

Heuristic Neighbor Selection Algorithm for Decentralized Load Balancing in Clustered Heterogeneous Computational Environment

Jay W.Y. Lim, Poo Kuan Hoong, Eng-Thiam Yeoh

System Innovation Group, Faculty of Information Technology, Multimedia University, Cyberjaya, Malaysia

{jay.lim, khpoo, etyeoh}@mmu.edu.my

Abstract—Clustered heterogeneous computing environment is used to execute parallel applications that require significant amount of computing resources either in the form of computational processing resources or data storage. A cluster, comprising of heterogeneous nodes, requires careful load balancing strategies in order to result in a good processing response time for parallel applications. The workload for each system in the cluster cannot be equally distributed but has to be taken as an important parameter for the load balancing strategy. A centralized load balancer would require a global load balancer that will be overwhelmed with communication messages in a large clustered environment. This overhead can be addressed by a decentralized approach to load balancing where decisions are formulated by the nodes performing scheduling algorithm vis-a-vis communicating with each member node of the cluster. Although this will lead to an overall increase of communications, it will eliminate the bottleneck of having a centralized load balancing node. In an overloaded node (sender) initiated decentralized load balancing strategy, each sender will need to have a predetermined list of neighboring nodes to offload their jobs. In this paper, we propose a heuristic based neighbor selection algorithm that selects a neighbor for job distribution whenever overload occurs. Simulations were conducted using the OMNeT++ simulator and the overall proposed performance was compared to other existing neighbor selection method. From our simulation results, we found that our proposed method demonstrates overall improvement for load balancing by reducing the average response time.

Index Terms—Decentralized distributed system, neighbor selection, dynamic load balancing, task re-distribution

I. INTRODUCTION

Distributed system is a computing model which comprises of a collection of computing nodes contributing their processing resources to achieve a common goal. It can be used for sharing files across the network or processing information. On the other hand, grid computing is a class for distributed systems which combines computing resources from multiple administrative domains to form a big supercomputer. In the matter of this, it benefits most to those computational intensive applications that generally demands a lot of computing time and memory storage. As an example, a scientific application, ERIS, the realistic computer simulation of the Milky Way took 9 months of computing time by using a collection of supercomputers in order to see the spiral of our galaxy [1]. In distributed environment, however, there is a higher probability a computing node could be in the state of idle, lightly loaded or even heavily loaded with jobs. Jobs that appear to be in

the heavily loaded computing nodes turn out to take a longer time to be executed since a job has to wait for the previous job to complete. In other words, the waiting time for a job to be executed increases as the job queue grows.

To distribute jobs equally, load balancing method is employed by distributing jobs from heavily loaded nodes to the lightly loaded nodes. It has been extensively studied where load balancing is implemented by using a number of different approaches. Thomas and Jon [2] studied on the taxonomy of load balancing and further grouped it into several subclasses, namely static and dynamic load balancing. Static load balancing is where all necessary information to formulate the job distribution decision were assumed to be known at the compilation time of a job. Dynamic load balancing, in contrast, attempts to make decisions during the run time of a job. Hence, each node has to frequently exchange load information with each other in order to obtain the most up to date load information to formulate the decision. Load information exchange can be done via periodically or on demand of a node and it has to be taken carefully in order to prevent any further overwhelming of communication messages in the system. Besides that, load balancing algorithm can be implemented in a centralized or decentralized approach. A centralized approach is where the responsibility for the job scheduling resides in a single node. Although this method is easier to be implemented, this approach is not scalable due to the communication bottlenecks at the centralized node as the size of the system increases. Decentralized approach, on the other hand, has the limitation of overall high communication overheads. However decentralized approach eliminates the bottleneck of having a centralized node.

In this paper, we optimized the selection of neighbors on each node individually instead of having the knowledge of all nodes in the system. In this case, the *load balancer* (LB) does not have to exchange load information with all other nodes. In addition, communication messages can be further reduced by triggering the load information exchange when there is only an imbalance of workload within the neighbors. The main contribution in this paper are as follows:

- Enhanced neighbor selection for job distribution.
- Optimizes load balancing by minimizing the standard deviation of average response time.

In the next section, we summarized the work has been done by previous researchers. Then in section III, we describe the model, simulation setup and implementation, followed by the results and discussions in section IV. Finally, we conclude our findings in section V.

II. RELATED WORK

Stankovic et. al. [3] proposed a bidding algorithm, sender initiated, which migrates job from heavily loaded nodes to the lightly loaded nodes in order to sustain load balancing. With reference to this algorithm, the heavily loaded node, the sender, initiates the bidding request of the job to all other nodes, the receivers, in the system. After all the receivers placed their bids, the sender chooses a receiver with the best bid and transfer the job to it. Over a prolonged period of time, this approach experience communication overheads and over migration of jobs. In order to overcome these issues, drafting algorithm, a receiver initiated algorithm, proposed by Ni et. al. [4] initiated by the lightly loaded nodes which send out a draft of the job that it can execute to all other nodes in the system. Only heavily loaded nodes with additional jobs that matches the draft request reply to the lightly loaded nodes. Upon the lightly loaded nodes receive all replies, each of them selects the best draft then request to transfer the job from the node. This approach tends to lessen the burden of the heavily loaded nodes by not performing the load balancing algorithm. However, it is possible that a lightly loaded node selects several drafts from heavily loaded nodes that eventually will repeat the drafting algorithm again. In order to overcome this, the lightly loaded node must be able to reject the draft. Apart from that, Acker et. al. [5] proposed job distribution algorithm to balance the load on a decentralized distributed systems. The proposed dynamic algorithm can accommodate nodes joining and leaving the system. Each node stores information with respect to the computing resources of other nodes and exchanged this information periodically. This approach is scalable at the cost of increasing communication overheads and memory storage. In order to address these issues, Malarvizhi et. al. [6], [7] proposed a neighbor selection algorithm to store only feasible nodes as neighbors instead of storing all nodes in the systems. Respecting to this, the highest processing capability nodes cannot redistribute their jobs to other nodes if they were overloaded. This is due to the neighbor selection algorithm which assumes that the highest processing capability nodes will never be overloaded. Besides that, each cluster is responsible for their own load balancing and redistribution of jobs to a higher processing capability neighbor upon overloaded.

III. SYSTEM MODEL

In this paper, the distributed system model proposed is based on distributed clusters and the communications delay are assumed to be minimal or negligible. Fig 1 illustrates the mentioned model of each cluster where the resources refers to computing nodes and user task is the job submitted by user at any cluster. Each cluster comprises of interconnected

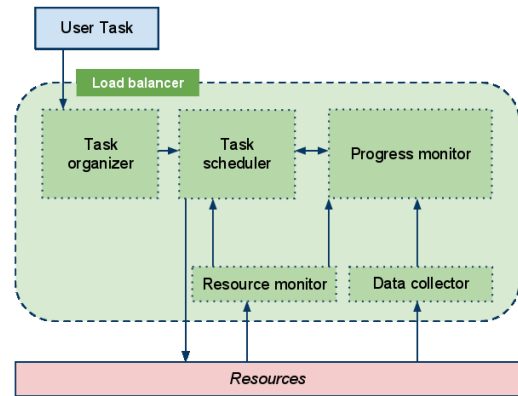


Fig. 1. Distributed cluster model

computing nodes and a LB. The LB is responsible for managing its own resources, scheduling users jobs and balancing workload. As for the job scheduling, LB examines every arrival jobs from users and decides whether to offload the job or not, based on both the load information of its neighbors and itself. Hence, this load balancing algorithm is initiated by the heavily loaded cluster. As regards to load information exchange, it is based on [5], [6], [7]. Initially, a LB does not have any load information respecting its neighbors, therefore, this information was assumed to be the same at the beginning. Over a prolonged period of time, LB learns about its neighbors by exchanging load information. The trigger of exchanging load information is when a heavily loaded cluster, the sender, needs to transfer jobs to a remote cluster which the load is light or upon job completion. The following are the notations used to determine our algorithm.

- J , the set of all jobs.
- C , the set of all clusters.
- $C_{neighbor}$, the set of neighbor.
- z , the cluster that is executing the algorithm.
- β , the load tolerance level.
- $\forall c \in C$ comprises of the following properties:
 - pwr , the processing power.
 - $delay$, the communication delay.
 - $load$, the current load index.
 - $time$, the last updated time for load index.

TABLE I
LB'S PROCESSING POWER

LB's ID	5-6	7-9	10	11-12	13-14
Processing Power (Job-size/sec)	10	20	30	40	50

A. Load information sharing

Load information sharing is an essential process in dynamic load balancing where a LB determines the job distribution

based on such information. The load information keeps on exchanging as long as there is a job distribution process is going on or upon a job completion reply. In exchanging load information, the sender sends the job and piggyback the load information of itself and α percentage of its neighbors to the receiver, which is also employed by [5], [6], [7]. The load information sent is mainly to update the receiver regarding load of other clusters in the system. However, with reference to the piggyback load information, it is possible that there were no neighbors to be included if the α value is small enough or the sender is the highest processing node which contains no neighbors at all. In order to address this, a sender, as long as is not the highest processing node, must include at least one neighbor to send to. Upon the receiver receives the job and the neighbors load information, the receiver updates its neighbor list based on the *time*. As regards to this, the receiver compares the *time* marked in the load information from the sender and its own, hence, the update can be only successful if the *time* value is greater than the one it has.

B. Neighbor selection for job distribution

Algorithm 1 Heuristic Neighbor Selection

Input: arrival job, $j \in J$

Output: neighbor cluster, $c \in C$

```

1: if the arrival of  $j$  is from  $z$  then
2:    $\exists \min \in C_{neighbor}$ , where  $\min$  possesses the least load.
3:   if  $z.load() - \min.load() < \beta$  then
4:     return  $z$ 
5:   else
6:     return  $\min$ 
7:   end if
8: else if the arrival of  $j$  is from  $c \in C$  then
9:    $Nbor \leftarrow$  set of neighbors piggybacked from  $c$ 
10:  if  $Nbor \neq 0$  then
11:     $\exists \min \in Nbor$ , where  $\min$  possesses the least load.
12:    if  $z.load() - \min.load() > \beta$  then
13:      return  $\min$ 
14:    else if  $z.load() - c.load() > \beta$  then
15:      return  $c$ 
16:    end if
17:  end if
18:  return  $z$ 
19: end if

```

Every individual LB possesses its own neighbor list for job distribution. Such neighbor list is formulated based on Malarvizhi [6], [7] model, where each LB selects those clusters that possess greater processing power and with the communication delay not more than 1.5 times of the lowest one. Based on simulation results in [6], [7], it was concluded that the ratio of 1.5 produces a good result. Upon imbalances of workload occur, this will trigger the LB to select its neighbor which possesses the minimum load to redistribute the job. However, the highest processing node will face problem on redistributing jobs due to an empty neighbor list. In this

abstract, the highest processing node always enqueue arrival jobs despite if it is overloaded. Eventually the job queue built up and forces each of the job in the queue to wait for the previous job to be completed. In order to address this issue, LB with empty neighbor list should make use of the arrival load information sent by the sender to reselect a neighbor for job processing. Algorithm 1 outlined the method we used to select a neighbor upon job arrival, namely Heuristic Neighbor Selection Algorithm (HNSA).

C. Implementation and simulation setup

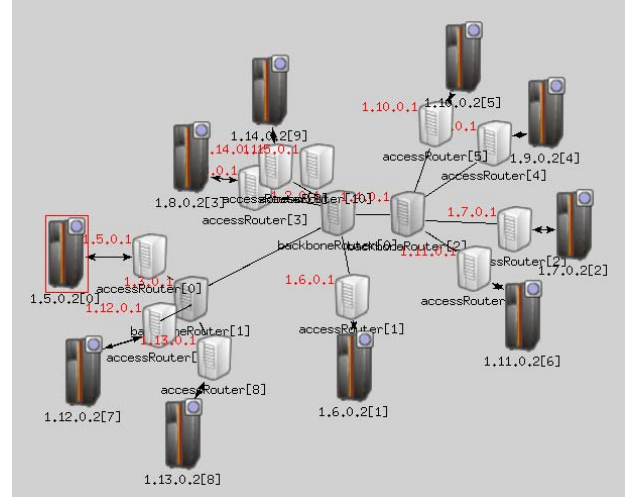


Fig. 2. Overlay distributed LBs in OMNET++

In order to demonstrate the issue of workload imbalance in distributed clustered system, we conducted the simulations under various workloads using discrete event network simulator, OMNET++ [8]. This simulation was intended to simulate the inter-cluster load balancing. Hence, only the inter cluster LBs were simulated instead of simulating the computing nodes in a cluster. The simulation model mainly comprises of 10 overlay distributed LBs with a network diameter of 6. Table I depicts the specifications of LB where the smaller the LB's ID, the smaller the processing power (the processing power is measured in terms of job size per second) and Fig 2 is the screen capture from the network simulator. In our simulation, the network topology consists of 3 backbone routers of the network which were connected using 10Gbps fiber line with 1ms delay and the rest of the connections were using 1Mbps digital subscriber line (DSL) with 20ms delay.

The jobs generated by a *job generator* into each cluster is at every 2 seconds. A *job generator* reads the job from a file which contains 10,000 jobs where each job is described by its size ranging from 20 to 99. Apart from generating jobs one by one, we also introduced *pour jobs*, which is an amount of jobs pouring into a cluster at a single rate of time in order to simulate the heavy workload environment. The simulation ends after all jobs are completed.

There are 4 different scenarios to simulate the heavy workload in different kinds of LB shown in Table II. Scenario 1

TABLE II
LB'S ID ON WHICH *pour jobs* OCCUR AND THE AMOUNT OF JOBS
POURED PER LB

Scenario	LB's ID	Number of Jobs
1	-	-
2	5	300
3	13,14	300
4	5,13,14	300

TABLE III
SIMULATION PARAMETERS

Parameters	Values
Total number of jobs	10,000
Job size	20-99
Job arrival rate (sec)	2
Total number of LBs	10
β , load tolerance level	0.9
α , percentage of neighbors	20%

is where all LB receives jobs at a constant rate of time, no *pour jobs* occur in this scenario. In scenario 2, is where the *pour jobs* occur in a single low processing capability LB at an amount of 300 jobs. As in scenario 3, the *pour jobs* occur in the 2 highest processing capability LBs with the same amount of jobs as the previous scenario. And in scenario 4, is the combination of both scenario 2 and 3.

For performance evaluation, we measured and recorded the response time for every single job. The timer for the response time starts immediately after a job generated at a cluster and stops after the job completion at the same cluster. The timer for waiting time starts together with the response time but stops after a job being executed.

IV. RESULTS AND DISCUSSIONS

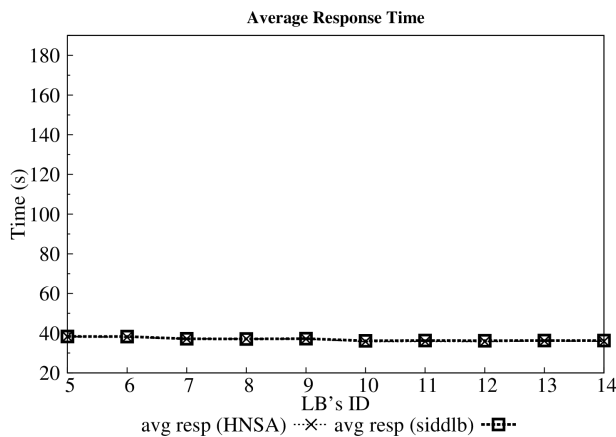


Fig. 3. Average response time for scenario 1

In this paper, the performance of this proposed method, *HNSA*, is evaluated using the simulation setup described and compared with the method proposed by Malarvizhi [6] namely *siddlb* with the simulation parameters depicted in Table III. The processing power of each cluster is shown in Table I.

Fig 3, 4, 5 and 6 show the results of an average response time of a job from the 4 scenarios that were described in Table II. In measuring load balancing, the difference of each job response time has to be small. The bigger the difference, the more variate the response time of a job. With regard to this, standard deviation is used. The overall average of response time of 10,000 jobs for the *HNSA* method and *siddlb* method in scenario 1 are 36.70s and 36.94s respectively. Moreover,

the standard deviation for both methods are relatively small, generally less than 1s and with our *HNSA* method produced approximately 12% higher than *siddlb* method, however, this difference is trivial. From our simulation result, we can conclude that in the normal situation the *HNSA* method works just as fine as the *siddlb* method.

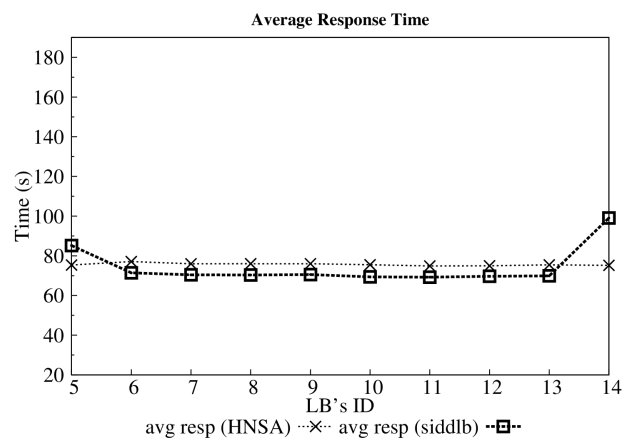


Fig. 4. Average response time for scenario 2

In scenario 2, *pour jobs* at an amount of 300 was introduced at a single rate of time in the weakest cluster which is LB 5. Upon such event occur in using *siddlb* method, LB 5 selects the neighbor which has the minimal load to transfer the job, LB 10 for instant. Upon LB 10 receives the job, it will either accept the job if it is not heavily loaded, or otherwise it will repeat the same thing as what LB 5 did. Eventually, the job reaches either LB 13 or 14 which possess the highest processing capability in the distributed system. These 2 LBs cannot further transfer the job to other neighbors due to an empty neighbor list, hence, they would have to enqueue the job, which eventually increase the size of the queue. With reference to this, LB 14 is the one which experiences an increase of average response time as shown in Fig 4. The *HNSA* method, on the other hand, work slightly different on handling arrival job. While job arrives on LB 14, at the same time it also receives the load information of the sender and its neighbors. In the case of *siddlb*, such information will be discarded. Our approach, on the other hand, is able to use this information to search for suitable neighbor which can process the job. Therefore, in scenario 2, the overall average response

time for the *HNSA* method and *siddlb* method are 147.04s and 153.38s accordingly. And, the *HNSA* method has a standard deviation of 4.92s which is approximately 67% less than *siddlb* method.

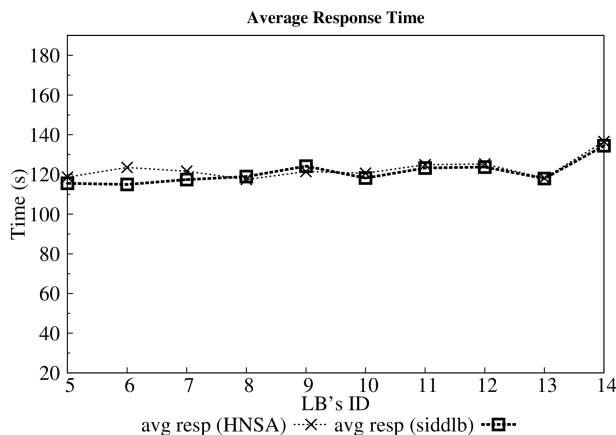


Fig. 5. Average response time for scenario 3

Scenario 3, where *pour jobs* event occurred in the 2 highest processing capability clusters which are LB 13 and 14. It is noticeable in Fig 5 that these 2 approaches have approximately the same average response time. The overall average response time of each job for the *HNSA* method and *siddlb* method are 122.82s and 120.84s correspondingly. Besides that, the standard deviation for these 2 methods were slightly different. Precisely, 5.53s for the *HNSA* method and 5.83s for *siddlb* method. The reason for this is mainly because of the 2 highest processing capability clusters. As explained in scenario 2, where the decision for an empty neighbor list LB to redistribute job is to make use of the sender's load information and its neighbors load information, otherwise, job have to be enqueued. In this scenario however, where the *pour jobs* occurred in the highest processing capability clusters, which do not have any neighbors and any knowledge about the existence of other clusters. Therefore, jobs have to be processed locally, which it is performed similar to *siddlb*.

Lastly in scenario 4, where the *pour jobs* occurred in LB 5, 13 and 14. This is primarily to prove that LB 13 and 14 will not be accepting any further jobs from the lower processing capability clusters. The *siddlb* approach shown in Fig 6 shows that average response time of a job in LB 5 increases greatly. This is due to jobs being processed in LB 13 and 14 which are already overloaded. On the other hand, our *HNSA* method, it is able to redistribute jobs to other clusters. The overall average response time for our *HNSA* method and *siddlb* method are 75.64s and 74.51s respectively. Due to the reason mentioned, the standard deviation for *siddlb* turns out to be approximately 7% greater than our proposed method which is 0.66s.

V. CONCLUSION

In this paper, we outlined an algorithm for job distribution by selecting a neighbor upon a job arrival into a cluster.

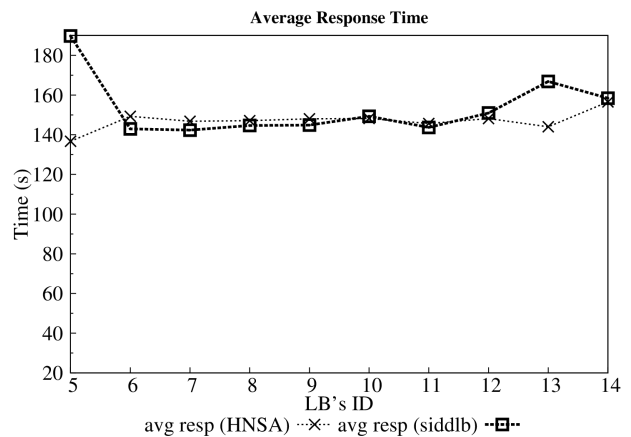


Fig. 6. Average response time for scenario 4

We constructed simulation for 10 overlay distributed clusters with different processing capability. We simulated 10,000 jobs under 4 different scenarios to simulate the overloading in clusters. The results show that our proposed method able to enhance the neighbor selection for job processing upon load imbalances occurred and minimizes the standard deviation of response time as a measurement of load balancing. However, this algorithm still experiencing some weaknesses where in some situation where jobs cannot be transferred due to no prior knowledge of neighbors. For our future works, we plan to work on these issues as well as variate the jobs arrival time.

REFERENCES

- [1] J. Guedes, S. Callegari, P. Madau and L. Mayer, (2011), "Forming Realistic Late-Type Spirals in a LCDM Universe: The Eris Simulation", In *Cornell University Library*, Retrieved October 5, 2011, from <http://arxiv.org/abs/1103.6030>
- [2] T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing System", In *IEEE Transaction on Software Engineering*, Vol. 14, No. 2, pp. 141-154, February, 1988.
- [3] J.A. Stankovic and I.S. Sidhu, "An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups", In *Fourth International Conference on Distributed Computing Systems*, San Francisco, CA, pp. 49-59, May 1984.
- [4] L.M. Ni, Chong-Wei. Xu, and T.B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing", *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 10, pp. 1153-1161, October, 1985.
- [5] D.S. Acker, and S. Kulkarni, "A Dynamic Load Dispersion Algorithm for Load Balancing in a Heterogeneous Grid System", In *IEEE Sarnoff Symposium*, Princeton, NJ, pp. 1-5, April 30 2007-May 2 2007.
- [6] N. Malarvizhi, K. Gokulnath, and V. Rhymend Uthariaraj, "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 (A2CWIC '10)*, Article 63, pp. 1-4, September 16-17 2010.
- [7] N. Malarvizhi, K. Gokulnath, and V. Rhymend Uthariaraj, "Load Distribution through Optimal Neighbor Selection in Decentralized Grid Environment", In *European Journal of Scientific Research*, Vol. 50, No. 4, pp. 575-585, March 2011.
- [8] OMNeT++ Community Site (2011), "OMNeT++ Discrete Event Simulation System [Online]", Available <http://www.omnetpp.org/>
- [9] I. Baumgart, B. Heep, and S. Krause, "OverSim: A Flexible Overlay Network Simulation Framework", In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007*, Anchorage, AK, USA, pp. 79-84, May 2007.