A* had. Similarly, this approach has been employed by Nandagopal et al. (2010, 2011) and based on their construction of neighbour list as explained in section 2.2.1, some nodes in distributed system will result in an empty neighbour list. In particular, the node with the highest computing power. Santana-Santana, Castro-Garcia, Aguilar-Cornejo, and Roman-Alonso (2010) presented another partial information management technique that incorporated bidding policy to populate the empty list with those non-empty list. Whenever a node has an empty list, it will start sending request to other nodes that has the information.

Plentz, Montez, and de Oliveira (2008, 2011) have developed a prediction mechanism called Available Slack (AS) to predict the response time of the distributed threads. They proposed a probabilistic method which allows them to estimate the response time of the distributed threads. Rao and Huh (2008) presented an adaptive scheduling that predicts a run-time of a task in a targeted Grid without actually submit the job to the Grid. In OS level, Beltran, Guzman, and Bosque (2008) have presented prediction algorithm that uses the current system state information gathered from their monitoring tool to assign task to the CPU. Yang, Foster, and Schopf (2003); Wu, Hwang, Yuan, and Zheng (2010) have incorporated the use of prediction algorithm in a distributed system. Some authors incorporate the game theory approach in load balancing algorithm to predict performance (Subrata, Zomaya, & Landfeldt, 2008;

Grosu, Chronopoulos, & Leung, 2008; Penmatsa & Chronopoulos, 2011). Dong, Li, Wu, Xiao, and Ruan (2012) demonstrated an adaptive load information exchange, which is to prevent frequent load information exchange, and a prediction of workload to minimise the communication messages. A prediction algorithm learns the behaviour of the load of a given node. Given enough historical data, the prediction algorithm is able to predict what will be the workload in the near future. The number of historical data used is called as the window size. In general, one of the strategies of load prediction is self-correcting the average load value, which is known as homeostatic prediction. With regards to this approach, it depends on the current load value and assumed that, if the current load value as compared to average historical load value of window sized N decreases, then the predicted load will be decreased as well. The same case is applied if the current load value increases. The second prediction approach is known as the tendency-based pridiction which is predicts the future load according to the tendency of the time series change. For instance, if the current load value as compared to the previous load value increases, then the tendency will likely to increase, which is the next predicted load value will increase. As for the case where the tendency decreases, the predicted value will be decreased.

CHAPTER 3

METHODOLOGY

This chapter describes the tools and methods that are used in this thesis. In section 3.1, the research tools that are used in this research are presented, follow by a study on the significant impact of workload variation towards the performances of distributed system in section 3.2. To study the performances of distributed system, a structured distributed system, namely a tree-based distributed system, is used and together with the use of π computation as the task to be executed. From the study of the performances of distributed system, the importance of load balancing is described. Finally, in section 3.3 describes the proposed dynamic load balancing algorithm, namely *HNSA* that improves the *SIDDLB* approach proposed by Nandagopal et al. (2011).

3.1 Network Simulator

3.1.1 OMNeT++

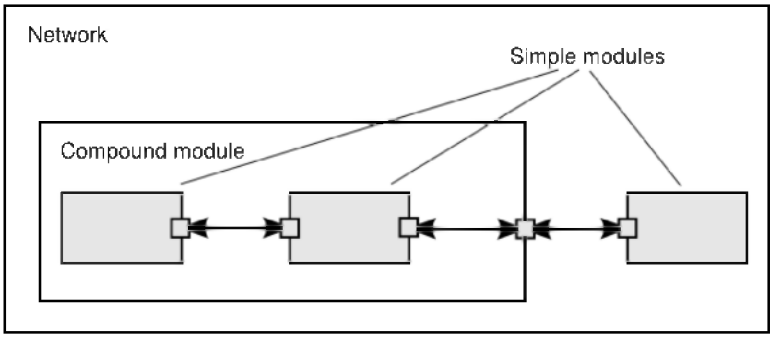

Figure 3.1: OMNeT++ Simulation Model

Throughout this research, the network simulator used was based on the OMNeT++ simulation library and framework. OMNeT++ is an open source, extensible, modular, component-based C++ simulation library and framework (Varga & Hornig, 2008). Generally, it is composed of 2 types of modules, which are, simple module and compound module. Simple module is also known as the active module which is where the logic

of the module is defined. On the other hand, a compound module is made up of 1 or more modules of any type with unlimited number of hierarchy levels. There is another module known as a network module which is can be categorised as a compound module. The difference between a network module and a compound module is that a network module is the final module defined for a simulator program. Fig. 3.1. illustrates the general structure of a network simulator using the OMNeT++ simulation library and framework. Modules can be linked together and communicate through message passing. Since OMNeT++ is modular based, it provides opportunity to build an extension on top of the OMNeT++ simulation library. The following section describes the extension library that is used in this thesis to accomplish the simulation.

3.1.2 OverSim

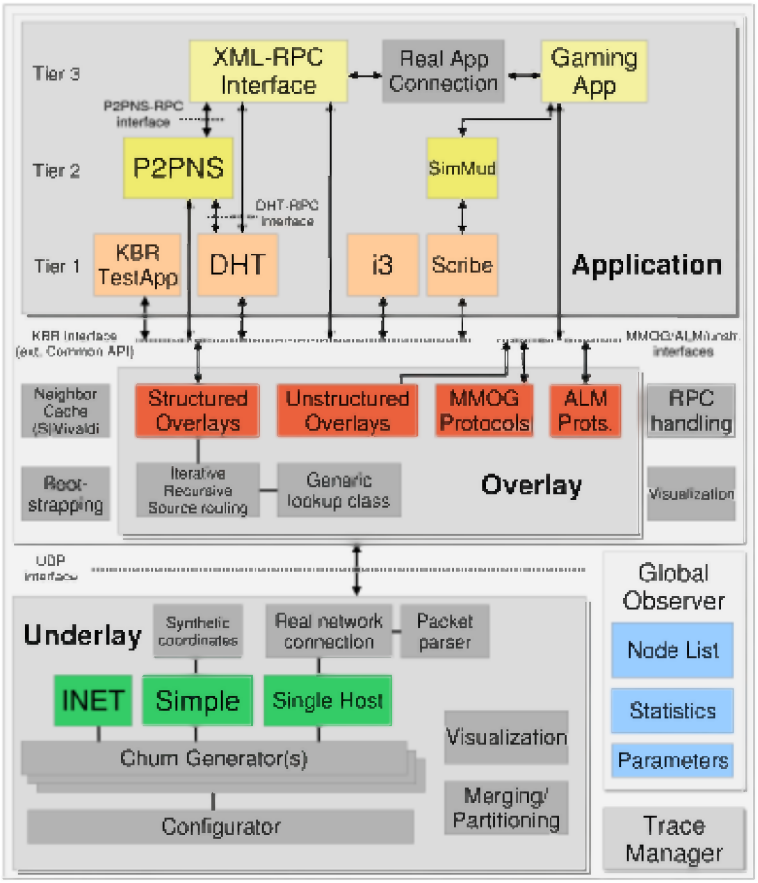

Figure 3.2: OverSim Framework Model

OverSim is an overlay and peer-to-peer (P2P) open source network simulation framework built on top of OMNeT++ simulation library (Baumgart, Heep, & Krause, 2007). It consists of the modules from the underlying network to the application

layer which has reduced the tasks for constructing a network simulator. Additionally, OverSim has provided structured P2P models and unstructured P2P systems protocols. An example of structured P2P model is Chord and unstructured is GIA.

Fig. 3.2. illustrates the structure of OverSim framework. The OverSim framework can be categorised into 3 different layers: underlay, overlay and application layer. The underlay layer consists of the network medium and handles all communication messages. Overlay layer sitting on top of the underlay layer visualises the network topologies such as a tree network. Each overlay node has an application running and the application is defined in the application layer of the OverSim framework.

3.2 The Impact of Different Sizes of Workloads

Distributed computing is a form of loosely-coupled parallel computing where by, multiple connected computing nodes work together and form into a single virtual supercomputer to achieve a common goal in a distributed system. Distributed system which often consists of heterogeneous nodes which are differ in terms of their computing capability such as the processing power and memory storage. Due to the heterogeneity of nodes, nodes can fall into either one of these states: idle, lightly loaded or heavily loaded. The load of a node depends on its computing capability. For example, if a node is said to be able to execute N number of jobs at a time, the maximum load will be N. So, if the total number of jobs being executed and waiting is less than N, the node is said to be lightly loaded. Whereas, when N is 0, the node is said to be idle. As for another possible state, heavily loaded, is when the total jobs being queued and executing are more than N number of jobs. In distributed system, when there are nodes fall in these states at the same time, the system is considered as not fully utilising its resources. As a result, increase in the response time of the task, that is the time needed to complete the task. While having jobs queueing at the heavily loaded nodes, they can be executing at either the idle or the lightly loaded nodes. This section describes the system model to be used to analyse the study of the impact of different sizes of workload towards the distributed system and the method to measure the system performance. Besides that, the set-up of several scenarios are described, to study and measure the significant

impact. To carry out the simulation which can be partitioned into smaller pieces, the computation of the value π is used and explained.

3.2.1 Distributed System Model

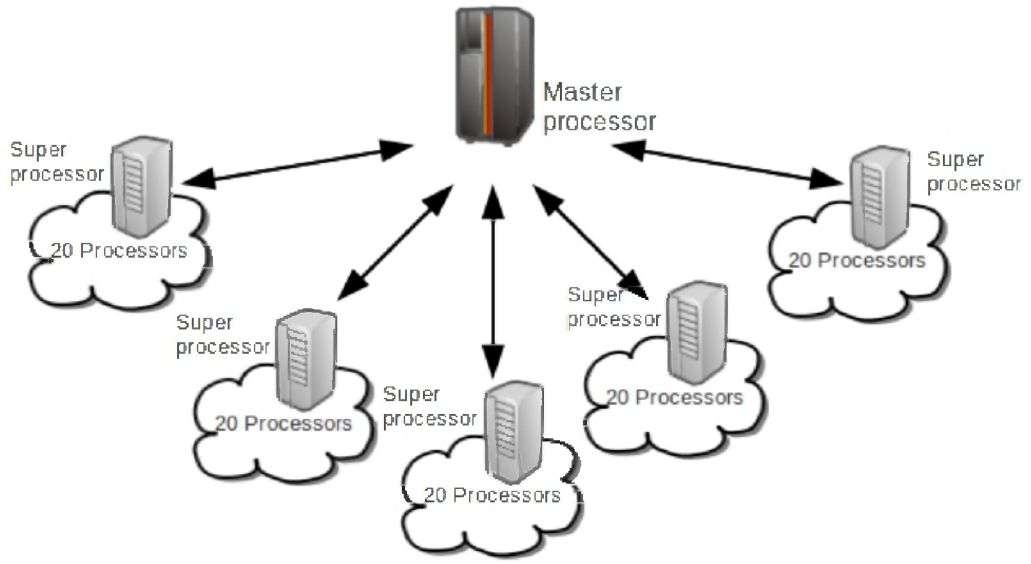

Figure 3.3: Tree-based Distributed System Model

The balancing of workloads between nodes is an important factor to be taken into consideration, and therefore in this experiment a structured tree-based distributed system is introduced as illustrated in 3.3. In other words, the changes in computational behaviour will affect the system's response time. In such cases, some nodes may have more jobs to execute as compared to other nodes in the system which have little or no jobs. The imbalance of workloads between nodes implies that the initial partitioning of the workload is no longer acceptable. It is thus imperative that the distributed system able to detect the amount of work to be assigned to each node should be balanced at run-time in order to increase utilisation of resources and improve the overall performances. To show and study the effects of imbalance node towards the significant impact on the overall performance in a distributed computational environment, a simulation of computational π value will be conducted.

Fig. 3.3. illustrates a structured tree-based distributed system which composed of the main task distributor labelled as Master Processor (MP), 5 Super Processors (SPs)

which act as a cluster manager and 20 Processors (Ps) under each SP that runs the same application. In a tree-based distributed system, MP initiates a task and partition it into smaller pieces in order to distribute to its 5 SPs. SP will further split the receiving piece of task into smaller pieces so that it can be concurrently executed at its Ps. In other words, a computational task will be splited from a node into several smaller portions until each independent job reaches to the leaf nodes to execute concurrently. With regards to such computational task that is able to divide the task into smaller pieces, the computation of π is used.

To evaluate the imbalance of workloads, there are two simulation models, namely *Single Processor* and *Distributed Processor*s. The *Single Processor* involves only 1 node to execute the computation task for several iterations. *Distributed Processor*s simulates a computational task for several iterations in a distributed computing environment as illustrated in Fig. 3.3. Both simulation models will simulate the computation of π value at 6 different millionth number of intervals (i.e. 100, 200, 400, 600, 800 and 1000 million number of intervals). Each interval is executed for 11 iterations in order to obtain the average result. The usage of various number of intervals is described in section 3.2.2. The *Single Processor* simulation is to study and compare the parallelisation performance of *Distributed Processor*s simulation. In *Distributed Processor*s simulation, various scenarios of workloads listed in Table 3.1 has been assigned to each SP to analyse the impact towards the response time. Response time is the measurement of the total time taken for a task upon it is created by MP until the final answer is formulated. In other words, response time is the time required to compute a task in a distributed computing environment which takes waiting time and execution time of each node into account. Therefore, in order to study the impact of workload imbalance, the response time will be measured. To further analyse the response time, the measurement of waiting time and execution time of each node will be conducted to investigate how much time has a node been waited until its turn to be executed and what is the actual computing time taken by each node to complete its portion of task. The main objective of the various scenarios listed in Table 3.1 is to show that an optimal workload distribution yields better response time as compared to those unbalanced workload

distribution. The most optimal workload distribution in Table 3.1 is Scenario 1, which evenly distribute workloads across each SP. Eventually, when it comes to Scenario 6, the workload distribution is concentrated more on one of the SP.

Table 3.1: Scenarios of Workload Assignment to SP

Scenario	Workload assign to each SP				
	SP1	SP2	SP3	SP4	SP5
1	20%	20%	20%	20%	20%
2	30%	20%	20%	20%	10%
3	30%	30%	20%	10%	10%
4	40%	30%	10%	10%	10%
5	50%	20%	10%	10%	10%
6	60%	10%	10%	10%	10%

3.2.2 Computation π by Numerical Integration

$$f(x) = \frac{4}{1+x^2} \tag{3.1}$$

The value of π can be calculated by formulating the area under the graph of equation (3.1) from the domain of 0 to 1 or one can simply write it in the form of an integral equation (3.2). Fig. 3.4 illustrates the graphical representation of the graph $f(x)$.

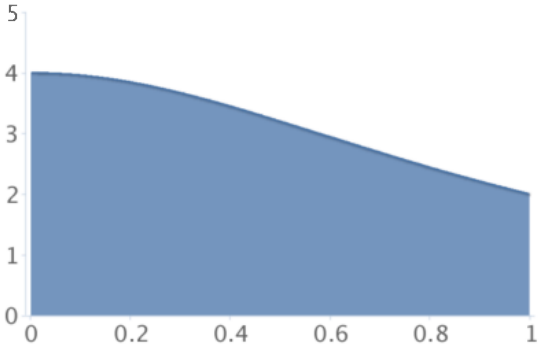

Figure 3.4: Visual Representation of Function $f(x)$

$$\int_0^1 \frac{4}{1+x^2} dx = \pi \tag{3.2}$$

In order to compute the area under the graph of (3.1) as illustrated in Fig. 3.4, the domain of the equation (3.1) is break into multiple smaller intervals then, the area of each interval is summed. The more intervals between the domain of (3.1) (i.e. from 0 to 1), the more precise is the value of π, leading to more computing resources to compute. The area of each interval is computed by multiplying the width and the height of each interval. The width of each interval can be obtained by dividing the size of the domain (i.e. 1), over the total number of intervals. The height of each interval is calculated by applying the *midpoint* value of an interval into function (3.1). This technique is also known as the midpoint approximation technique. As a conclusion, the computation of π numerical integration is used in this experiment is mainly due to that *i*) it can break in to multiple smaller pieces, *ii*) each piece can be computed independently and thus, it is possible to perform parallel processing, and *iii*) it requires more computing power as the number of interval increase.

3.3 Heuristic Neighbour Selection Algorithm

The previous section shows the importance of workload distribution in a structured network. This section, the problem is described in a heterogeneous environment, whereby there is no structured network. In a heterogeneous computing environment, nodes are geographically dispersed and vary according to their hardware specification which may cause load imbalance between nodes. Load imbalance between nodes are very likely to occur as compared to structured network described earlier. The load imbalance between nodes in a distributed system can lead to performance degradation. This section describes the proposed load balancing algorithm, *HNSA* enhanced from *SIDDLB* approach. *HNSA* is a decentralised dynamic load balancing algorithm where the decision of job distribution is based on the current state information of the system. Besides that, with regards to the decentralised approach, the job distribution is formulated by several load balancers. Lu et al. (2006); Nandagopal et al. (2010, 2011) have shown that the decentralised approach can prevent the overwhelming of communication messages at a single load balancer (centralised) that could lead to single point of failure as the system size increase. On the other hand, the trade-off by using decentralised approach is the increase of overall communication messages in the system. With regards

to this, by employing the partial information technique described by Lu et al. (2006); Nandagopal et al. (2010, 2011) could minimise the overall communication messages.

3.3.1 Load Balancer Model

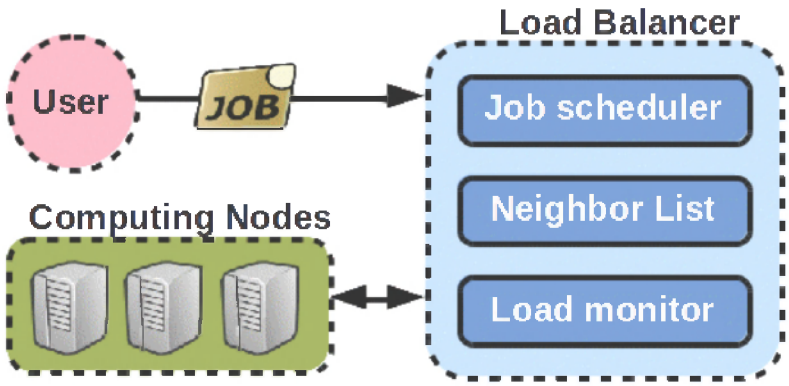

Figure 3.5: Decentralised Dynamic Load Balancing System Model

When there are idle or lightly loaded nodes and overloaded nodes in a distributed system, the system is not fully utilising its computing capability. Consequently, the overall response time of jobs in the distributed system increase. The role of a load balancer is to optimise the utilisation of resources in a distributed system and minimise the overall response time. Fig. 3.5 illustrates the load balancer model for *HNSA* that generally composed of 3 main modules: job scheduler that schedules the arrival jobs, neighbour list that keeps track of the load information of other load balancers, and load monitor to monitor the load of its own computing resources. The job scheduler is where the dynamic load balancing algorithm lives in and it works closely together with the scheduler. A dynamic load balancing can be explained in four different policies (Casavant & Kuhl, 1988; Mukhopadhyay et al., 2010): *information policy*, *transfer policy*, *location policy* and *selection policy*.

Upon a job arrives to a load balancer, the job scheduler determines an optimal location for the job to be executed. This decision is coined as the job distribution decision which is the decision to either schedule the job for local execution or remote

execution. The formulation of this decision is based on the load status of a load balancer. Load status determines whether or not a load balancer is lightly or heavily loaded. If a load balancer is lightly loaded, it is considered as being under utilised. Otherwise, the load balancer is considered as overloaded. Knowing the load status, a load balancer then, can determine whether or not it has been overloaded or lightly loaded. This method and process of determining the load status is known as the *transfer policy* of a dynamic load balancing algorithm.

In a decentralised dynamic load balancing environment, there are at least two or more load balancers are working together to formulate a job distribution decision. Hence, the acquisition of load information from other load balancers is an important factor to determine the load status of one load balancer. This is where the *information policy* comes along. It mainly outlines i) what are the information that are required to formulate the load status of one load balancer, and ii) how to obtain such information. Besides obtaining the load information, determining who are the neighbours and how to obtain their load information will also affect the formulation of the decision. The formation of neighbours and the method to obtain their load information are explained further in section 3.3.2.

After the load status of a load balancer has been identified as heavily loaded, a node from the neighbour list will be selected to relief some of its jobs. This is where the *location policy* of a dynamic load balancing algorithm determines which node should be selected to execute the job. After a neighbour has been selected, the overloaded load balancer will select a job for remote execution. The process for a load balancer based to select a job for transfer it to the designated node is written in the *selection policy*. Fig. 3.6 illustrates the general flow of the system and the usage of these policies in a dynamic load balancing algorithm.

3.3.1 (a) Notations

This following describes some of the common notations used throughout this thesis to determine the proposed *HNSA*:

- Let N be the set of all nodes
- Let J be the set of all jobs
- Let $Nbor_i \subset N$ be the set of neighbours for node i
- Let $NborSecond_i \subset N$ be the secondary set of neighbours for node i
- Let PWD_i be the computing power of node i
- Let λ_i be the average job arrival rate of node i
- Let β be the load tolerance level

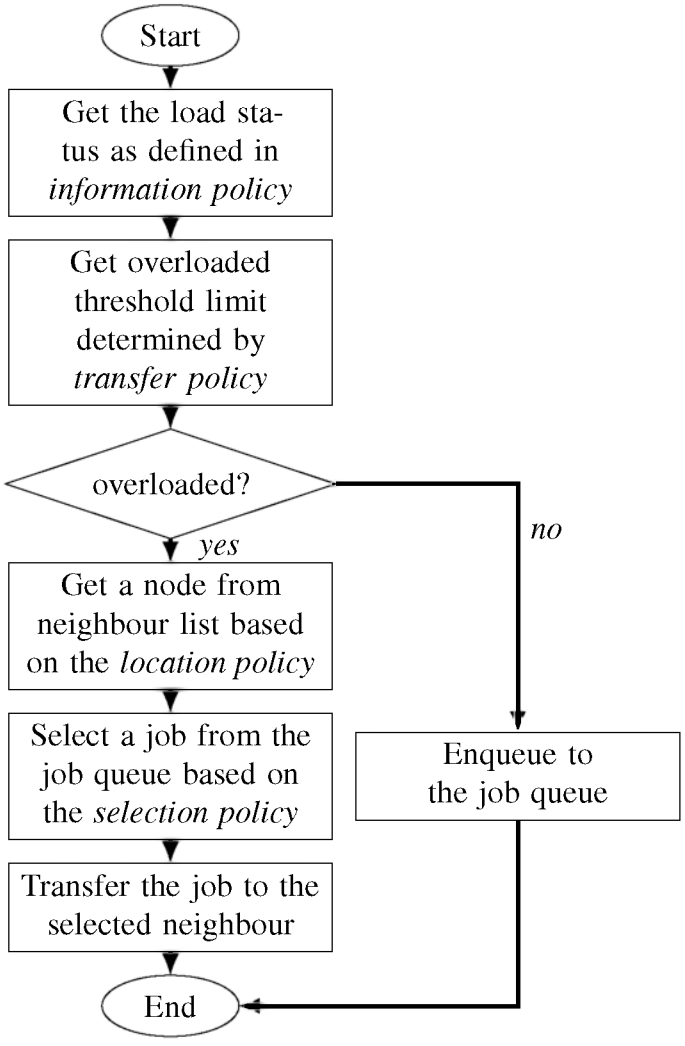

Figure 3.6: General Flow of a Load Balancing Algorithm

Algorithm 1 Information Update Process, *UpdateNbor*

Inputs:

1: $r \in N$ – the receiver
2: $s \in N$ – the sender
3: $nodes_s \subset N$ – set of nodes received from load balancer s

Steps:

1: $L \leftarrow (Nbor_r \cup NborSecond_r) + r$
2: **for all** $a \in nodes_r$ **and** $a \in L$ **do**
3: **if** $s.time_a > r.time_a$ **then**
4: $r.load_a \leftarrow s.time_a$
5: $r.time_a \leftarrow s.time_a$
6: **end if**
7: **end for**

Figure 3.7: Information Update Process Algorithm

3.3.2 Information Policy

Load information sharing is an important process in dynamic load balancing algorithm, in which a load balancer needs the information to formulate the job distribution decision. Throughout this research, the properties of load information contained by each load balancer are notated as follows, $\forall n \in N$:

- $load_n$ – The load index of node n which describes as the time required to complete all jobs in the job queue of node n.
- $time_n$ – The local time of node n which is the last updated time of the load index, $load_n$.

Hence, the neighbour list of a load balancer would contain these information regarding the other load balancers, to formulate the job distribution decision. This list needs to be constantly updated in order to increase the accuracy of the job distribution. A technique named mutual information feedback is used to update the list, which allows a load balancer to share some of its neighbours' load information with another load balancer (Nandagopal et al., 2010, 2011) by piggybacking onto a job transfer message. That

is, the load information exchange is take place at the event of a job transfer from one load balancer to another. Before a load balancer can transfers a job to its neighbour, it will piggyback the load information of itself and at least one or a small amount of its neighbours to the job transfer message. At the receiver side, upon it receives the job transfer message, it acknowledges the sender by with the same action. While a load balancer is updating its neighbour list, it will only select those neighbours that found in its list to update, and provided the $time_n$ value is higher than the one in its list. If a node's load information is not found in ones neighbour list, it will not be added into its neighbour list, instead the load balancer will have to discard that particular load information. This is to ensure that the neighbour list consists of those nodes that of higher computing capabilities. The Algorithm 1 outlined in Fig. 3.7 is the steps for a load balancer to update its neighbour list only if there is a matched of similar neighbours with the latest load information. In order to identify which is the latest load information of a load balancer, $time_n$ is used to determine when was the last update time of the $load_n$ of a neighbour n. Furthermore, this information exchange process will only take place during a job transfer, that is also upon the load imbalance of a system.

3.3.3 Neighbour List

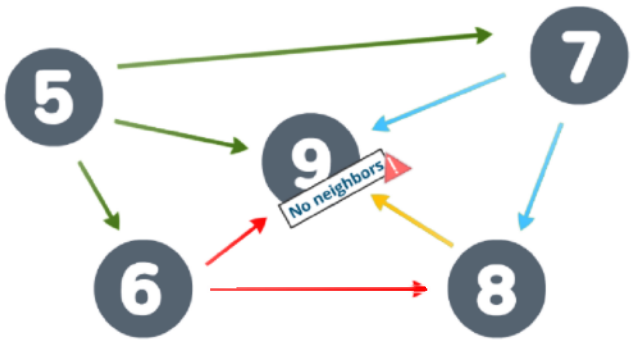

Figure 3.8: Scenario of a Node With an Empty Neighbour List

The construction of neighbour list is based on reliable nodes, which is similar to the neighbour construction model proposed by (Nandagopal et al., 2011, 2010). The list of reliable nodes refers to the list that only contains nodes which are likely to be able to accept and compute any jobs sent from the sender. So, the sender do not need to collect and store the load information of the nodes which are unlikely to accept the jobs from the sender.

During the system start-up, each load balancer will construct their own neighbour list based on two criteria: i) computing power or computing capabilities (these 2 terms will be used interchangeably), and ii) network delay . In this research study, the network delay will be assumed to be minimal or negligible, and hence, it will not be taken into account. As the end result, the neighbour list of a node *A*, would consists of those load balancers with their computing power (or computing capability) relatively high as compared to node *A*. By applying this construction model throughout the system, the load balancer with the highest computing power eventually does not have any neighbours. Fig. 3.8. illustrates such scenario where the load balancers are represented in circles and the number in the middle represents their computing power; the smaller numbered circle means having lower computing power as compared to a higher numbered circle. Besides that, the directed arrows pointing from a node *A* towards other nodes illustrate the (*has a*) relationship between node *A* and the other nodes, but not vice versa. For example, the neighbours of node 5 are nodes 6, 7, and 9. But, the neighbour of node 6 does not include node 5. The Fig. 3.8 shows that node 9 does not have any neighbours associate with it because it has the highest computing power as compared to the others.

As far the reliable nodes are concerned, the nodes received from the load information exchange as explained in the previous section, cannot be simply appended into the list if it is not found. Instead, it will be discarded. Again, the neighbour list in the highest computing powered nodes will be still remain empty, and it cannot offload jobs when it is overloaded.

3.3.4 Secondary Neighbour List

In order to solve the empty neighbour list, specifically the highest computing powered node, a secondary neighbour list, *NborSecond*, is then introduced to takeover the absence of the main neighbour list. The reason to have another secondary list for such nodes is to prevent them from being overloaded. The 2 main differences of between the main neighbour list and the secondary neighbour list are that: i) the secondary list able consists of unreliable load balancers, that is, the neighbour list of

a node A might include node B of which its computing power is relatively lower as compared to node A, and ii) the dynamic construction of a neighbour list at the runtime of the system.

Whenever a load balancer with the secondary neighbour list receives a job transfer request, from the mutual information feedback, a load balancer will include the load information of the sender, as well as some of the neighbours from the sender to its secondary neighbour list. If a neighbour has already exists in the secondary neighbour list, then the load information of regarding the neighbour will be updated. However, there are times when the neighbours in secondary neighbour list are not up-to-date and thus, the load balancer will formulate the job distribution decision based on an outdated load information. Since the information exchange process is bind together with the job transfer and such load balancer cannot simply transfer jobs to unreliable load balancers, the information update process cannot take place.

3.3.5 Load Prediction

$$F_i = \frac{load_i + (\frac{D_i}{\lambda_i} \times Largest(J))}{PWD_i} - D_i \tag{3.3}$$

The following describes the notations used in equation 3.3:

- Let F_i be the estimated time to complete all jobs queued in node i
- Let J be the set of all jobs
- Let $load_i$ be the load of node i (i.e. the time needed to complete all jobs at node i)
- Let D_i be the time difference between now and the last recorded time of the variable $load_i$ of node i
- Let PWD_i be the computing power of node i
- Let λ_i be the average job arrival rate of node i

- Let $Largest(J)$ be the largest job size in the set of J

In order to overcome the issue encountered in secondary neighbour list, whereby the formulation of job distribution decision is not optimised, a simple load prediction method is applied to estimate the current load information of the neighbours in the list. The load prediction is described in an equation 3.3, F_i, that predicts the $load_i$ for a neighbour i. The variable D_i in the equation refers to the time elapsed since $time_i$, which is also known as the time difference between $time_i$ and present. Besides that, λ_i refers to the average job arrival rate of 100 jobs and $Largest(J)$ refers to the largest job in the set of J. So, in addition to Algorithm 1, for those load balancers which their primary neighbour list is empty, they will be using Algorithm 2, load prediction approach, outlined in Fig. 3.9 to estimate the load of those nodes in secondary neighbour list upon every job arrival.

Algorithm 2 Load Information Prediction, *LoadPrediction*

Output: $L \subset N$ – a set of load balancers with their load information predicted
Inputs:

1: $n \in N$ – the load balancer that is currently executing this algorithm

Steps:

1: **if** $Nbor_n = \varnothing$ **and** $NborSecond_n \neq \varnothing$ **then**
2: **for all** $c \in NborSecond_n$ **do**
3: $load_c \leftarrow F_c$ from equation 3.3
4: $time_c \leftarrow$ current time
5: **end for**
6: **return** $NborSecond_n$
7: **end if**
8: **return** $Nbor_n$

Figure 3.9: Algorithm For The Prediction of The Secondary Neighbour List

3.3.6 Transfer Policy

Algorithm 3 Heuristic Neighbour Selection Algorithm (*HNSA*), *Schedule*

Output: $min \in N$ – the least loaded node

Inputs:

1: $n \in N$ – the load balancer that is currently executing this algorithm
2: $j \in J$ – the arrival job

Steps:

1: **if** $Nbor_n \neq \varnothing$ **then**
2: $\exists min \in Nbor_n$, where *min* possesses the least load
3: **if** $load_n > load_{min} + \beta$ **then**
4: **return** *min*
5: **end if**
6: **else**
7: $L \leftarrow LoadPrediction(n)$ from algorithm 2
8: $\exists min \in L$, where *min* possesses the least load
9: **if** $load_n > load_{min}$ **then**
10: **return** *min*
11: **end if**
12: **end if**
13: **return** *n*

Figure 3.10: Algorithm To Select The Optimal Neighbour

Transfer policy outlines the steps to determine the load status of a load balancer among other load balancers (i.e. overloaded, or lightly loaded). In *HNSA*, the load status of a node is determined through a fixed threshold which is by comparing the load index of itself and its neighbours' load index that should not exceed the tolerance level β. The usage of this approach is similar to *SIDDLB* approach and it produces a good result in certain scenarios. This will be further discussed in chapter 4.

On top of that, with regards to the secondary neighbour list, the proposed approach *HNSA* has a different method of determining load status for those nodes which their *Nbor* is empty. Especially the highest computing powered node. Since the information exchanged process takes place during a job transfer, the update of information in the secondary neighbour list is infrequent and hence, these nodes should not simply transfer to any lower computing powered nodes based on these load information. Because of this, the load prediction method is used for the secondary neighbour list to assist the load balancer in determining a more suitable load status.

If the predicted *time* value of a neighbour is smaller than the load balancer that is formulation the decision, the load balancer is considered to be overloaded. That is because the predicted value also specifies the amount of time required to complete all jobs in the job queue. The Algorithm 3 depicted in Fig. 3.10 outlines the transfer policy of the proposed algorithm *HNSA* which uses 2 different methods of determining the load status of a load balancer. Furthermore, the Fig. 3.11 illustrates the flow of the *HNSA* from receiving new job or job transfer to the end of the scheduling decision.

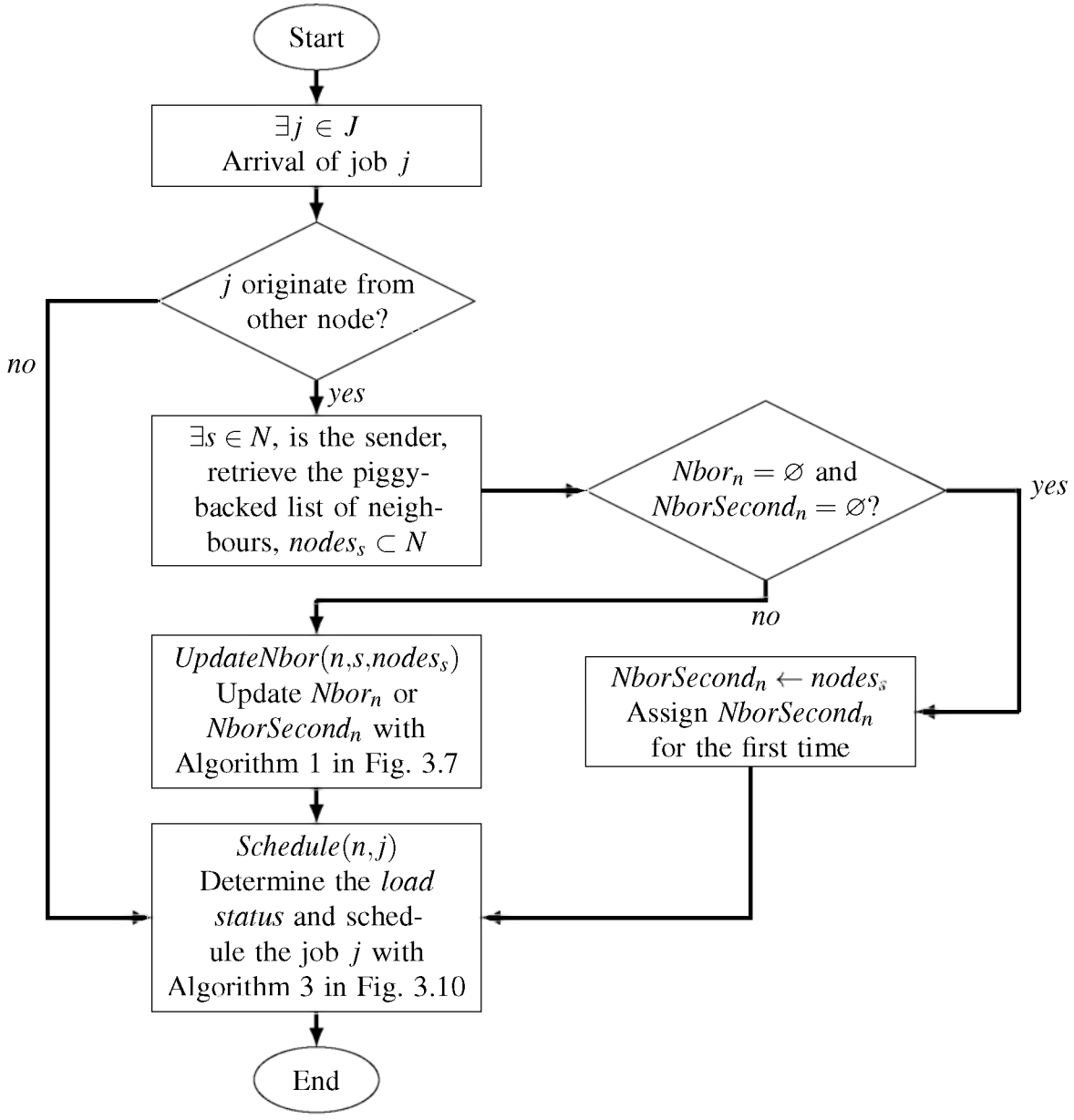

Figure 3.11: Process Flow When Job Arrives At Node

3.3.7 Simulation Model

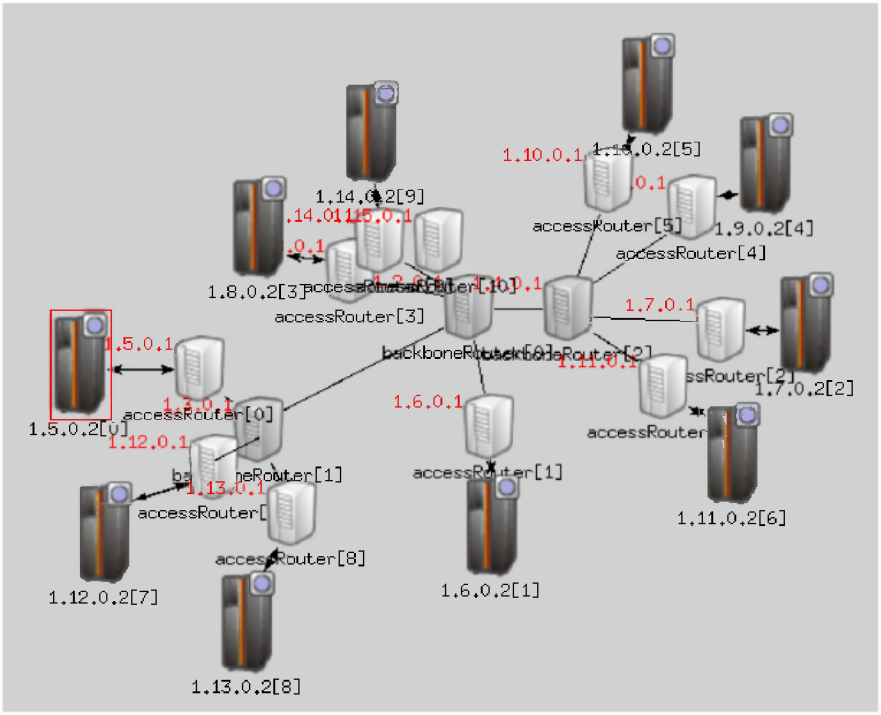

Figure 3.12: Network Diagram of 10 Load Balancers

This thesis focused on the study of the neighbour list, which refers to the *information policy* that describes the approach of obtaining load information from other load balancers, and input into the *transfer policy* to determine the load status of other load balancers. To study and measure the performances of the proposed *HNSA* method, a simulations study will be conducted to analyse the overall average and the variation of response time of a job for each load balancer. The results will be compared against *SIDDLB* algorithm that is also focusing on the *information policy* towards load balancing.

Simulation will be simulated in a network of total 10 load balancers with a maximum network diameter of 6; i.e., the furthest distance between a 2 load balancers is at most 6 hops away. Fig. 3.12 illustrates the network of 10 load balancers in this study. In order to simulate heterogeneity of computing nodes, each load balancer will be distinguished by their computing power. For simulation purposes, each load balancer will be identified by a range of identifiers from 1 to 10, where as 1 being the lowest and 10 being the highest computing power. The computing power of a load balancer will be

measured in terms of job-size per second (*Job-size/sec*). That is, given a job of size 10, it will only need 1 second to compute the job for a node that is 10 job-size per second. The Table 3.2 depicts the range of load balancers and their computing capabilities used throughout this simulation study. The variation of computing power in Table 3.2 depicts the heterogeneity of the distributed system.

Table 3.2: Various Computing Powered Load Balancers

Node ID	1–2	3–5	6	7–8	9–10
Computing Power (*Job-size/sec*)	10	20	30	40	50

3.3.7 (a) Job Model

Each of the scenario will execute 10,000 jobs before ending the simulation. The job model used in this simulation uniformly distribute job size ranging from 20 to 99. The value of a job size is a natural number greater than zero. Job value represents the complexity of a job; the greater the job size, the more computing power needed to compute the job. For example, a job size of 50 will be taking 5*s* to compute for a node that is capable of computing at the speed of 50 *Job-size/sec*. Each job will be created at any one of the nodes at a time. That means, no jobs will be created to more than 1 node at a time. Furthermore, the arrival time of a job at each node will be based on Poisson distribution with a mean deviation of 2*s*.

3.3.7 (b) Pour Jobs

There are various scenarios set-up in this simulation to study and analyse the algorithm in a heterogeneous environment. In order to simulate an overloaded node, *pour jobs* is introduced. Listed in Table 3.3, there are the 4 different scenarios set-up to simulate overloading nodes of various types of computing powered nodes. Again, the Node ID depicts the computing power of the given node; i.e. Node ID 1 has relatively low computing power as compared to Node ID 2. In order to intentionally overload a node, *pour jobs* simply create a huge number of jobs at that specific node at an instance of time. For example, at time t, create 300 jobs for node i. Eventually, the size of the job queue increases as well as the response time of the jobs in the queue of node

i. In the third column (**Number of Jobs**) of Table 3.3, is the number of jobs that will be created to overload at the designated load balancers which are listed in the second column (**Node ID**). This method of overloading a node is called as *pour jobs* and will be used throughout this thesis. Note that in scenario 1, there will be no event of *pour jobs* occurred in any of the nodes in the distributed system. This scenario is to analyse how will the algorithm reacts as there are no *pour jobs* event occur. As for scenario 2 and 3, the event of *pour jobs* will occur to the lower computing powered node, and higher computing powered nodes respectively. These 2 scenarios study how each of the node reacts when they are overloaded. Finally in scenario 4, both lower and higher computing nodes will expect a *pour jobs* event about the same time. In this simulation study, the event of *pour jobs* will only take effect after the nodes have created their first 50 jobs.

Table 3.3: Scenarios of *Pour Jobs* in Various Computing Powered Load Balancers

Scenario	Node ID	Number of Jobs
1	-	-
2	1	300
3	9, and 10	300
4	1,9, and 10	300

CHAPTER 4

RESULTS AND DISCUSSION

This chapter discusses the results obtained from the simulation studies described in chapter 3. In section 4.1, the result of the significant impact on various sizes of workload towards the performances of the distributed system is discussed. This section, first shows the importance of parallel distributed computing follow by the need of load balancing in distributed system. Finally, in section 4.2, the results obtained from the proposed dynamic load balancing algorithm, namely *HNSA*, is discussed. Furthermore, this section also discussed how the proposed algorithm can minimise the response time of a job and also maximise the utilisation of the distributed system.

4.1 Effect of Workload Imbalance

This section discusses the results from the importance of parallel computing and the impact study of various sizes of workloads towards a distributed system presented in section 3.2. In this study, there were two experiments simulated: i) *Single Processor* to show the importance of parallel computing, and ii) *Distributed Processor*s to show the impact of different sizes of workloads towards the parallel distributed system. In both simulations, the computation of the value $pi(\pi)$ was used as the computing task to be computed. The response time was measured in both simulations in order to measure and analyse the total time taken to compute the value $pi(\pi)$. With regards to *Distributed Processor*s, the waiting time of SP and MP were also measured to study how much the time were spent for them whilst waiting for the other nodes to complete the computation of the value $pi(\pi)$. Besides that, each of the computing time taken by P to compute its own portion of task were measured to analyse the actual time spent. Several scenarios have been simulated by varying the different amount of workload to each SP. Furthermore, the results of these scenarios were presented to discuss the importance of load balancing.

4.1.1 *Single Processor* Simulation

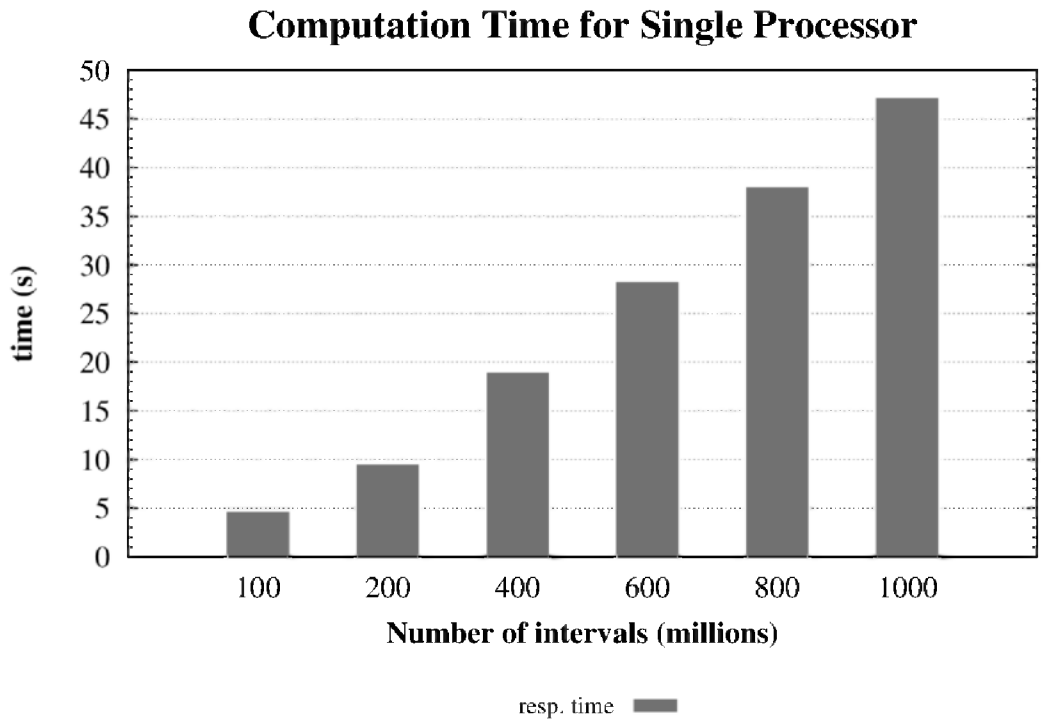

Figure 4.1: Overall Computation Time for *Single Processor*

In this simulation, the importance of parallel computing has been identified. Fig. 4.1, illustrates the increasing need of computing time as the number of intervals increases. The result has shown that the increasing number of intervals between the domain of 0 and 1 requires more computing time to execute the value $pi(\pi)$. This is because the accuracy of the value $pi(\pi)$ improves proportionally to the number of intervals between 0 and 1, and as well as the computing time. However, parallel computing can be applied to speed up the process. In the *Distributed Processor* simulation where distributed computing environment is set-up to perform parallel computing task, the overall results obtained from it was less than $5s$ to compute the value $pi(\pi)$. However, in a distributed system, the load of nodes are changing all the time and it may affect the response time and eventually the performances of the system. The next section discussed the significant impacts for nodes that have various amount of workload towards the performance of a distributed system.

4.1.2 *Distributed Processor* Simulation

The main focus of this *Distributed Processor* simulation to analyse the impact of different sizes of workloads. In Table 3.1, the assignments of workloads for 6 scenarios was used to set-up this simulation and Fig. 4.2, Fig. 4.3, Fig. 4.4, Fig. 4.5, Fig. 4.6, and Fig. 4.7 show the results from those 6 scenarios described in Table 3.1. The results are denoted as follows:

- *resp. time* – response time
- *MAX(COMP)* – maximum computation time
- *MIN(COMP)* – minimum computation time
- *MAX(COMM)* – maximum communication time
- *MIN(COMM)* – maximum communication time
- *MP AVG(WAIT)* – average waiting time of MP
- *SP AVG(WAIT)* – average waiting time of SP

The results illustrate that in all the scenarios and for the selected workload sizes (computation using 100,000,000 to 1,000,000,000 intervals), the overall response time is in the region of $2.5s \pm 0.5s$. This is a far-off from the theoretical speed-up of using 100 times more processors. The speed-up obtained from the scenarios ranges from about 2.5x to 15x.

By varying the distribution of workloads to the SPs based on scenario 2, 3, 4, 5 and 6, the theoretical speed-up that is expected will be governed by the SP with the heaviest workload. It is observed that the maximum computation time for processor, *MAX(COMP)* not only increase with the number of intervals used to compute the value of but also increases with each scenarios. This is because the SP with the heaviest workload in each of the scenario increases from 20% (scenario 1), 30% (scenario 2 and

3), 40% (scenario 4), 50% (scenario 5) to 60% (scenario 6). However, it is also observed from the simulation results obtained, the *MAX(COMP)* is a minor contributor to the overall response time of each scenarios. From the results, it is evident that the overheads from the communication and the waiting times at the MP and SPs significantly impact the scalability of the distributed system. The waiting times on SPs and MP is due to having to wait for the latest (slowest) nodes and SPs to send back their computed results respectively.

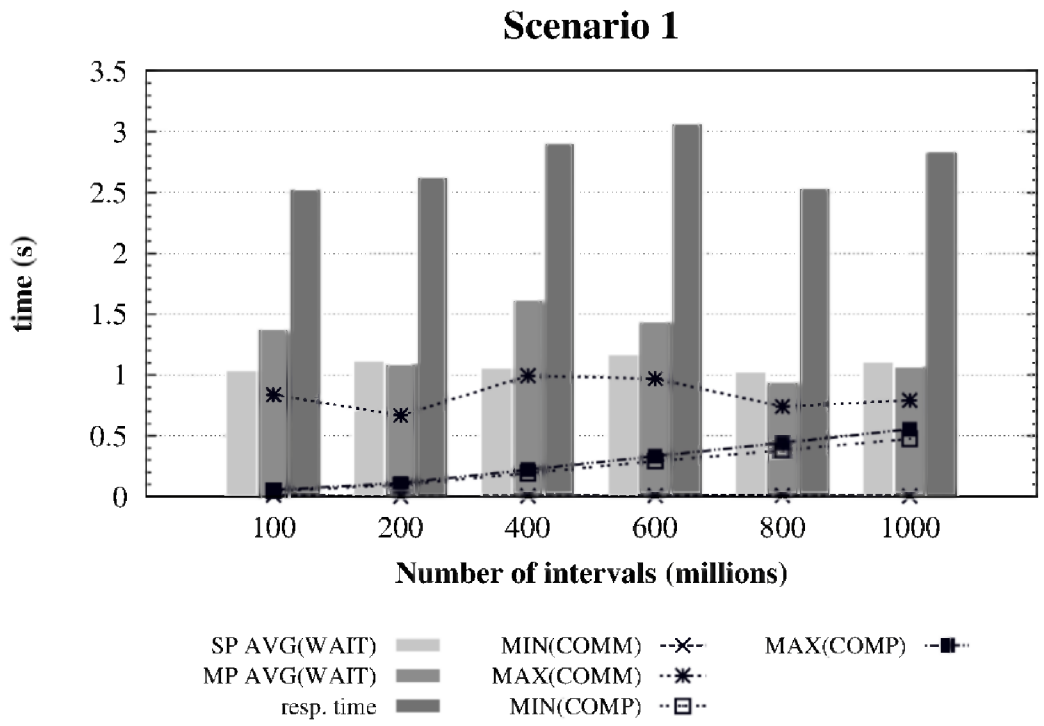

Figure 4.2: Overall Computation time of Scenario 1 for *Distributed Processor*

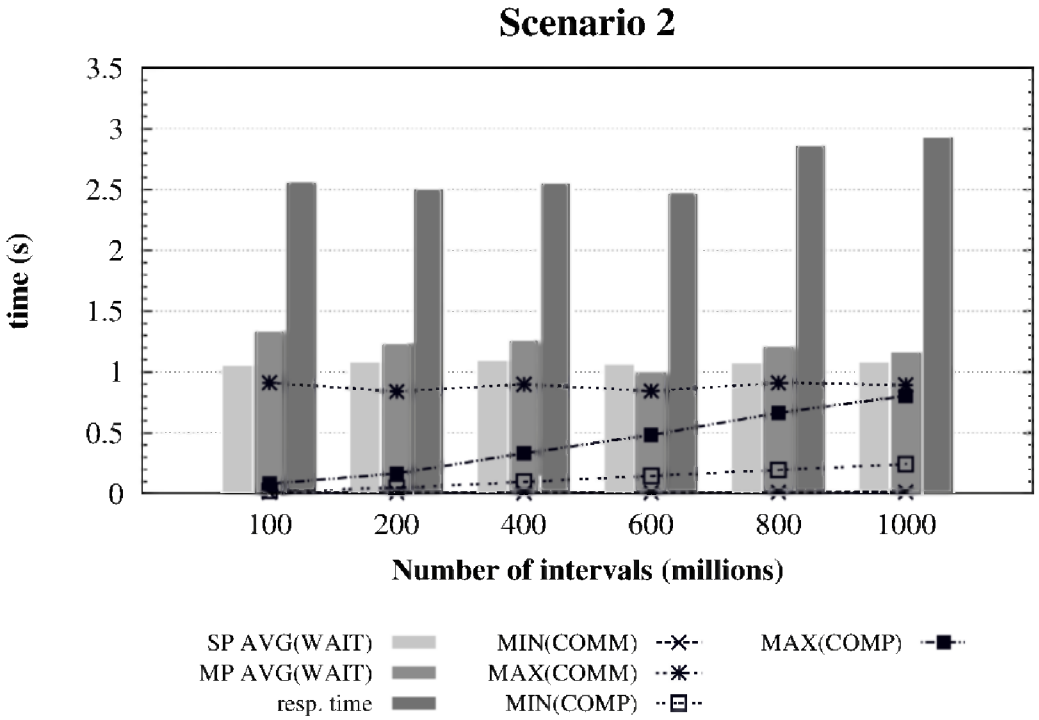

Figure 4.3: Overall Computation Time of Scenario 2 for *Distributed Processor*

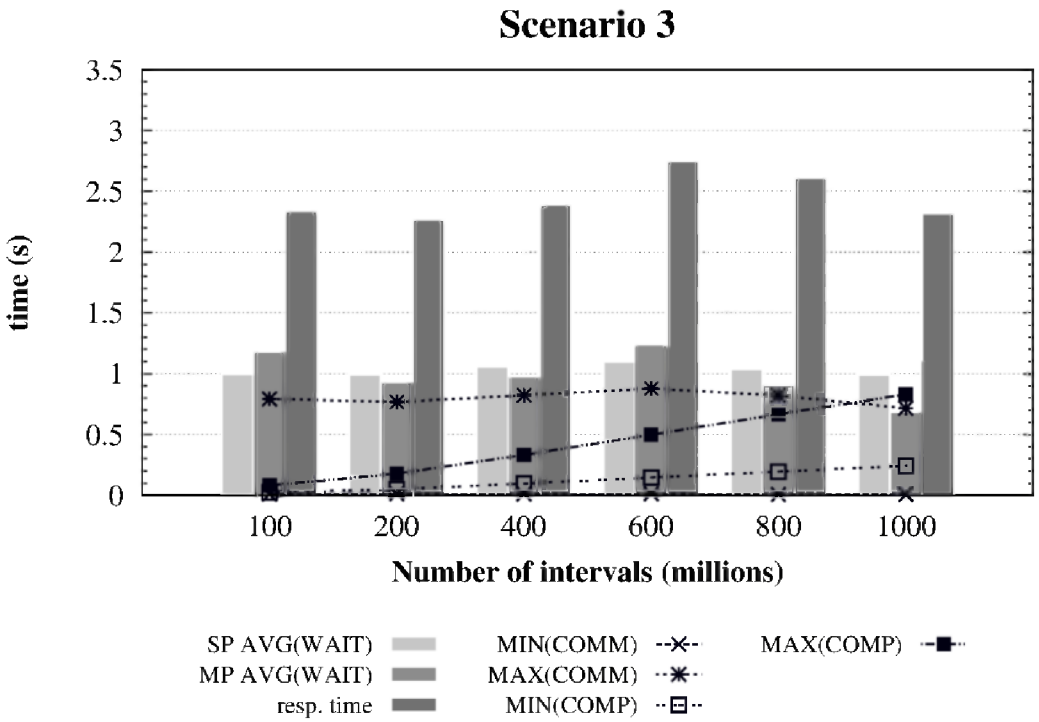

Figure 4.4: Overall Computation Time of Scenario 3 for *Distributed Processor*

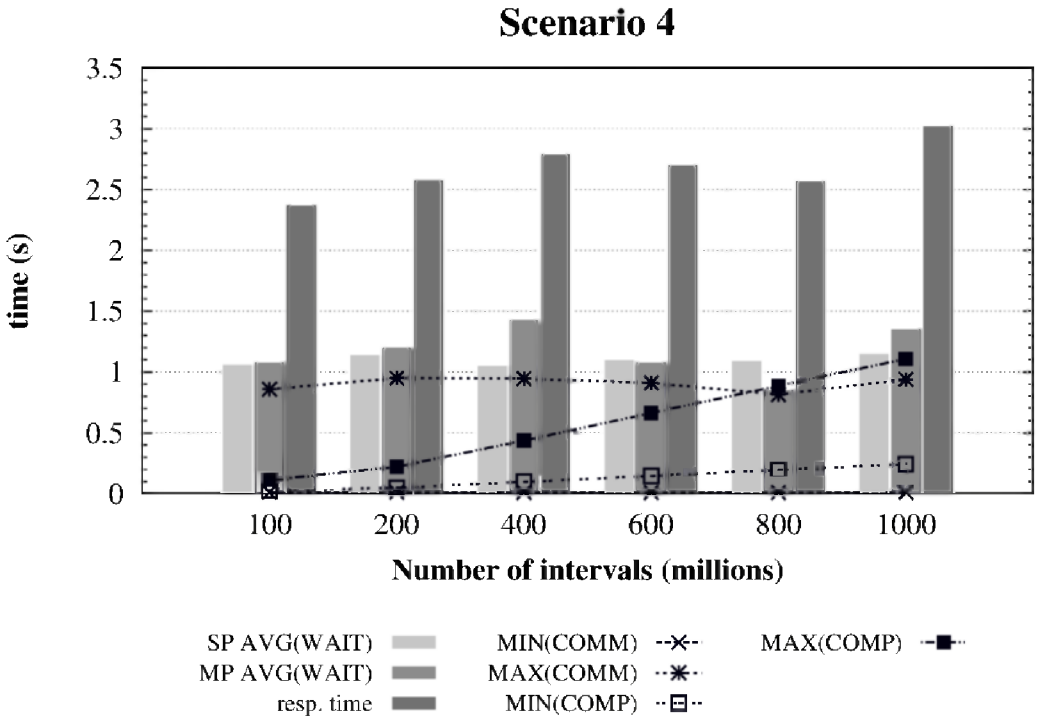

Figure 4.5: Overall Computation Time of Scenario 4 for *Distributed Processor*

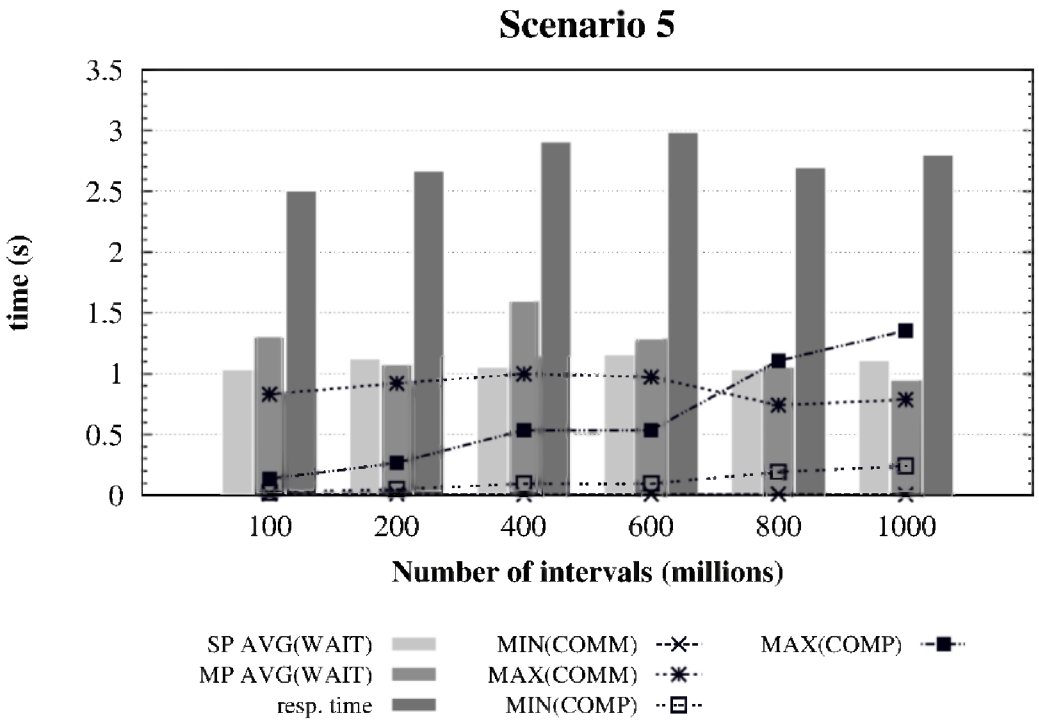

Figure 4.6: Overall Computation Time of Scenario 5 for *Distributed Processor*

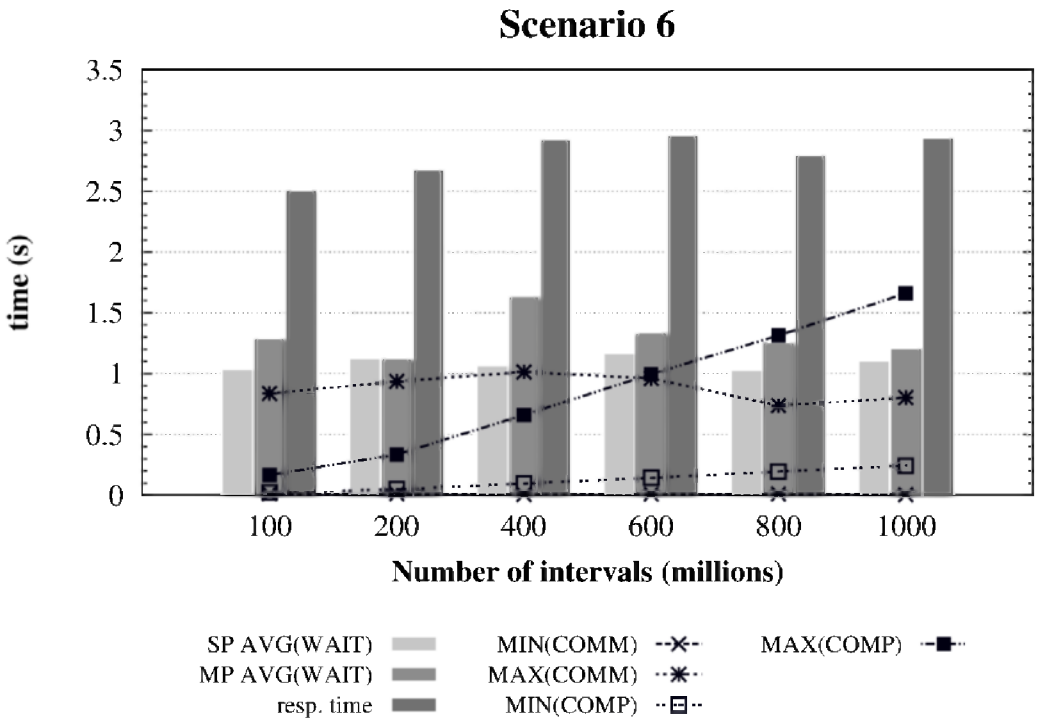

Figure 4.7: Overall Computation Time of Scenario 6 for *Distributed Processor*

4.2 Decentralised Dynamic Load Balancing

4.2.1 Effect on Response Time

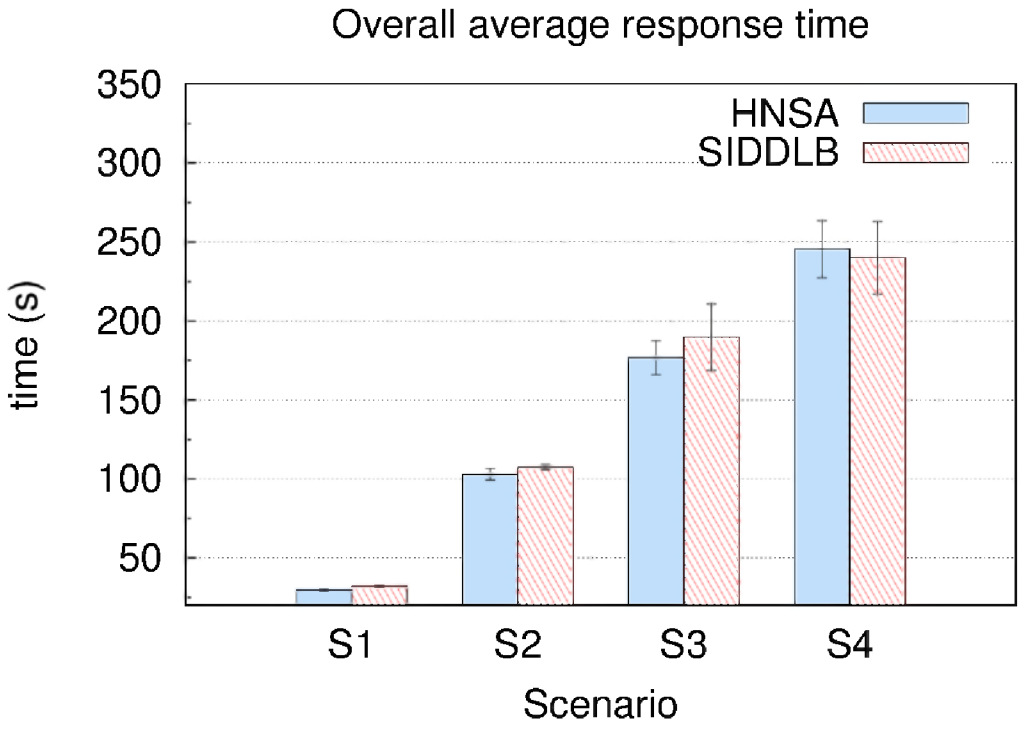

Figure 4.8: Overall Average Response Time for Each Scenario

The performance of the proposed approach, *HNSA*, was evaluated using the simulation set-up described in section 3.3.7 and it was compared against the method proposed by Nandagopal et al. (2010, 2011) *SIDDLB*. In order to analyse the performance of the load balancing algorithm, the average response time for each node was recorded. The response time is the time taken for a node to complete a single job, whereas the average response time of a node is the average of all the response time of the jobs computed in the node. In addition to that, the variation of response time for each job was also taken into account and thus, standard deviation of the response time is used. The smaller the value of the standard deviation, the better the load balancing algorithm reacts to the distributed system. Fig. 4.12 and Table 4.1 show the overall average response time and standard deviation for each scenario. Fig. 4.8, 4.9, 4.10, and 4.11 show the average response time of each job for each node from the 4 scenarios described in Table 3.3.

Table 4.1: Overall Average Response Time

Scenario	*HNSA* response time (*s*)		*SIDDLB* response time (*s*)	
	avg.	**stdev.**	**avg.**	**stdev.**
S1	29.63	0.75	32.14	0.91
S2	102.89	3.65	107.27	1.68
S3	176.69	10.72	189.76	21.03
S4	245.38	18.29	239.94	22.95

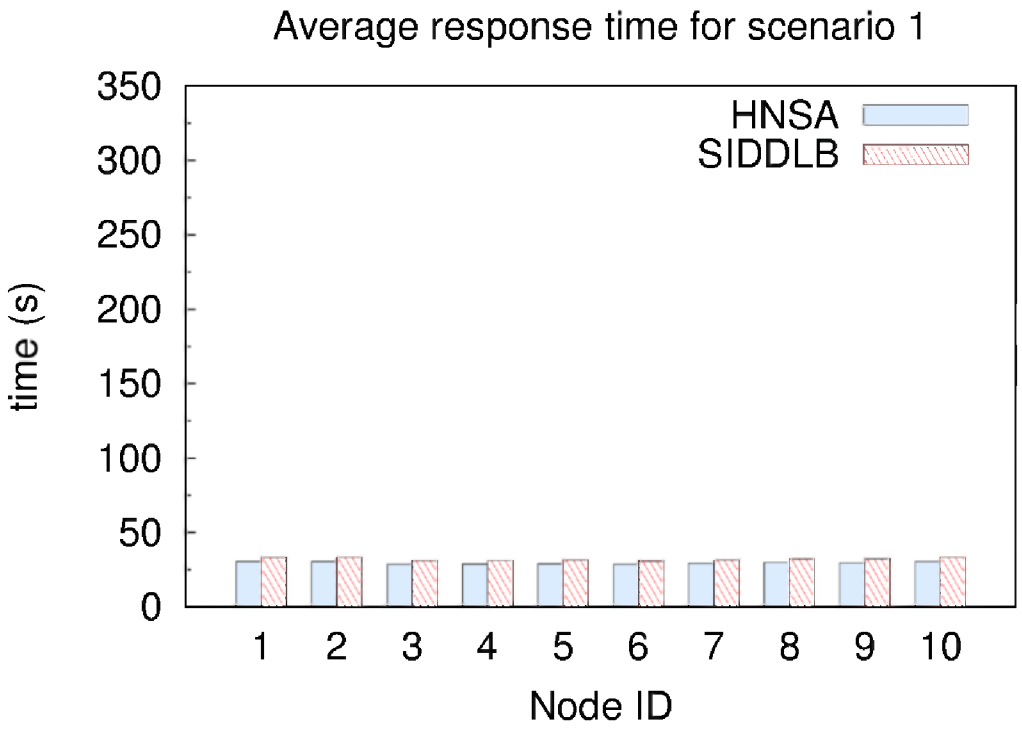

Figure 4.9: Average Response Time for Scenario 1

In Fig. 4.8, the overall response time for all of the scenarios show some improvement over *SIDDLB* except for scenario 4 which result in a small rise in the average response time. With regards to scenario 4, where the event of *pour jobs* take place at the lowest and highest computing powered nodes, the difference of average response time of both approaches is only 5.44*s*, which means *SIDDLB* is about 2% faster than *HNSA*. However, the result from the standard deviation shows that the variation of jobs in *SIDDLB* is about 20% higher as compared to *HNSA*. This means that the overall decision of distributing jobs across nodes in this scenario for *HNSA* is still better. Besides this scenario, in scenario 2, it is noted that the althought the average response time *HNSA* performs better than *SIDDLB*, the variation of jobs results in around 50% more than

SIDDLB. This is because, in this scenario, the advantage of *SIDDLB* is to be able to re-distribute jobs from the overloaded nodes, which have higher computing powered nodes in their neighbour list. So, whenever the a node is overloaded, it will transfer a job to a higher computing powered node and eventually, the highest computing powered node has to accept the job. In contrast to *SIDDLB*, *HNSA* gives the highest computing powered node the chance to formulate job distribution decision. For example, in scenario 3, whereby the *pour jobs* event occurred at the highest computing powered node, both the average response time and variation of jobs in *HNSA* result better than *SIDDLB*.

Looking at scenario 1 in detail, in which the event of *pour jobs* did not take place, the result shows that the standard deviation of response time for both approaches are relatively small, generally less than $1s$. Specifically, the standard deviation of response time of each job for *HNSA* approach and *SIDDLB* approach are $0.75s$ and $0.91s$ respectively. There was only slight improvement over *SIDDLB* approach which can be considered as insignificant. In fact, *SIDDLB* approach itself should be able to handle the normal situation. Although from Fig. 4.9, each of the node using *HNSA* approach results in lower average response time. The purpose of this scenario is to analyse whether if the *HNSA* is also able to fairly distribute the jobs across the nodes. As a conclusion for this scenario, both approaches were able to balance the load among each other.

In scenario 2, the event of *pour jobs* took place at the lowest computing power node, specifically the Node ID 1, with an additional of 300 jobs. From the result illustrated in Fig. 4.10, for each node that used *HNSA* approach has resulted in lower average response time as compared to *SIDDLB*. However, the overall average response time from both approaches has rise. This is because of the additional 300 jobs created at an instance of time. During that time, for both approaches, Node ID 1 forwards some of the jobs to higher computing power nodes and eventually builds up their queue. This eventually will affect the response time of other nodes as well, thus, the increase in an overall response time. Note that there is a slight drop of average response time in Node

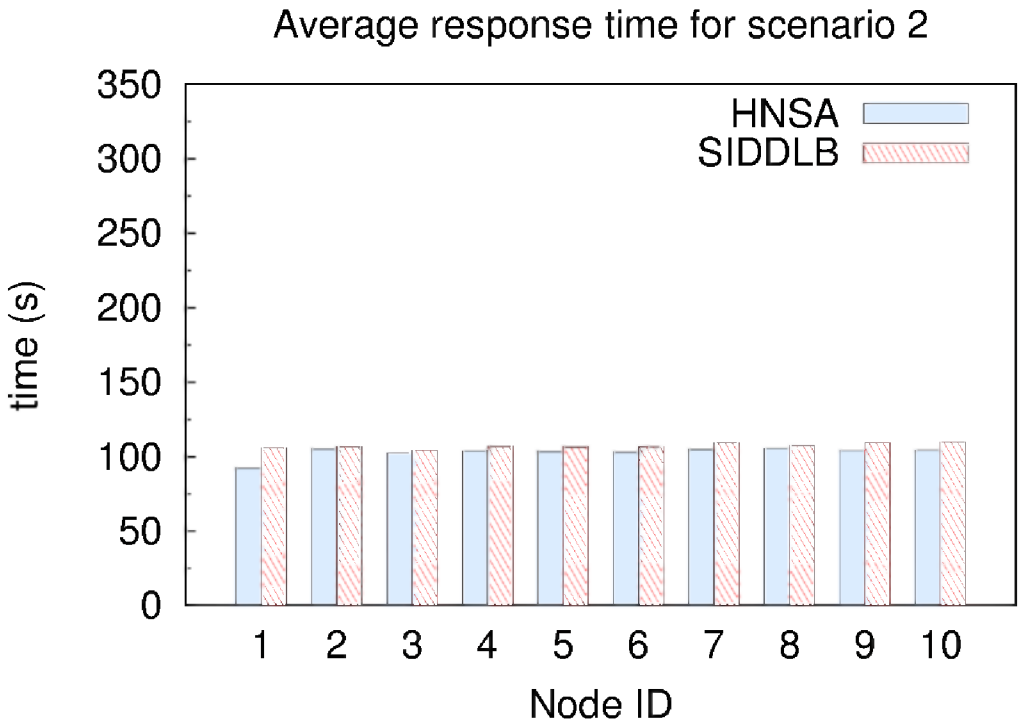

Figure 4.10: Average Response Time for Scenario 2

ID 1 for *HNSA*. Due to the jobs that were transferred up to highest computing powered node, such node in *HNSA* uses the secondary neighbour list to predict other neighbours load and formulate job distribution decision based on such information. Hence, due to the load prediction method which highly depends the $time_n$ of a node n to predict the $load_n$, has probably predicted that Node ID 1 is overloaded.

In scenario 3, instead of overloading the lowest computing powered node, the *pour jobs* event occurred at Node ID 9 and 10, which both of them have the highest computing powered. The objective of this scenario is to verify that in *HNSA* approach, it is able to predict the load of other lower computing powered nodes and able to re-distribute the job to other feasible node. The result shows that the standard deviation of response time for *HNSA* approach and *SIDDLB* approach are $10.72s$ and $21.03s$ accordingly. *HNSA* approach has shown an 50% of improvement against *SIDDLB* in terms of variation of response time. This has shown that in *SIDDLB* approach, although higher computing power nodes are overloaded, they will still accept any jobs that are from lower computing power nodes. This is because in *SIDDLB* approach, each

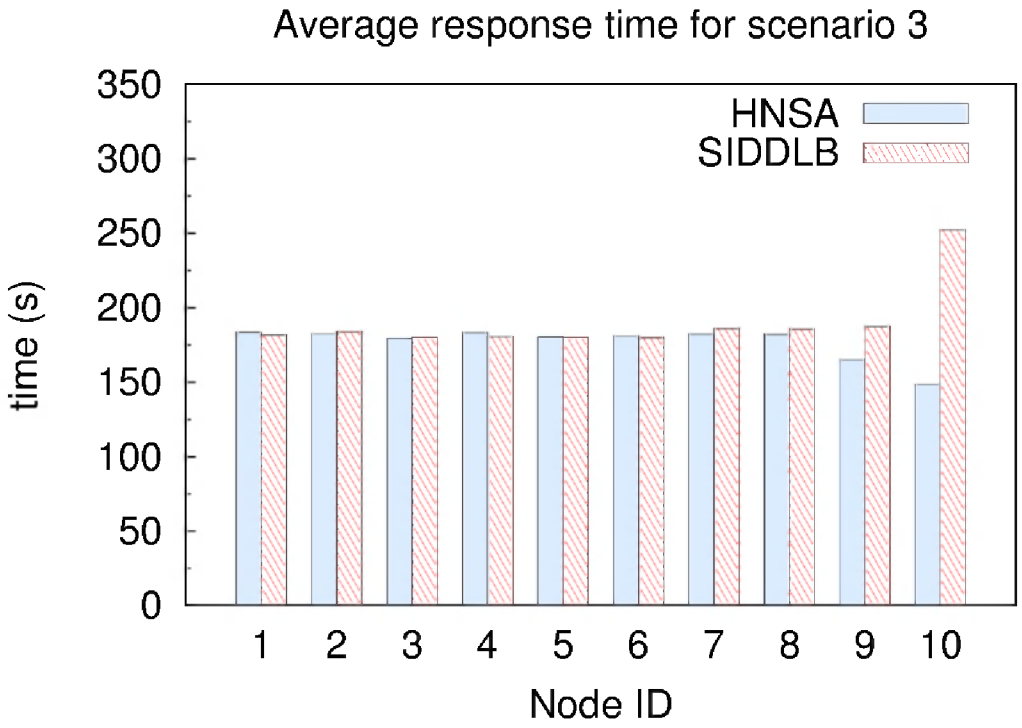

Figure 4.11: Average Response Time for Scenario 3

node will only consists of higher computing powered nodes. Hence, if a job has been escalated until the Node ID 9 or 10, such nodes will assume that there are no other nodes in the distributed system can execute this faster than they do. On the other hand, in *HNSA* approach, it consists of another secondary neighbour list for nodes that do not have the primary neighbour list. This list is mainly to learn the existence of other nodes in the system and predict their loads from time to time. Job will be re-distributed to lower computing power node if it is predicted to be a feasible node. However, as can be seen in Node ID 9 and 10, both recorded slightly lower response time than the other nodes. This is due to the static transfer policy, which does not incorporate threshold adjustment and result in some unnecessary migration of jobs. Nevertheless, it can still be concluded that the prediction of load in secondary neighbour list helps to formulate a better job distribution decision.

Finally, in the last scenario 4, which is to test the awareness of the prediction algorithm if both lowest and highest computing powered nodes are overloaded, specifically Node ID 1, 9 and 10. The result of *HNSA* approach shows that it only lower down

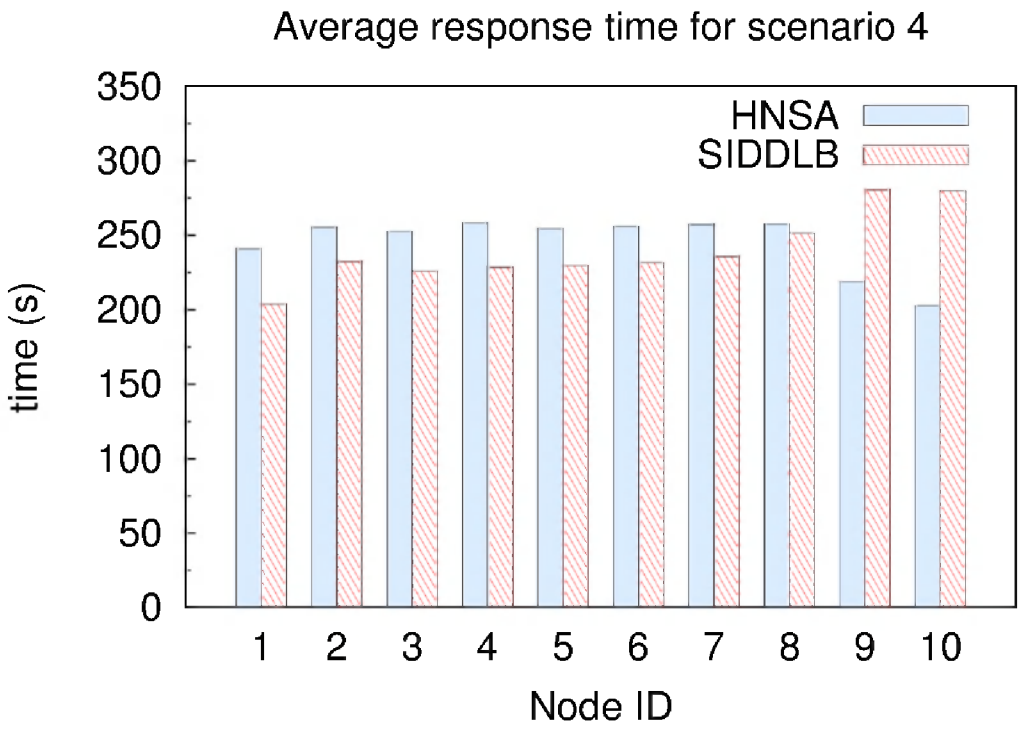

Figure 4.12: Average Response Time for Scenario 4

the load in Node ID 9 and 10 but not in *SIDDLB*. Throughout the load from Node ID 1 to 8, *HNSA* has resulted an increasing average response time. This means that Node ID 9 and 10 were not aware that other nodes in the distributed system is also experiencing a rise in average response time. This is because during the *pour jobs* event in Node ID 9 and 10, the job arrival rate also affect the load prediction algorithm to learn that the arrival rate of the job has been constantly $0s$ for over 300 jobs and it did not realise that Node ID 1 is also encountering *pour jobs*. On the other hand, for *SIDDLB*, the load will only be increasingly rise from Node ID 1 to 10 because the job keeps on escalating up until it reaches to the highest computing powered node. Nevertheless, the overall average response time for *HNSA* as compared to *SIDDLB* is only 2% difference. Besides, the variation of response time in *HNSA* still lower than *SIDDLB* approach.

4.2.2 Effect on The Number Communication Messages

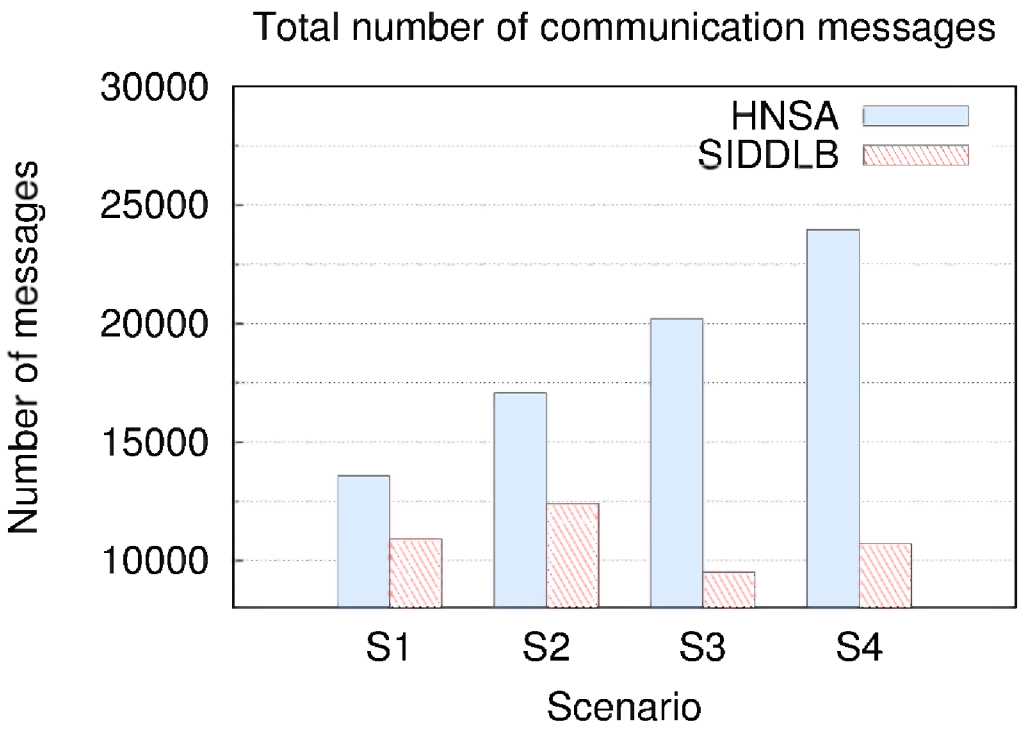

Figure 4.13: The Total Number of Communication Messages

This section disccused about the effect of the number of communication messages. The communication messages included are: i) job transfer message, ii) job acknowledgement message, and iii) job completion message. Both 3 types of communication message include the piggybacked list of neighbours. It is noted that in Fig. 4.13, the *HNSA* has resulted in an increasingly number of communication messages. On the other hand, *SIDDLB* has a lower number of communication messages. The *SIDDLB* is able to result in such a low communication message is partly due to the total number of jobs created at each node. Table 4.2 depicts the total number of jobs created throughout this study include the additional 300 jobs created by *pour jobs*. So, the number of jobs in each scenario should be 10,000 in total. Since *HNSA* allows those nodes that do not have any neighbours in their primary neighbour list to re-distribute jobs, it should always result in a higher total number of communication messages because such nodes are able to formulate the job distribution decision. As for *SIDDLB*, whenever a job has arrived at the highest computing powered node, that node has to accept the job and will not be able to transfer until the completion of the job. Therefore, from S2 to

S3, *SIDDLB* resulted in lower number of communication messages is due to the jobs created at Node ID 9 and 10 which have an empty neighbour list.

Table 4.2: Total Jobs Created at Each Node

Node ID	S1	S2	S3	S4
1	1010	1365	924	1290
2	1001	964	920	879
3	1003	963	927	885
4	1018	982	950	907
5	1013	972	928	884
6	990	951	916	879
7	979	942	907	866
8	1012	975	929	884
9	950	916	1273	1238
10	1024	970	1326	1288

CHAPTER 5

CONCLUSION

In the current distributed parallel computing environment which connects various types of computing resources have enabled us to seamlessly access to an unlimited source of computing power. Often, in a parallel computing paradigm, a task is split up into smaller pieces and concurrently execute them in multiple computing nodes. Due to the heterogeneity of computing nodes, evenly splitting a task could decrease the utilisation of distributed system. The system utilisation decreases when the system load is imbalance. In other words, there are lightly loaded nodes and heavily loaded nodes at the same time. Over prolonged period of time, it could affect the system performances and increase in the response time of a task. This thesis includes the study of the different sizes of workloads and the impact towards the distributed system. With regards to the heterogeneity and different sizes of workloads, dynamic load balancing algorithm is applied to fairly distribute task across computing nodes so that each node would receive a fair amount of workload. Dynamic load balancing algorithm can be implemented in either centralised or decentralised approach. In this research, a decentralised approach has been used so that it can eliminate the bottleneck issue that raised from the centralised approach. In a decentralised dynamic load balancing algorithm, load balancers use the current system load state to formulate job distribution decision. Since there is no centralised load balancer to keep track of other load balancers in the system, each load balancer has to keep track of their neighbours. This research also includes the study of the construction of the neighbour list and the method of accessing the load information of the neighbours.

The first study in this research is regarding the significant impact of different sizes of workloads towards distributed system. A structured tree-based distributed system was used to study the impact of different sizes of workloads. Because of a small changes in a response time of a computing node at the bottom of the tree can easily

impact the overall response time of a given task. A simulation study has been conducted by calculating the value $pi(\pi)$. In order to simulate different sizes of workloads, a set of scenarios has been set-up. Various scenarios ranging from an evenly partitioned load to different sizes of load has been conducted. This study concluded the importance of load balancing whereby the scenario with evenly partitioned load has the lowest overall response time.

The second study presented in this thesis is to analyse the proposed dynamic load balancing algorithm, *HNSA*. The goal of a load balancing algorithm is to minimise the overall response time of a task and at the same time, maximise the utilisation of distributed system. A decentralised dynamic load balancing algorithm has been chosen because of various reasons: i) single point of failure experienced by the centralised approach, and ii) the heterogeneity of computing nodes which leads to the usage of dynamic load balancing. A dynamic load balancing algorithm can be categorised into 4 different policies: *information policy*, *transfer policy*, *selection policy*, and *location policy*. In this study, the focus is on improving the *information policy* and *transfer policy*. The proposed *HNSA* algorithm which is improvised from the *SIDDLB* has been simulated and compared with *SIDDLB*. Response time and the standard deviation of response time has been measured. As a result, the proposed *HNSA* approach is able to minimise the overall response time as compared to *SIDDLB* with an increasing number of communication messages generated. Besides that, the standard deviation value produced from *HNSA* is lower as compared to *SIDDLB*. This means, the proposed algorithm *HNSA* has a higher system utilisation as compared to *SIDDLB*.

This research shows that the proposed algorithm *HNSA*, able to minimise the response time and also maximise the system utilisation as compared to *SIDDLB*. *HNSA* focuses mainly on the construction of neighbour list and also the approach of obtaining the load information of other load balancers. Future work may include the integration of an adaptive *transfer policy* so that a dynamic load balancing algorithm can further optimise the system utilisation.

5.1 Contributions

This thesis contributes to the area of dynamic load balancing policies. Specifically, the information policies:

1. Constructed a neighbour list for a node to consists of nodes that are realiable; that is, nodes with relatively higher computing power.

2. Designed a method of updating load information that combines the mutual information feedback technique and load prediction method, that is based on statistical response time.

REFERENCES

[1] Acker, D. S., & Kulkarni, S. (2007, May). A dynamic load dispersion algorithm for load-balancing in a heterogeneous grid system. In *Sarnoff symposium, 2007 IEEE* (pp. 1–5).

[2] Ahn, H. C., Youn, H. Y., Jeon, K. Y., & Lee, K. S. (2007). Dynamic load balancing for large-scale distributed system with intelligent fuzzy controller. In *IEEE international conference on information reuse and integration, 2007. iri 2007* (pp. 576–581).

[3] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002, November). SETI@home: an experiment in public-resource computing. *Commun. ACM*, *45*(11), 56–61.

[4] Antonis, K., Garofalakis, J., Mourtos, I., & Spirakis, P. (2004, January). A hierarchical adaptive distributed algorithm for load balancing. *Journal of Parallel and Distributed Computing*, *64*(1), 151–162.

[5] Barazandeh, I., & Mortazavi, S. S. (2009, December). Two hierarchical dynamic load balancing algorithms in distributed systems. In *Second international conference on computer and electrical engineering, 2009. ICCEE '09* (Vol. 1, pp. 516–521).

[6] Barazandeh, I., Mortazavi, S. S., & Rahmani, A.-M. (2009, November). Two new biasing load balancing algorithms in distributed systems. In *First asian himalayas international conference on internet, 2009. ah-ici 2009* (pp. 1–5).

[7] Baumgart, I., Heep, B., & Krause, S. (2007, May). OverSim: A flexible overlay network simulation framework. In *IEEE global internet symposium, 2007* (pp. 79–84).

[8] Beltran, M., Guzman, A., & Bosque, J. L. (2008, July). A new cpu availability prediction model for time-shared systems. *IEEE Transactions on Computers*, *57*(7), 865–875.

[9] Casavant, T. L., & Kuhl, J. G. (1988, February). A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, *14*(2), 141–154.

[10] Dong, B., Li, X., Wu, Q., Xiao, L., & Ruan, L. (2012). A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers. *Journal of Parallel and Distributed Computing*, *72*(10), 1254 - 1268.

[11] Foster, I., Kesselman, C., & Tuecke, S. (2001, August). The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, *15*(3), 200–222.

[12] Grande, R. E. D., & Boukerche, A. (2011). Dynamic balancing of communication and computation load for hla-based simulations on large-scale distributed systems. *Journal of Parallel and Distributed Computing*, *71*(1), 40 - 52.

[13] Groot, S., Goda, K., & Kitsuregawa, M. (2010). A study on workload imbalance issues in data intensive distributed computing. In S. Kikuchi, S. Sachdeva, & S. Bhalla (Eds.), *Databases in networked information systems* (Vol. 5999, p. 27-32). Springer Berlin Heidelberg.

[14] Grosu, D., Chronopoulos, A. T., & Leung, M. Y. (2008, November). Cooperative load balancing in distributed systems. *Concurr. Comput. : Pract. Exper.*, *20*(16), 1953–1976.

[15] Gupta, R., & Gopinath, P. (1990, April). A hierarchical approach to load balancing in distributed systems. In *Proceedings of the fifth distributed memory computing conference* (Vol. 2, pp. 1000–1005).

[16] Iosup, A., Dumitrescu, C., Epema, D., Li, H., & Wolters, L. (2006). How are real grids used? the analysis of four grid traces and its implications. In *Grid computing, 7th ieee/acm international conference on* (p. 262-269).

[17] Kalantari, M., & Akbari, M. (2008). Fault-aware grid scheduling using performance prediction by workload modeling. *The Journal of Supercomputing*, *46*(1), 15-39.

[18] Lin, W., & Shen, W. (2010, June). Tree-based task scheduling model and dynamic load-balancing algorithm for p2p computing. In *IEEE 10th international conference on computer and information technology (CIT)* (pp. 2903–2907).

[19] Lu, K., Subrata, R., & Zomaya, A. Y. (2006, May). Towards decentralized load balancing in a computational grid environment. In *Advances in grid and pervasive computing* (Vol. 3947, pp. 466–477). Springer Berlin Heidelberg.

[20] Lu, K., Subrata, R., & Zomaya, A. Y. (2007, December). On the performance-driven load distribution for heterogeneous computational grids. *J. Comput. Syst. Sci.*, *73*(8), 1191–1206.

[21] Lu, K., & Zomaya, A. Y. (2007, July). A hybrid policy for job scheduling and load balancing in heterogeneous computational grids. In *Sixth international symposium on parallel and distributed computing, 2007. ispdc '07.* (pp. 19–27).

[22] Mamat, A., Lu, Y., Deogun, J., & Goddard, S. (2012). Efficient real-time divisible load scheduling. *Journal of Parallel and Distributed Computing*, *72*(12), 1603 - 1616.

[23] Mukhopadhyay, R., Ghosh, D., & Mukherjee, N. (2010). A study on the application of existing load balancing algorithms for large, dynamic, heterogeneous distributed systems. In *Proceedings of the 9th wseas international conference on software engineering, parallel and distributed systems* (pp. 238–243). Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS).

[24] Nandagopal, M., Gokulnath, K., & Uthariaraj, V. R. (2010, September). Sender initiated decentralized dynamic load balancing for multi cluster computational grid environment. In *Proceedings of the 1st amrita acm-w celebration on women in computing in india* (pp. 63:1–63:4). New York, NY, USA: ACM.

[25] Nandagopal, M., Gokulnath, K., & Uthariaraj, V. R. (2011, March). Load distribution through optimal neighbor selection in decentralized grid environment. *European Journal of Scientific Research*, *50*(4), 575–585.

[26] Ni, L. M., Xu, C.-W., & Gendreau, T. B. (1985, October). A distributed drafting algorithm for load balancing. *IEEE Transactions on Software Engineering*, *SE-11*(10), 1153–1161.

[27] Penmatsa, S., & Chronopoulos, A. T. (2011). Game-theoretic static load balancing for distributed systems. *Journal of Parallel and Distributed Computing*, *71*(4), 537 - 555.

[28] Plentz, P., Montez, C., & de Oliveira, R. (2008). Deadline missing predictor based on aperiodic server queue length for distributed systems. *Computer Communications*, *31*(17), 4167 - 4175.

[29] Plentz, P., Montez, C., & de Oliveira, R. (2011). AS prediction mechanism for distributed threads systems. *Journal of Parallel and Distributed Computing*, *71*(10), 1367 - 1376.

[30] Ranjan, S., & Knightly, E. (2008, Sept). High-performance resource allocation and request redirection algorithms for web clusters. *Parallel and Distributed Systems, IEEE Transactions on*, *19*(9), 1186-1200.

[31] Rao, I., & Huh, E.-N. (2008). A probabilistic and adaptive scheduling algorithm using system-generated predictions for inter-grid resource sharing. *The Journal of Supercomputing*, *45*(2), 185-204.

[32] Riakiotakis, I., Ciorba, F. M., Andronikos, T., & Papakonstantinou, G. (2011). Distributed dynamic load balancing for pipelined computations on heterogeneous systems. *Parallel Computing*, *37*(10â11), 713 - 729.

[33] Santana-Santana, J., Castro-Garcia, M. A., Aguilar-Cornejo, M., & Roman-Alonso, G. (2010, February). Load balancing algorithms with partial information management for the DLML library. In *2010 18th euromicro international conference on parallel, distributed and network-based processing (PDP)* (pp. 64–68).

[34] Stankovic, J. A., & Sidhu, I. S. (1984). An adaptive bidding algorithm for processes, clusters and distributed groups. In *ICDCS* (pp. 49–59).

[35] Stavrinides, G. L., & Karatza, H. D. (2009, August). Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations. *Simulation*, *85*(8), 525–536.

[36] Stavrinides, G. L., & Karatza, H. D. (2010). Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *Journal of Systems and Software*, *83*(6), 1004 - 1014. (Software Architecture and Mobility)

[37] Subrata, R., Zomaya, A., & Landfeldt, B. (2008, Jan). Game-theoretic approach for load balancing in computational grids. *Parallel and Distributed Systems, IEEE Transactions on*, *19*(1), 66-76.

[38] Varga, A., & Hornig, R. (2008). An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st international conference on simulation tools and techniques for communications, networks and systems & workshops* (pp. 60:1–60:10). ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[39] Waraich, S. S. (2008, April). Classification of dynamic load balancing strategies in a network of workstations. In *Fifth international conference on information technology: New generations, 2008. itng 2008* (pp. 1263–1265).

[40] Wu, Y., Hwang, K., Yuan, Y., & Zheng, W. (2010, July). Adaptive workload prediction of grid performance in confidence windows. *IEEE Transactions on Parallel and Distributed Systems*, *21*(7), 925–938.

[41] Yang, L., Foster, I., & Schopf, J. M. (2003, April). Homeostatic and tendency-based cpu load predictions. In *Proceedings of international parallel and distributed processing symposium, 2003* (p. 9 pp.-).

[42] Zhou, S. (1988, September). A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, *14*(9), 1327–1341.

PUBLICATION LIST

[1] Lim, J. W. Y., Poo, K. H., & Yeoh, E.-T. (2012a, February). Heuristic neighbor selection algorithm for decentralized load balancing in clustered heterogeneous computational environment. In *Advanced Communication Technology (ICACT), 2012 14th International Conference* (pp. 1215–1219).

[2] Lim, J. W. Y., Poo, K. H., & Yeoh, E.-T. (2012b, November). Neighbor's Load Prediction for Dynamic Load Balancing in a Distributed Computational Environment. In *TENCON 2012 - 2012 IEEE Region 10 Conference* (pp. 1–6).

[3] Lim, J. W. Y., Poo, K. H., Yeoh, E.-T., & Tan, I. K. T. (2011, November). Performance analysis of parallel computing in a distributed overlay network. In *TENCON 2011 - 2011 IEEE Region 10 Conference* (pp. 1404–1408).