Dynamic Load-Balancing Based on a Coordinator and Backup Automatic Election in Distributed Systems

Tarek Helmy*
College of Computer Science and Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31261, Mail Box 413, Kingdom of Saudi Arabia
Tel: 966-3-860-1967 & Fax: 966-3-860-2174
**** On leave from College of Engineering, Department of Computers Engineering and Automatic Control, Tanta University, Egypt.***
helmy@kfupm.edu.sa

Fahd S. Al-Otaibi
Department of Computer, College of Education
King Abdulaziz University
Jeddah, Kingdom of Saudi Arabia
fsalotaibi@kau.edu.sa

Abstract: *In a distributed system environment it is likely that some nodes are heavily loaded while others are lightly loaded or even idle. It is desirable that the work-load is fully distributed among all nodes so as to utilize the processing time and optimize the whole performance. A load-balancing mechanism decides where to migrate a process and when. This paper introduces the load-balancing mechanism as a new scheme to support the reliability and to increase the overall throughput for distributed systems environment. The idea is to assign one node as a coordinator in addition to a backup node, with the possibility of automatic election in case both coordinator and backup fail. The presented scheme has been integrated into a Zap system. Zap provides a transparent checkpoint-restart mechanism for migrating a PrOcess Domain (POD). A POD provides a group of processes with a private namespace that presents the process group with the same virtualized view of the system. Experimental results show that the load among all nodes is balanced and the freezing time is low compared with other load-balancing mechanisms such as random selection of the destination, unless the number of communication messages needed for migrating a POD becomes high.*

Keywords: *Load-Balancing, Distributed Systems, Zap, POD, CBAE.*

Received: June, 2010 | ***Revised:*** December 10, 2010 | ***Accepted:*** February 10, 2011

1. Introduction

IN the literature, there are many load-balancing algorithms, especially for process migration [1], [4], [6], [10], [14], [15], [21]. Each mechanism has pros and cons. Generally, the load-balancing mechanisms are divided into two main categories: central-based algorithms and decentralized algorithms. In the centralized approach [7], one coordinator node regularly pools other nodes to obtain their current load. Whenever the coordinator notices that a node is lightly loaded, it selects this node to receive processes from an overloaded node. The coordinator migrates a process from an overloaded node to the lightly loaded node. In the decentralized approach [17], [11], each node has its own load balancer, and if the load exceeds a pre-defined threshold then the node is overloaded. There are three main categories of this kind of load balancing; *Sender-Initiated*, *Receiver-Initiated* and *Symmetric* [2].

One of the well-known decentralized mechanisms is a random selection where no exchange of state information is required. It works by selecting a destination node randomly and then migrating a process to that node. If the destination node is currently overloaded, then it selects another destination node randomly, and so on. One drawback of this mechanism is the possibility of migrating the same process to more than one node. Another way to randomize the algorithm is to select a destination node

randomly so that the source node pools the current load of the destination node. Before migrating process, if the destination load is in an overloaded state then the process will continue running locally; otherwise the process will migrate directly.

The rest of the paper is organized as follows. Section 2 presents related work and our motivation. Section 3 describes the proposed algorithm, Coordinator Backup Automatic Election (CBAE). In Section 4, the performance of CBAE compared with random load balancing algorithm in a series of simulations. Finally, Section 5 concludes the paper and presents the future work direction.

2. Related Work

Different types of load-balancing algorithm have been developed and used in distributed environments [1], [4], [6], [10], [14], [15], [18], [19], [20]. In Sprite [3], load-balancing is restricted to two occasions. This may be when a resource intensive program is about to start, or during eviction from a remote host. A combination of centralized and distributed load-balancing mechanisms is used by allowing each node to run a process called the load-average daemon to monitor the load of that node. When the node becomes idle, the load-average daemon notifies the central migration server that the node is ready to accept migrated processes. The new arrival process is migrated to that idle node. The most complicated load balancing mechanism is proposed in MOSIX [3]. It differs from the previous systems since the load balancing has to be done continuously, not just during creation or eviction of a process. Processes get migrated whenever the cluster gets unbalanced. If a process requirement exceeds a certain threshold, then the process becomes a candidate for migration. Each node has a load vector which contains information about the load of a random subset of neighboring nodes. This load vector is constantly updated through "load information dissemination". Some researchers combined the mechanism of load balancing with a scheduling algorithm. A Global Scheduler (GS) is implemented by MPVM [5]. This scheduler is a centralized manager. GS decides when and where to migrate a task, in addition to which task is to be migrated. The load-balancing is done when the node is under heavy load or during task creation or eviction as in Sprite [3].

Another way of handling load-balancing called multi-leader is proposed by Wills et al. [16]. This algorithm assigns one of the nodes to be the leader. The leader can be any node in the cluster, and is chosen randomly from among all of the nodes. Since the leader works as a coordinator, it has three basic roles: (1) Accepting and storing information about the load of lightly nodes. (2) Maintaining a list of available light load nodes. (3) Returning a suitable node to any clients that request it. PANT [9] uses this algorithm and employs a fault-tolerant communication architecture based on multicast communication that minimizes the load on busy cluster nodes. Semi-distributed strategy is used by PEACE [12] and provides good results. It works by partitioning the whole cluster structure into several local clusters. Each level has its own mechanism of load balancing. Within each cluster a centralized approach is used, while distributed mechanism is a candidate to be done at cluster level. Each node in any cluster becomes overloaded when it exceeds a specific threshold. This threshold is not fixed but it can be modified depending on the situation of the global and centralized mechanism.

For the grid systems, Kai et al. [8] presented an efficient decentralized load-balancing algorithm for heterogeneous systems. It is different from the traditional distributed environment since it takes care of the site heterogeneity and the communication overhead. This algorithm takes into consideration that each node in the site has its own power and communication delay. It applies the concept of sender initiation for an overloaded node. Zap [13] is a system that provides a transparent checkpoint-restart mechanism for migrating a group of processes. It introduces a thin virtualization layer on top of the operating system that introduces a PrOcess Domain (POD). A POD provides a group of processes with a private namespace that presents the process group with the same virtualized view of the system. The POD itself can be stopped, migrated and then restarted at the destination node transparently. Nevertheless, Zap has a number of limitations and requires future research.

In this paper, we focus on designing a load-balancing mechanism that helps to decide where to migrate a POD. The responsibility of this mechanism is to guarantee the load of all nodes to be balanced and to support the reliability and fault tolerant in case of failure. The proposed load-balancing algorithm is a hybrid of centralized and dynamic load balancing. It works based on centralized coordinator and backup to avoid the possibility of coordinator failure. It supports the decentralization by allowing each node to take its decision on migration. It takes into consideration the possibility of simultaneous failures by allowing the election concept. It supports fair load-balance among the nodes with a low freezing time.

3. Coordinator and Backup with Automatic Election

Coordinator and Backup with Automatic Election (CBAE) is a load-balancing mechanism that works by assigning one node as a coordinator and another node as a backup, with the ability of automatic election when both coordinator and backup fail. The importance of backup is to increase the reliability of the load-balancing algorithm and the overall system.

We apply CBAE to be as a load-balancing algorithm that can be embedded into Zap [13].

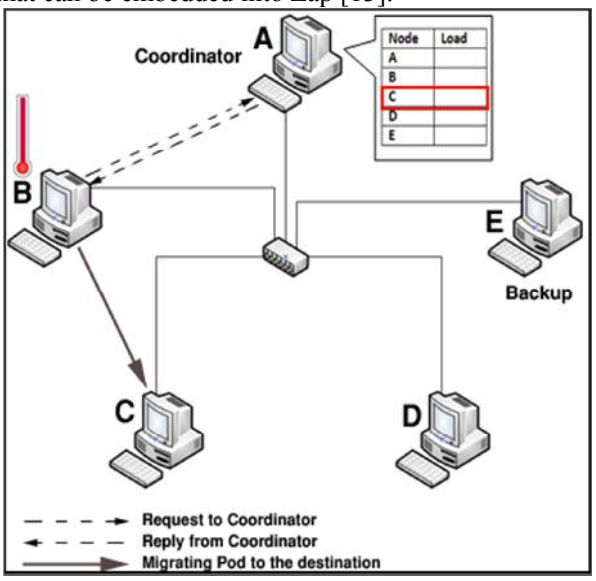

Figure 1. No Failure.

3.1 CBAE Basic Setup

The coordinator and the backup nodes are the main players in this scheme. Coordinator and backup assignments are done randomly. The main responsibility of a coordinator is to keep track of the current load state of each lightly loaded node in the community. Each lightly loaded node needs to send its current load state periodically to the coordinator. The coordinator itself maintains a table to hold incoming valuable information from all nodes and then sends any modification to the backup node. Every overloaded node needs to ask the coordinator for an appropriate node to migrate a process directly to that destination node. We introduce a backup node that works in case of coordinator failure. Moreover, we enhance this mechanism by allowing an automatic election of new coordinator and backup nodes in case of simultaneous failures. Each node has to know exactly the coordinator and the backup to communicate with. Each node has its own defined threshold assigned by the system administrator. A load balancer is presented in each node to decide whether the node is overloaded when it exceeds its threshold. This load balancer guarantees that the migration decision is done by the node itself without any intervention of the coordinator. The overloaded node needs only to ask the coordinator for an appropriate lightly loaded node.

Three possible states for this algorithm can be faced. The no-failure state occurs when the coordinator is still running. The second state represents the case of coordinator failure. The worst case is when both coordinator and backup nodes fail. In this paper, we present each case, and then we make an experiment to ensure that CBAE helps to utilize the CPU time in each node. To illustrate these three states, Figure 1 shows 5 nodes from A to E connecting via a local network. We assign node A and E to be the coordinator and backup nodes respectively.

3.1.1 No Failure

Each overloaded node asks the coordinator for an appropriate destination node. As shown in Figure 1, suppose node B is overloaded. It must do the following steps:

1. Node B sends a request to the coordinator, asking for an appropriate destination.
2. Coordinator node then replies B with a lesser load node and the maximum available free size that can be migrated to the less load node, node C in this example.
3. Node B sends a communication message to node C to check its availability.
4. Node C replies Node B by an acknowledgment.
5. Node B selects one POD large enough but not exceeding the maximum available size received by coordinator.
6. Selected POD is migrated directly to node C.

This scenario occurs each time a node is overloaded, and the coordinator is the main significant player in this scheme.

3.1.2 Coordinator Failure State

CBAE algorithm treats the possibility of coordinator failure by keeping track of backup node. In case of coordinator failure, the overall load-balancing mechanism is still running perfectly. Whenever the backup node receives an incoming request for an appropriate destination, it checks that the coordinator has definitely failed. If so, it changes its status to act as the coordinator and chooses one node randomly to act as a backup, and then the whole information about the nods' loads is transferred to the new backup node. The main idea is represented in Figure 2, which shows the case of coordinator failure.

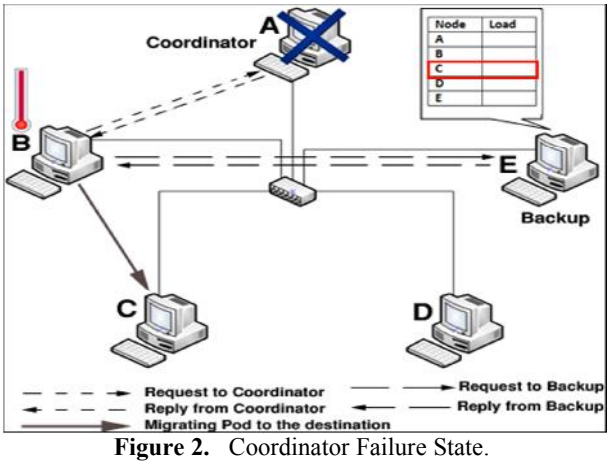

Figure 2. Coordinator Failure State.

An overloaded node follows the same mechanism represented in normal state, but there is a restriction for sending a request to backup node after checking that the coordinator has failed. It is clear from Figure 2 that the same situation is assumed where node B is overloaded, so that the following procedure is done:

1. Node B sends a request to the coordinator for an appropriate destination.

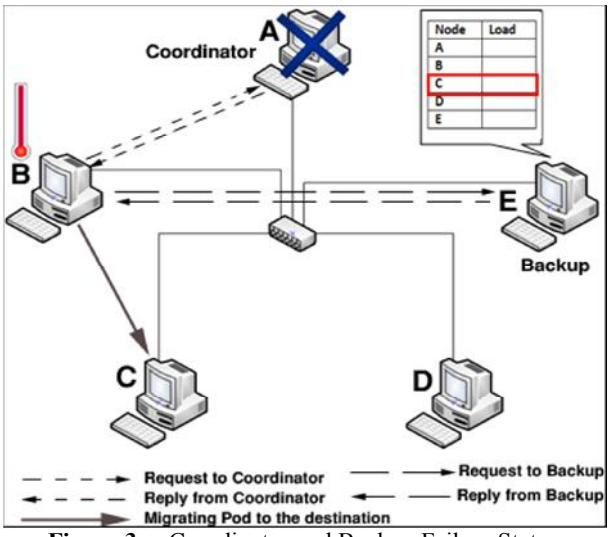

Figure 3. Coordinator and Backup Failure State.

2. If the coordinator doesn't reply in the specific period, B sends another request to the coordinator to checks whether the coordinator is alive.
3. If the second request is not answered in the specific period, B knows the coordinator has failed.
4. Node B sends its request to the backup node for an appropriate destination.
5. Backup node changes its state to be coordinator, and it selects randomly one of the other nodes to be, the new backup node.
6. New coordinator transfers the status table to the new backup node.
7. New coordinator informs all other nodes to update the coordinator and backup nodes information that were kept in each node.
8. The new coordinator then replies B with a suitable destination node which has a lesser load node and the maximum available free size that can be migrated to the destination, node C in this example.
9. Node B sends a communication message to node C to check its availability.
10. Node C replies Node B by an acknowledgment.
11. Node B selects one POD large enough but not exceeding the maximum available size received by coordinator node.
12. Selected POD is migrated directly to the destination, node C in our example.

The main benefit of using a backup node is significantly clear. Without backup, a lot of time and communication messages are needed, for instance to elect one of the nodes to be a new coordinator by negotiation to elect the lightly loaded node to act as a new coordinator. With backup the only communication messages needed are to inform all nodes to change the coordinator and the backup node's addresses, so that the communication messages are almost halved.

3.1.3 Coordinator and Backup Failure State

With CBAE, this issue can be resolved by introducing an election concept. In case of coordinator and backup failure, the overloaded node, that needs to migrate a POD, elects itself to be as a new coordinator. It informs and asks all other nodes to pass their load status to it. The new backup node is chosen randomly by the new coordinator, and then the whole load information table will be transferred to it.

There are no restrictions on choosing a coordinator and backup, and so the most feasible way is to allow the overloaded node to elect itself to be a new coordinator. This helps to save an election time that may spent for negotiation and to ensure the system reliability. Figure 3 shows that both coordinator and backup have failed. The overloaded node B interacts with this situation as follows:

1. Node B sends a request to the coordinator for an appropriate destination.
2. If the coordinator doesn't reply in the specific period, B sends another request to the coordinator to check whether the coordinator is alive.
3. If the second request isn't replied in the specific period, B knows the coordinator is failed.
4. Node B sends its request to the backup node for an appropriate destination.
5. If the backup node doesn't reply in the specific period, B sends another request to the backup node to check whether the backup fails.
6. If the second request isn't answered in the specific period, B knows the backup node has failed.
7. The overloaded node (node B) elects itself to be new coordinator and then sends this election to others to record it.
8. New coordinator selects randomly one of the other nodes to be the new backup.
9. All other nodes change the previous coordinator and backup information that were kept in each node.
10. New coordinator asks all nodes to send their current load status.
11. New coordinator transfers the nodes' load status table to the new backup.
12. In case of recovering an old coordinator or backup nodes, they will be considered as a normal node.

The remaining steps follow the same normal state steps for selecting a suitable node as a destination. This election mechanism provides a valuable effect on nodes community in the context of reliability. It guarantees that the load distributes among all active nodes to maximize the CPU utilization, and as a result the throughput increases.

4. Experimental Results

We have implemented the CBAE load-balancing algorithm using a programming simulation. We conducted our experiments on machines with Intel Core 2 Duo CPU 2.00 GHz, 2 GB RAM. The operating system is Windows Vista Home Premium platform. The simulation is programmed by C#

programming language and MS Access for storing results.

4.1 Main Characteristics

Initially, we simulated the main Zap components and implemented the proposed algorithm on them. Mainly Zap components consist mainly of:

- **Node**
  Physical machine is represented as a logical node that has a pre-defined node's power that can be assigned by the system administrator. Here, for simulation, we present specification for each node that reflects the node's power. High specification means a powerful machine and vice versa. Each node can be a normal node, a coordinator or a backup node.
- **POD**
  Each node has a random number of PODs. A POD itself has a size which is the sum of all process sizes inside it. We assign a field with each POD to count how often a specific POD is migrated. This field helps to avoid choosing a specific POD continuously "to avoid *Thrashing*" which prevents this POD from executing.
- **Process**
  Set of processes is assigned randomly to each POD. Each process has size, time of execution, period of time to be executed.

4.2 Experimental Assumption

To experiment and evaluate the CBAE load-balancing mechanism, we assume that we have 10 nodes, 1 up to 10. Each node has a specification reflecting its power. We assume specification 1 means the lowest power, whereas 7 mean a high power node. Defined threshold is assumed and assigned to each node depending on the following relation. 1 specification = 100000 Threshold "Capacity of POD's size".

Table 1. Nodes Information

Node ID	Specification	Threshold	Coordinator	Backup
1	7	700000	0	1
2	6	600000	0	0
3	5	500000	0	0
4	5	500000	0	0
5	4	400000	0	0
6	4	400000	0	0
7	3	300000	1	0
8	2	200000	0	0
9	2	200000	0	0
10	1	100000	0	0

Table 1 shows that, we have 10 nodes each of which has a specification and defined threshold. As we can see, node 10 has the lowest specification, thus it has the lowest threshold too. Node 7 is chosen randomly to be as a coordinator node whereas a backup is assigned to node 1. We assume having a case study of nodes and their load running for 200 time units. For simplicity, we just display the odd nodes in our Figures. It is clear from Figure 4 that node 1 is extremely over loaded for a long time. Node 3 fluctuates between overloaded and lightly loaded. Node 5 and 7 are rarely overloaded whereas node 9 is completely lightly loaded.

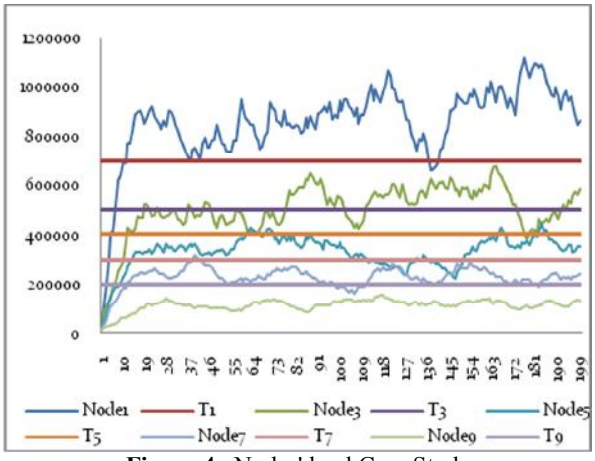

Figure 4. Nodes' load Case Study.

We have performed a number of experiments on the same case study as an input for the task of comparison between the CBAE load balancing algorithm and one of the well known random load-balancing algorithms [4]. To measure CBAE performance, we evaluate and compare three main factors:

1. Node's load which represents the total PODs' size in a particular time unit for each node.
2. Number of communication messages needed for migrating one POD.
3. Freezing time which is the time a specific POD spent during the migration process.

The CBAE mechanism works by selecting a coordinator and backup nodes initially, as shown in Table 1. Applying the CBAE algorithm to balance all nodes yields significant results. Clearly, the gaps are fully utilized as a result of load-balancing and POD migration. One of the main important issues in load balancing is to keep all machines' load-balanced and ensure that none is idle while others are fully busy. CBAE guarantees that, whenever a node is overloaded, the lightly loaded node contributes by taking such a POD from an overloaded node. Thus, almost all nodes are busy running either their own processes or the migrated processes from other nodes. Figure 5 shows the load of all nodes during 200 time units. The load of node 1 is sharply reduced and, in 91 time units its load is less than or equal to the threshold. The remaining nodes, 3, 5, 7 and 9, are collaborating efficiently to share the load from node 1 and they are fully balanced. The CBAE load-balancing mechanism accomplishes its job by sending a number of communication messages. In each migrating POD, the CBAE needs to exchange 4 communication messages per POD. Two messages are needed to request and for a reply from the coordinator node. In addition two communication messages are needed to check the destination availability by exchanging acknowledgements with a destination. So, we can formalize the number of communication

message per node as follow. If node A is overloaded, then the communication messages needed to migrate one POD are 4. Figure 6 shows the number of communication messages issued for each time unit. The freezing time, which concerns the time needed to select one proper destination and migrate a POD to it, is very low in the CBAE mechanism.

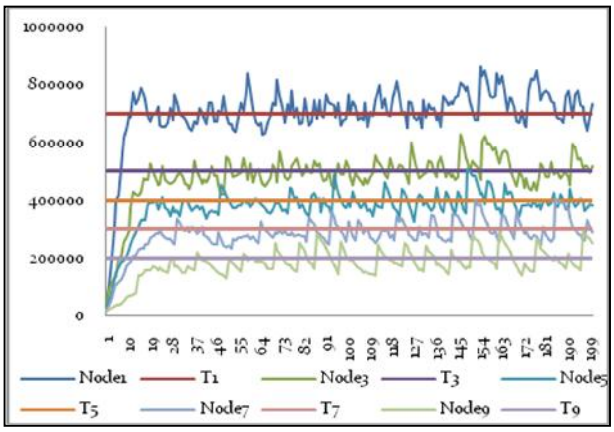

Figure 5. Nodes' load with CBAE load-balancing.

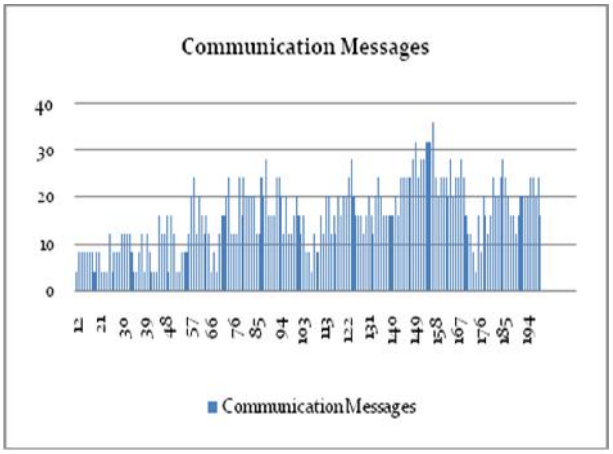

Figure 6. Communication Messages for CBAE Load Balancing.

Figure 7 shows the freezing time for each migrated POD by using CBAE. The majority of PODs have a freezing time less than 4.5 ms. The average freezing time is 4.084 ms. The shorter freezing time results because the overloaded node needs just to ask a coordinator for an appropriate light load. This information kept by the coordinator has already been stored via coordinator in advance. So, the POD just needs to wait for a reply from the coordinator, and then it is directly migrated. The CBAE load-balancing mechanism increases the systems reliability by introducing the concept of backup and the automatic election. We simulate two cases when the coordinator fails alone and when both coordinator and backup fail. Figure 8 shows the case of coordinator failure. As we can see the node 7 is assigned randomly to be coordinator. At the time unit 50, the coordinator fails, so the backup comes up and takes the duty of the coordinator.

The overall load-balancing mechanism works fine and guarantees the reliability. The worst case is represented when both coordinator and backup nodes fail. In this case the overloaded node which needs the service of migration elects itself to act as a new coordinator. As we can see, in Figure 9, both node 7 and 1 have failed, which are the coordinator and backup respectively. Applying the backup concept for storing another copy of overall loads' status is worthwhile. The backup node is useful to reduce the number of communication messages needed, in comparison with applying the coordinator concept alone [7]. In section 4.6 we evaluate the number of communication message in the presence of the backup concept and when it is omitted.

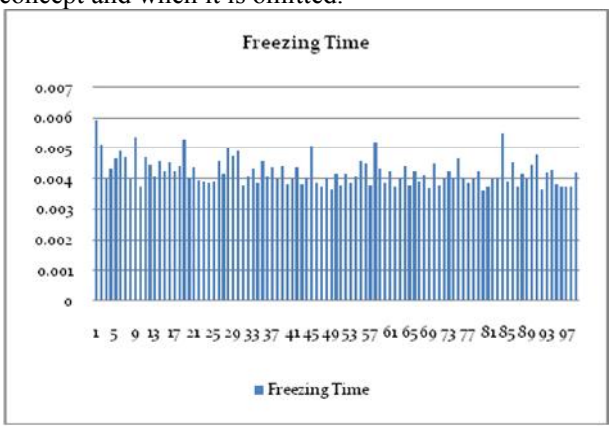

Figure 7. Nodes' load Case Study.

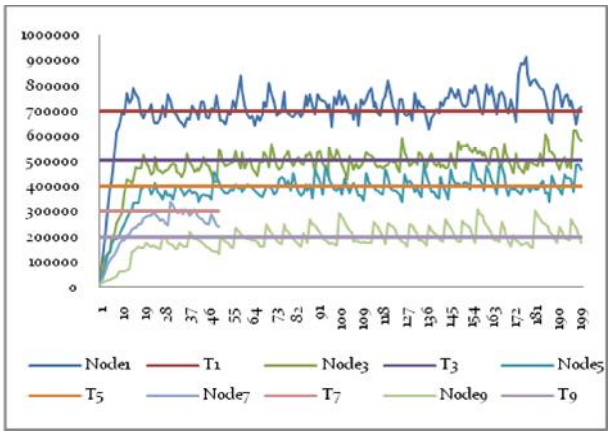

Figure 8. Coordinator Failure.

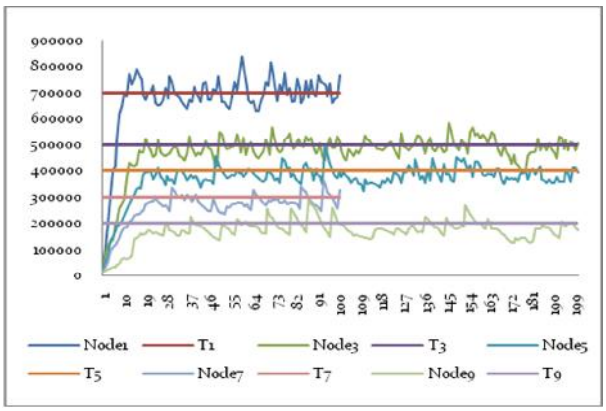

Figure 9. Coordinator and Backup Failure.

4.3 Random Load-Balance Mechanism

The random load-balancing algorithm can be implemented in different ways. An overloaded node selects one node randomly and then migrates one of its PODs to that destination [4]. If a destination node is already overloaded, then this destination node will migrate that POD to another node, that chosen randomly. A variation of this mechanism is done by allowing an overloaded node to pull the destination node before migrating step [4, 10]. If the destination node is overloaded, or if the migration will put the destination node in an overloaded state, then the POD continues running locally. Figure 8 shows the load-balancing algorithm based on random selection in the same case-study. The random approach efficiency depends on the possibility of choosing a non-overloaded node. This possibility increases whenever the number of nodes increases. As we can see in Figure 10, the load among all nodes is totally unbalanced. Node 1 is still overloaded at 168 time units; whereas node 9 is a lightly loaded node for 196 time units. The number of communication messages needed in the random load-balancing mechanism is very low compared with what we evaluated in CBAE. The random algorithm needs only 2 communication messages before migrating a specific POD whereas CBAE consumes twice what the random needs. Although the number of communication messages in the random algorithm is very low, it does not help much to balance the load over all nodes. The overall communication messages needed by the random algorithm are shown in Figure 11.

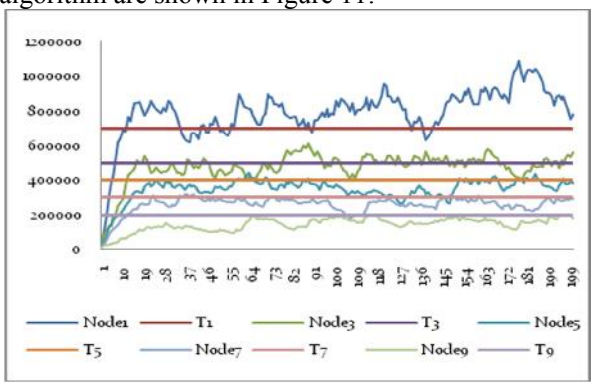

Figure 10. Nodes' load with Random Load-Balancing.

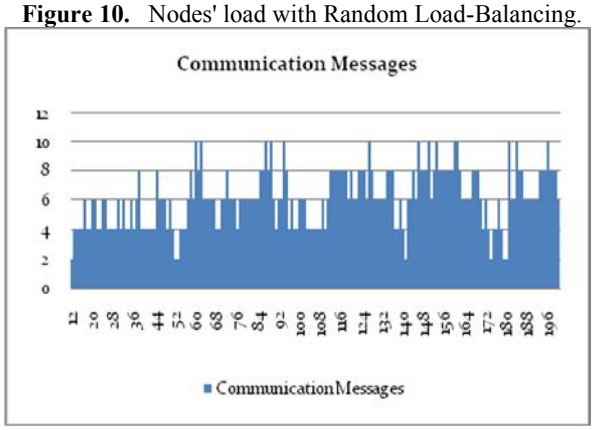

Figure 11. Coordinator and Backup Failure.

A candidate POD for migrating is chosen randomly from an overloaded node. This overloaded node then asks the destination node about its load and threshold.

After that the overloaded node checks if the destination node is overloaded. If the migration process will put the destination node in an overloaded situation, then the POD continues locally. This mechanism significantly increases the freeze time for a candidate POD. Although the number of communication messages shown in Figure 11 is fewer than the number resulting from CBAE, the average freeze time of a random approach is three times the CBAE average freeze time. Figure 12 shows the freezing time for a migrated POD during 200 time units. The majority of migrated PODs have freezing time less than or equal to 9 ms, and this value is very high compared with CBAE. The average freezing time for PODs is 9.252 ms. The freezing time has to be as small as possible since it affects the overall performance and the overall throughput.

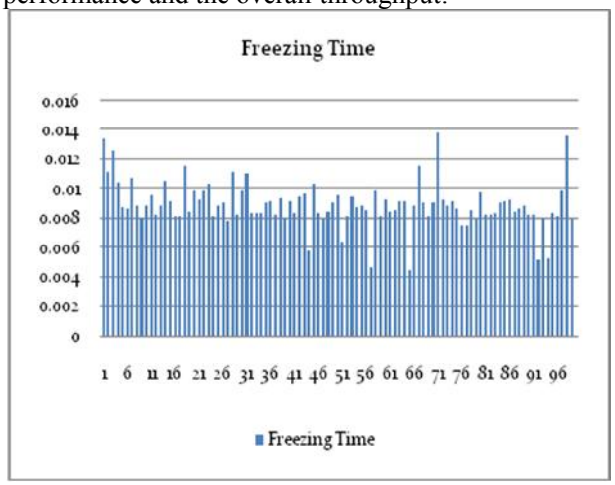

Figure 12. Freezing Time for migrated PODs using CBAE.

4.4 The Benefit of Backup

Let us consider the case of being able to elect coordinator concept. In this situation, each time a coordinator fails the overloaded node elects itself to act as a new coordinator. Each time the coordinator fails, the new coordinator needs to send its election to all other nodes and receive from them their current load status. Thus, the number of communication messages increases. Assuming 50 nodes and applying the previous case (only coordinator and the ability of election), and our proposed CBAE to compare the number of communications needed. As we can see in Figure 13, the coordinator has failed 13 times. Applying only the coordinator and election load-balancing increases the communication messages needed in case of coordinator failure. The CBAE algorithm which depends on backup in case of coordinator failure needs half the communication messages compared with the mechanism without the backup concept.

In CBAE, the only communication messages needed are to inform the other active nodes to change the coordinator address information to the new one, which is the backup.

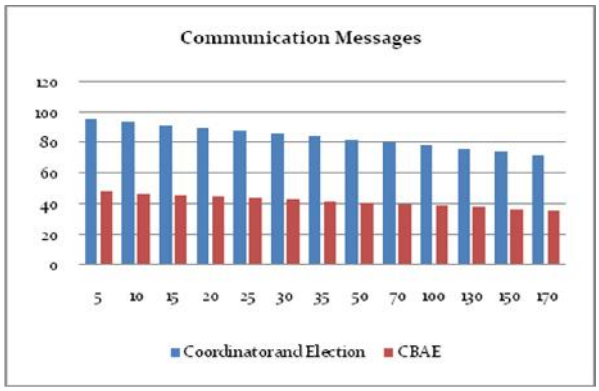

Figure 13. Communication Messages needed in case of Coordinator Failure.

5. Conclusion

In this paper, we have proposed a fault-tolerant and reliable load balancing CBAE algorithm that works by assigning one node as a coordinator and another node as a backup. We compare the CBAE with a well-known random election algorithm. Results show that all nodes are balanced by using CBAE, whereas by using a random approach they are obviously imbalanced. So we conclude that CBAE produces a balanced load among nodes as well as low freezing time. The random approach is better in relation to the number of communication messages needed. Some attributes cannot be measured by simulation, e.g. the network traffic and the actual freezing time that is affected by the network delay. As a future work in this context, therefore, we plan to verify our simulation by experiments in a real environment.

Acknowledgment

We would like to thank King Fahd University of Petroleum and Minerals for providing the computing facilities. Special thanks to Mr. David Birkett for his help in editing the paper.

References

[1] S. Dejan Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, Songnian Zhou, "Process migration", ACM Computing Surveys, pp. 241-299, Vol. 32, Issue 3, Sept.2000.

[2] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," IEEE Transactions on Software Engineering, vol. SE-12, pp. 662–675, May 1986.

[3] D. Frederick, "Transparent Process Migration in the Sprite Operating System (PhD Thesis, University of California, Berkeley), September 1990.

[4] J. Jahnich, Achim Rettberg, "Towards Dynamic Load Balancing for Distributed Embedded Automotive Systems, Springer, Volume 231 Embedded System Design: Topics, Techniques and Trends, 2007.

[5] J. Casas, D. Clark, R. Konoru, S. Otto, R. Prouty, J Walpole, "MPVM: A Migration Transparent Version of PVM", OGI Technical Report, Feb 1995.

[6] J.M. Bahi, C. Vivier, and R. Couturier, "Dynamic Load Balancing and Efficient Load Estimators for Asynchronous Iterative Algorithms," IEEE Trans. Parallel and Distributed Systems, vol. 16, no. 4, Apr. 2005.

[7] Luís Paulo Peixoto, Escola De Engenharia, Peixoto Santos, Peixoto Santos," Load Distribution: a Survey, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.2301.

[8] L. K. Subrata, R. Zomaya, A.Y., "An Efficient Load Balancing Algorithm for Heterogeneous Grid Systems Considering Desirability of Grid Sites", pp. 320, IEEE Proce. Of the 25th Inter. Conference of Performance, Computing, and Communications, 10-12 April 2006. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1629422

[9] M. Claypool and David Finkel, "Transparent Process Migration for Distributed Applications in a Beowulf Cluster", In Proceedings of the International Network Conference (INC) Plymouth, United Kingdom, July 16th-18th, 2002.

[10] M.M. Hayat, S. Dhakal, C.T. Abdallah, J.D. Birdwell, and J. Chiasson, "Dynamic Time Delay Models for Load Balancing. Part II: Stochastic Analysis of the Effect of Delay Uncertainty," Advances in Time Delay Systems, vol. 38, pp. 355-368, Springer-Verlag, 2004.

[11] M. Mitzenmacher, "The power of Two Choices in Randomized Load Balancing", IEEE Trans. Parallel and Distributed Systems, vol. 12, no. 10, pp. 1094-1104, 2001.

[12] R. Krahl and Jörg Nolte and Lars Büttner, "A Load Balancing Approach for the PEACE Operating System", 1993.

[13] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The Design and Implementation of Zap: A System for Migrating Computing Environments", In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, Dec. 2002.

[14] S. Penmatsa, Anthony T. Chronopoulos, "Dynamic Multi-User Load Balancing in Distributed Systems. IEEE IPDPS", pp. 1-10, 2007.

[15] S. Dhakal, Majeed M. Hayat, Jorge E. Pezoa, Cundong Yang, David A. Bader, "Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach," IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 4, pp. 485-497, April 2007.

[16] C. Wills, and Finkel, D., "Scalable Approaches to Load Sharing in the Presence of

Multicasting", Computer Communications, 18(9), pp. 620-630, 1995.
[17] W. Osser, "Automatic Process Selection for Load Balancing", MS Thesis, University of California, 1992.
[18] Tarek Helmy, S. A. Shahab, "Machine Learning-Based Adaptive Load Balancing Middleware Framework for Distributed Object Computing", Springer LNCS Journal, Volume 3947, pp. 488 – 497, 2006.
[19] Tarek Helmy, Syed S. Jafri, "Avoidance of Priority Inversion in Real Time Systems Based on Resource Restoration", International Journal of Computer Science and Applications (IJCSA), Vol. III, No. I, 2006, pp. 40–50.
[20] Sunil Thulasidasan, "Issues in Process Migration, USC, 2000, http://public.lanl.gov/sunil/pubs/csci555tp.pdf
[21] Tarek Helmy, Irfan Ahmed, Aleem Alvi "A Framework for Fair and Reliable Resource Sharing in Distributed Systems", Chapter of Book Titled: Distributed and Parallel systems, in focus: Desktop Grid Computing, pp. 115-128, 2008. Springer Publisher, ISBN: 978-0-387-79447-1.

Tarek Helmy is currently with the department of Information and Computer Science, College of Computer Science and Engineering at King Fahd University of Petroleum and Minerals (KFUPM). On leave from the College of Engineering, Department of Computers Engineering and Automatic Control, Tanta University, Egypt. He received his Ph.D. in Intelligent Systems from Kyushu University, Japan, in 2002. His research interests include Distributed Operating Systems, Multi-Agent Systems, Personalized Web Services, and Cooperative Intelligent Systems. He has published more than 50 papers in major international journals and conferences in the fields of cooperative intelligent agents, artificial intelligence distributed operating systems and computational intelligence. Dr. Helmy is on the program/organizing committee of various international journals/conferences in the fields of artificial intelligence, multi-agents, intelligent and distributed systems.

Fahad Al-Otabi is a graduate assistant at King Abdulaziz University, Faculty of Education, Department of Computers, Kingdom of Saudi Arabia. He received his bachelor degree in a Computer Education from Jeddah Teachers' Collage of King Abdulaziz University. He is currently a master candidate in the School of Computer Science at the University of Hertfordshire, UK.