Benchmark Framework for a Load Balancing Single System Image

Bestoun S. Ahmed †, Khairulmizam Samsudin†† and Abdul Rahman Ramli†††

†Department of Computer & Communication Systems Engineering,
University Putra Malaysia, 43400 Serdang, Selangor, MALAYSIA

Summary
Recent developments in the load balancing single system image (SSI) clusters enabled workstations to provide a cost effective and high performance environment which has become increasingly attractive to many users. However, in practice, clusters of workstation failed to exploit their performance potential advantages. This paper presents and propose a framework for benchmarking and performance evaluation of a load balancing SSI and shows how this framework used as a methodology for a comprehensive examination of load balancing SSI clusters performance and behavior.

Key words:
Single System Image, NOWs (Network of Workstations), Load balancing algorithm, Distributed systems, openMosix, MOSIX.

1. Introduction

Cluster as a popular plate form for executing computationally intense applications becomes very important nowadays. The major objective in the cluster is utilizing a group of processing nodes so as to complete the assigned job in a minimum amount of time by working cooperatively. The main and important strategy to achieve such objective is by transferring the extra loads from busy nodes to idle nodes.

In another hand, single system image (SSI) as an important part of clusters, provides the delusion of a single powerful and high available computer to user and programmers of a cluster. The most important paradigm of SSI over other types of clusters are the needless of paralyze code, automatic process migration, i.e. load balancing (LB) [1]. That is means simply it frees the end user to know where the application will run and where resources is located for such application.

Kai Hwang and Hai Jin [2] identifies the useful and available services of SSI including single entry point, single control point, single job management single file hierarchy and other services of SSI. Rajkumer [3] introduces several level of single system image including hardware level, software level, application level, middle ware level and operating system level. He and then Christine Morin in here research [1]; they mention that Sprite, QNX, Genesis and UnixWare are examples of systems providing operating system kernel level SSI in additions to MOSIX. Each of them is designed for a single or more than purpose. The performance of these systems differs from each other in a different manner depending on the design of these systems. The operating system layer has been an important layer because of its ease of use. In such layer, most of the mechanisms that must be used in the clustering hide by means user do not interact with the system and the complexity of its implementation [3].

Load balancing mechanism in this type of clusters plays the major part of performance. The main part of this feature is the algorithm that responsible for load balancing between nodes operating systems. Although there are many implementations of SSI as mentioned, including OpenSSI [4] and Kerrighed [5], MOSIX [5] and openMosix [6] are promising for providing load-balancing operating system for high performance SSI. Due to opensource availability of openMosix, it makes an attractive choice for load balancing SSI research.

Our earlier work [7] on load balancing single system image focuses on describing a performance model of such system. The results lead us to investigate a dramatic analyze of an existing opensource solution. Based on those results, this paper presents benchmark

Manuscript received May 5, 2008
Manuscript revised May 20, 2008

framework for load balancing SSI clusters. This practical benchmark framework helps us to understand the performance of load balancing SSI. The goal is to develop strategies for understanding the performance of load balancing SSI that could provide crucial information to SSI designs for improving the performance of the system. This framework is shown to be effective and practical for solving a specific load balancing design issues. It is also demonstrating improved performance by enhancing the load vector management of openMosix as compared with original algorithm.

The need to make a performance evaluation, benchmarking and knowing the behaviour of this kind of cluster is far more than curiosity about how fast the system runs. Due to the complexity of implementing a load balancing SSI, much of the work took the form of simulations. The limitations of simulation research in this area demonstrate the need for empirical study of implemented systems. Existing researches on the performance evaluation and benchmarking focus on the Beowulf cluster that has a different approach due to its dependencies on MPI [8], PVM [9] or any other middleware libraries [11]. In another hand, most (if not all) of the standard benchmarks also depend on MPI or PVM or any other libraries that have different approach since they use a simple static job assignment algorithm that completely ignores the current state of dynamic load-balancing algorithm in SSI based system [10]. Thus with simple time measurement and with the current benchmark, the performance cannot be summarized and be known in Load balancing single system image.

The reminder of this paper is organized as follows: After reviewing on a brief background of related topics of this work in section 2, section 3 is dedicated for the explanation of openMosix architecture and describes openMosix load balancing mechanism in detail. Section 4 gives an introduction for the benchmarking and a brief description of the existing metrics for the performance and their use in our research. Section 5 will give a good description about the research method by describing the testbed of the research and the experimental procedure. Finally, the results of the framework and the modification of the system followed by the discussion.

2. Background

The main goals of SSI clusters are complete transparency of the resources management, scalable performance and system availability [12]. Furthermore, it provides a dilution of a single powerful computer to users or programmers. From the performance point of view in SSI, the main feature is the strategy it takes to balance the load around the nodes.

The implemented and simulated strategies of load balancing fall mostly in to either one of two classes static or dynamic. With static load balancing a single system image like any multi computer system, distributes tasks across nodes by using priory known information of the tasks and the load distribution remains unchanged during running time. In contrast with this, by dynamic load balancing there is no priory information about the tasks, as a result the task distribution decision held during running time.

In turn, dynamic load balancing can be either centralized or decentralized. In the centralized load-balancing scheme, there is a single node responsible for all the decisions in the whole system. While in decentralized load balancing, the central node can be removed in a way that each node communicates to each other and can decide directly [11]. Dynamic load balancing becomes an attractive technology now days because of its use in SSI operating systems widely [1].

In the cluster of workstation, the load balancing becomes effective when there is accurate knowledge of the state of individual node around the cluster. This is used for accurate assignment of the task to the appropriate nodes. The Information collection and dissemination

algorithm manages how this load information communicates to global task schedule. For such purpose, either broadcast or multi cast or probabilistic mechanism holds information exchange within decentralized systems. In broadcast mechanism, each node periodically broadcast its load information to each node in the cluster. By this way, each node receives and processes a number of messages that equal to the number of nodes in the cluster [12]. While in multicast mechanism, load information messages are sent to members of certain multicast group to limit the drawback of the traffic in the formal one. The probabilistic mechanism method tries to minimize the information messages between nodes in the decentralized algorithm by making the algorithm to send messages to a specific number of nodes randomly in the cluster instead of sending messages to all.

The studies in this area are distributed generally in two directions: simulation studies and experimental studies. The simulation studies are done to simulate and study each component alone and then they implemented in the real work. It is important to mention that most of the studies on load balancing in distributed systems took place during 80's to late 90's. It is clear also that due to the complexity of implementation, most of these work done by simulation. Most of these simulation studies deals with the algorithms that balance the loads around a cluster of nodes and the comparison between them [13, 14, 15].

By the developments of a lot of simulation studies and load balancing strategies, different research groups take advantages of these studies to implement their load balancing SSI operating systems like Sprite, QNX, Genesis UnixWare, OpenSSI, Kerrighed and MOSIX as mentioned before. For this reason, different studies are carried out to show the behaviour and difference between these systems in specific properties. Earlier, OpenSSI, Kerrighed and MOSIX systems took the major part of the research between these systems due to their attractive features and due to their availability in opensource.

In this area of research, an important study can be found in [16]. In that research, a comparative study of these three modern SSI operating system for clusters was presented. The research examines and evaluates some specific features in these three systems. Although it was an experimental study, but the research examine each system according to some specific features and state more on file system and kernel design and communication. Most of the OpenSSI feature take advantage from MOSIX, especially the process migration and load balancing mechanism while MOSIX group recently changes the licence from opensource and as a result openMosix derived from MOSIX to be a real alternate opensource of MOSIX. Although the development of openMosix for 2.6 kernel was stopped but it is still the only opensource version for researcher and the load balancing algorithm is still in use. In another hand Kerrighed is still a research prototype and less robust than the two other system [16].

3. OpenMosix Architecture

OpenMosix is an open source project forked from MOSIX. Most of its design is similar to that of MOSIX. OpenMosix balance the load on the CPU by means of using classical load balancing. OpenMosix implemented on a Linux standard kernel by extending it. The main feature is its load balancing mechanism. This load balancing mechanism tends to balance the processes on processors around the nodes by migrating extra processes. For such case, a deputation introduced inside the kernel as a similar case with kernel thread. This deputation keeps a record of migrated processes. As a result, when a process running it appears to run on the node on which it was spawned that is known as Unique Home Node (UHN) even it migrated elsewhere by keeping a representative named deputy [17]. Whenever possible a process

uses local resources, but often has to make system calls on its UHN. The migrated user context that is called the remote contains all data about the processes such as code, stack, data, memory maps and even registers. As long as the remote needs system call, openMosix intercept all site dependent system call and forward them to its deputy from remote node to UHN.

The main tool for the resource management algorithm is the pre-emptive process migration (PPM). As long as the requirements for resources, such as CPU are below certain threshold point, all users processes are restricted to their home node. When these requirements exceed the CPU threshold levels, some processes will be migrated transparently to other nodes [17].Memory management is provided in OpenMosix through a memory ushering. This algorithm will be active when a free memory of a node falls below a threshold value and OpenMosix attempts to transfer process to other nodes, which have sufficient free memory. A better understanding of this architecture and mechanism can be introduced as in the figure 1.

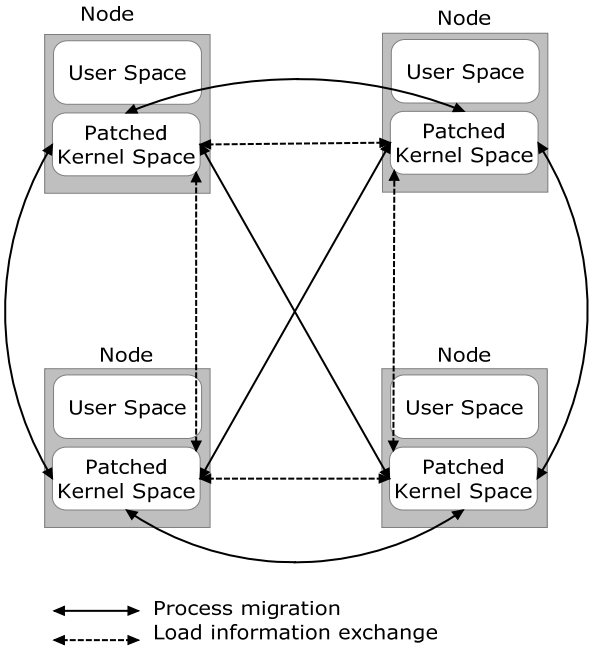

Fig.1 OpenMosix architecture and work mechanism

3.1. Load Balancing in OpenMosix

The main load-balancing component of OpenMosix is the information dissemination, process migration and memory migration.

The information dissemination daemon as noted in fig.2, disseminate load balancing information to other nodes in the cluster. The daemon runs on each node and is responsible for sending and receiving load information to and from other nodes. The sending part of this daemon will periodically send load information each second to two randomly selected nodes. The first node is selected from all nodes that have contacted the node "recently" with their load information. The second node is chosen from any nodes in the cluster [19]. The receiving portion of the information dissemination daemon receives the load information and attempt to replace information in the local load vector. The standard implementation simply utilizes a first-in-first-out (FIFO) queue of eight entries. Thus, the oldest information is overwritten by newly received information [20].

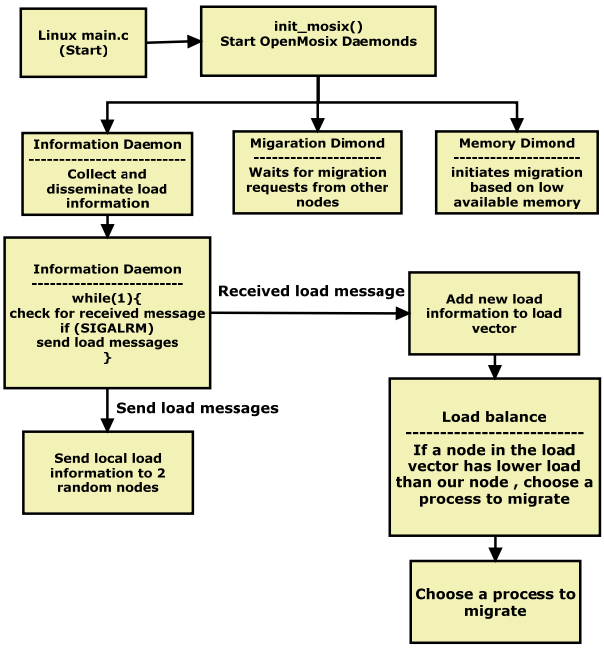

Fig.2 Load information Dissemination and Collection Management

4. Benchmark strategy

Benchmarking simply means measuring the speed of a given task on a given system to discover the performance and behaviour of a system. Performance evaluation by benchmarking is the primary method for measuring the performance of any system. In addition to that, it can be done in a way to allow a comparison between different hardware/software combinations [21]. In this way, it deals with facts and figures not opinion or approximation. Benchmarking carries out to conduct two main points: first is providing a means of system comparison and the second is allowing system performance and behaviour estimation [25].

Before starting any benchmarking process, decision must be made to use either synthetic benchmark or application benchmark. Synthetic benchmarks are designed specifically to know and measure the performance of individual component of a computer system by examining the component to its maximum capacity. Whetstone suite is a well-known example of this kind of benchmarks that was originally programmed in 1972 [22].

In another hand, the recent benchmarking techniques are going to the direction of choosing a common application and use it to test the performance of complete system. This direction comes to face by the starting and developing in the network technologies and distributed system.

The benchmark frame works vary from single computer to a cluster of computers. It is also varied by the variation of the system. Most of the benchmarks take the execution as a main parameter to benchmark. As a result, the benchmarks results are useful for measuring the performance of the system as seen by the user, but provide little information to a system designer and to the researcher [23]. As a result, the needs of benchmark framework coming to face for the designer to give a methodology for those designers who want to know their information more than execution time. For such framework different metrics must be chosen and different factors must be selected to know the behaviour. The following sections give brief information about existing performance metrics that is used in this research to help us to form the benchmark framework later.

4.1. Existing performance metrics

Different performance metric has been declared to know the performance of a specific system in a specific case. Existing performance metrics are derived either from a common benchmark program or individually depending on the kind of the system itself and their meaningfulness; also it depends on ease of measurement. These existing performance metrics vary from metrics defined for a single computer to the metrics defined for the Beowulf systems. In addition, there are common metrics between the two types. In the following section we will explain in detail some performance metrics and its use or useless in our research.

4.2. Run Time

Run time performance metric simply means the total time for completing a given job without considering the number of operations performed. It is the standard performance measurement for systems running the same application. This metrics used commonly in the first time when predicting the performance of any system.

Since the essential aim of load balancing algorithm is the improvement of mean response time of the running program, therefore, the main observation and focusing point is on the Mean Response Time (MRT).

4.3. Speedup

Speedup measure is the ratio between the time required to solve problem on a single processor node to the time taken to solve the same problem using more than one node. This performance

metrics is suitable to measure the scalability of the OpenMosix cluster, as the scalability is a feature of performance [24]. Speed-up is given by:

$$\text{Speed-up}=T(1)/T(N) \quad (1)$$

Where T (1) is the run time for a particular program on a single node and T (N) is the run time on N nodes.
It is important to mention that speedup of a specific benchmark on a specific hardware system cannot be compared with the speedup of such benchmark on a different hardware system due to the difference of a hardware specification [24].
Therefore, we measure the speedup of the benchmark on same system for different configurations as a second step of the benchmark framework concurrently with the efficiency performance metric.

4.4. Efficiency

Efficiency is defined as the percentage of speed up divided by the number of nodes. Efficiency is given by:

$$\text{Efficiency} = \text{speed-up} * 100 / \text{number of nodes} \quad (2)$$

It is very useful to measure the percentage of a node time spent in useful problem solving. A rigorous analysis of the efficiency metric can be used to identify dominant overhead factors and measure it in practice. In another words, cluster efficiency also demonstrate the overhead of the system as well [24].

5. The Research Method

In this section, we describe the empirical methodology that used to achieve the aim of this research. This has been done by describing the cluster that built for the purpose of this research then going through the experimental procedure.

5.1. Hardware/Software Testbed

As shown in fig.3, we used openMosix to implement a load balancing single system image that described in Section 3 for our experiments.

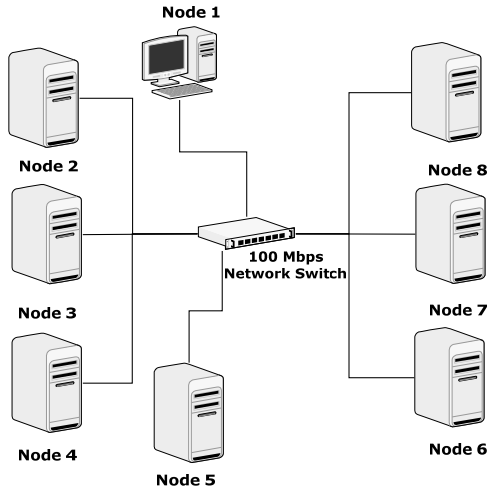

Fig.3 Load Balancing Experiment Testbed

As mentioned, the reason behind using openMosix is that it is the opensource version of MOSIX that is the recommended solution nowadays for load balancing SSI. In addition, its load-balancing algorithm is clear and said to be efficient nowadays.
In our experiments we use eight commodity off the shelf (COTS) single CPU nodes based on Intel Pentium 4, 1.6 GHz processor with 256 MB of main memory, all running the Fedora Core 1 GNU/Linux distribution with 2.4.26 kernel, which supports the requirements of installation and compilation of kernel with gcc 3.3.2 that is patched with openMosix 2.4.26 patch. All nodes were interconnected with star topology using an Ethernet 10/100 Mbit/s switch. There is no graphic card on the nodes except for the main node to reach the low cost requirements.
All nodes in the system are homogeneous. This will help us to neglect the mismatch of processing speeds that may cause additional process coordination complexity and therefore affect the execution performance of the programs.

Each performance metrics considered has been measured by using a Distributed Key Generator (DKG) [25]. The program generates 4000 RSA public and private key pairs with 1024 bits. To ensure that the load exceed the CPU threshold of our cluster we modify the program so that the parent distribute computation to a specific number of child processes using fork () from the default value of four as will be shown in the results section. It is important to mention here that although in the systems that depends on libraries like MPI or PVM, the algorithm of the program that written plays the major part of the performance since the performance will varied from algorithm to others. However, in dynamic load balancing SSI, since the scheduling and distribution of the job is dynamic, the scheduler can manage processes in the cluster in order to efficiently use available resources in the transparent way from the user [26], and as a result, the performance depends on the system not on the program itself. Only the important point is that program must be migratable as stated before.

5.2. Experimental Set-up

As in any measurement experiment, the consideration of experiment environment and platform variability must be taken into account. In our test bed, as described before, the nodes in the system are homogenous by means all nodes have same hardware specification and same performance individually. This because of the availability of the hardware and it helps us to neglect the factor of the mismatching of processing speeds in different computers that may cause process coordination problems and therefore affect the execution performance of the programs as mentioned before. Therefore, this will reduce the influence of heterogeneous factor of the system. However, as our focus of the research on the dynamic load balancing, the run time may vary from one run to another through the experiment due to the job placement variation. This factor is minimized by running each experiment several times and taking the mean value of each measurement. From the literature, the maximum repetition of the experiment that we note was five times. Therefore, we decide to repeat each experiment six times and take the mean of the observed result to ensure the level of confidence.

The investigation of the framework depends on the aim of the work itself. As we declared in the first and second chapter, our aim of the study is load-balancing algorithm, it is important to distinguish between load balancing algorithm and memory ushering algorithm. Most of the SSI systems depend on the CPU load for balancing, but some SSI systems add the memory load feature for balancing in a separate algorithm as in [18] [27].

In most of the research in this area, they used matrix multiplication to evaluate the performance of the cluster at all but although the matrix multiplication makes a big load on the processer due to its heavy calculation requirements, but it affects the memory ushering algorithm also by means of loading the elements of the matrix to the memory.

In another hand, the behaviour of the system from the performance point of view may vary depending on the workload [26]. As a result, choosing the workload depends on many factors, for example, is the workload affect the memory. Alternatively, is the load using memory ushering algorithm?

As a result, we use DKG as a benchmark for performance evaluation that uses processes to make load on the processor only with a little load on the memory. In this way, we can prove the memory ushering algorithm is inactive due to the little load on the memory.

6. Benchmark framework

As we stressed before, the first step is choosing a suitable load for performance evaluation and benchmark. Then we can investigate the

components of the framework that can be used in the experiments.

In [28], a simulation framework of a load balancing algorithms proposed. Furthermore, the authors stated that any load balancing algorithm depends on some important factors including communication delay, topology, workload and negotiation protocol. In another hand the research concluded that for any simulation study framework of a load balancing algorithm, there are some factors that must take in to account. In real state two factors of those factors must take into account that are the performance evaluation and the cost evaluation.

Depending on the above conclusion, the framework isolates the performance factors of the system in to three main components. As depict in fig.4 clearly.

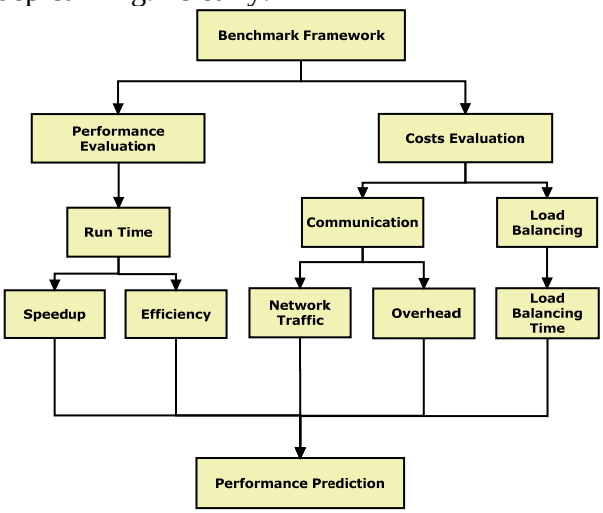

Fig.4 The Components of the Framework

The first component is the overall performance itself; the second is the communication mechanism and finally the load balancing mechanism. As a result, the first component is said to be an overall performance evaluation while the other components is said to be the cost evaluation that evaluate the costs caused by an extra components.

The first components done by three performance metrics, run time, speedup and efficiency. The second component is analysed by traffic measurement of the node and its behaviour then the overhead measurement. The last component is the load balancing mechanism and its time inquired to balance the load around the nodes.

7. Benchmarking Experiments and Results

As we mentioned before, we demonstrate the importance of having enough child process spawned from the parent. The average CPU utilization during the running time in a cluster have been measured for each node without considering time by monitoring all nodes processors since the utilization is the percentage of the time that the CPU is busy. The following figures illustrate the CPU utilization for DKG running with 4-child and 8-child.

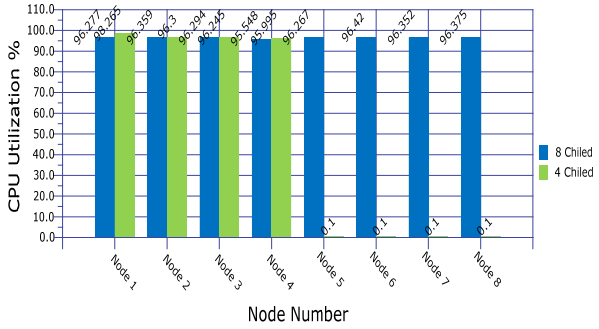

Fig.5 CPU Utilization in Each Node with 4 and 8 child DKG

Fig.5 illustrates CPU utilization for each node in the cluster by executing DKG with four and eight child. We can note that the load generated from 4-child DKG does not exceed CPU threshold of every participate cluster nodes, therefore the processes are not migrated to other nodes. The fifth node was shown to be idle during running time compared with the other utilized nodes at approximately 99%. Therefore, we increase the number of child processes to obtain enough load to conduct the benchmarking process. By executing DKG with 8-child, It is clear from fig.5 also that the load surpass the CPUs threshold as all the nodes in the cluster utilize all of their CPU resources. The importance of this step is to ensure that all nodes

in the cluster will be participating in the process of load sharing and load balancing.

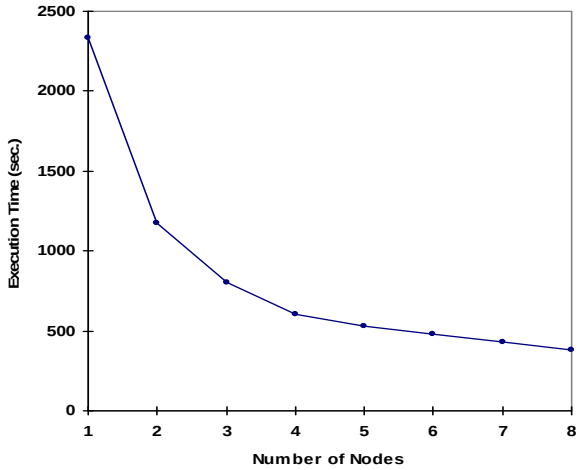

Fig.6 Run Time of DKG on the cluster

As a suitable load for the benchmark has been found, a run time benchmark is conducted to investigate the behaviour of the system under load with varying number of nodes, as shown in Fig. 6.

We note that there is a significant performance improvement between one node and two nodes (nearly 50%); however, there is no significant improvement after having five nodes. We predict that there would be no significant performance gain after eight nodes.

To investigate the performance point of drop and the scalability of the system, the speedup and efficiency metrics will be very important. Fig.7 and Fig.8 illustrate the speed-up and efficiency measurement of DKG on the cluster respectively. It is clear that the speed-up improvement is not linear with the number of nodes, while efficiency of DKG on the cluster starts to degrade after having more than four nodes (nearly 4.37 speedup and 87.58% at five node).

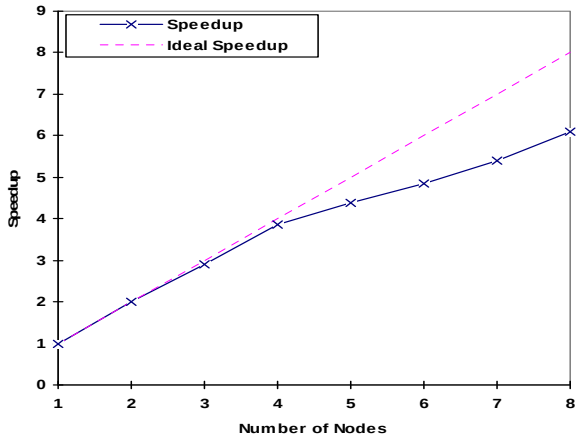

Fig. 7 Speed-up of DKG on OpenMosix cluster

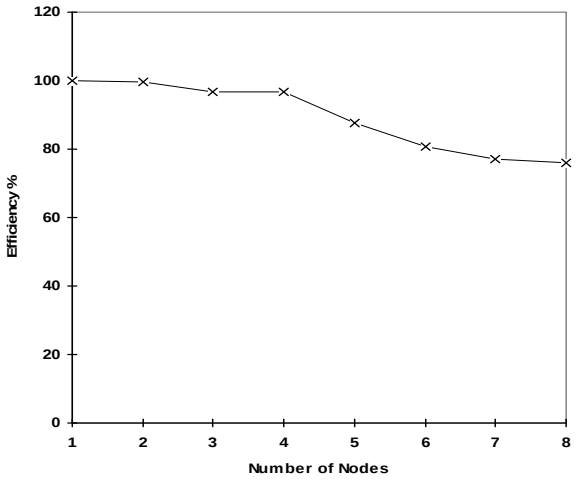

Fig. 8 Efficiency of DKG on OpenMosix Cluster

The result from fig.7 is in line with what was discovered in [16] regardless of the nodes specification and the value of speedup, with the difference of number of nodes and the benchmark program.

To identify the cause of speed-up and efficiency degradation, that is lead us to expect extra overhead , we investigate the effect of network traffic and the process migration overhead and extra overhead on the performance of OpenMosix. The traffic measurements are held by measuring the traffic on the home node while executing 8-child DKG in a controlled network environment. IPtraf [29] was used to perform the measurement of network traffic as it is recommended and used by [30].

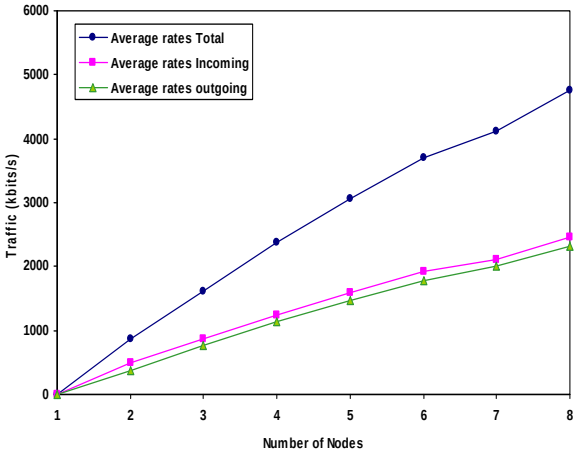

Fig. 9 Nodes Relation with Traffic

Fig.9 shows the network traffic generated on the home node and the actual inbound and outbound traffic of the network interface. Obviously, network traffic increases with each additional node (by approximately 50% with each node addition) and as a result contributes to the degradation of efficiency on performing calculation.

The increases of the traffic influence the home node in such a way that the performance of the cluster itself will noticeably drop. Furthermore, the deputation mechanism will be the main responsible for this increasing in traffic.

Communication among nodes in the OpenMosix cluster relies on both TCP and UDP protocol. TCP is used to migrate the processes while UDP datagrams are used solely to send load messages [19]. However, from the measurement, the amount of UDP traffic is insignificant compared to TCP. As a result, we can say that the traffic is increased by adding more nodes since the deputation mechanism decrease.

The results are completed by an experiment of the process migration overhead. We use the method that was used in [31]. The overhead is measured by starting the application on the home node and migrate it to a remote node to know the migration overhead time caused by the process during the running time. The measurement has been done in two cases, first with idle nodes then with loaded nodes by four processes DKG. We run a dummy cycle with a large number of loops for this test as used in [32]. These cycles have a processing cost with a good time to know the overhead costs. The program runs in the home node and then runs the same process in the remote node from the home node the difference of the time will be the overhead.

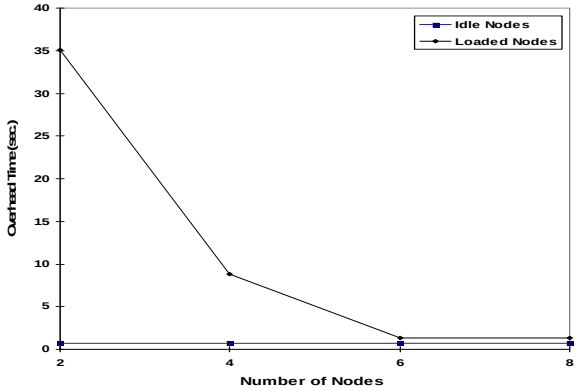

Fig. 10 Process Migration Overhead Time

As we noted from fig .10, the process migration overhead measured by adding two nodes at a time for better recognition of overhead since this time cannot recognized clearly by adding single node at a time. The overhead is said to be constant value of time and the changes by adding more nodes will be very small and cannot be recognized in real time systems under no load condition. In the next matter, when the cluster loaded by running four processes DKG, the traffic will increase by increasing the number of nodes as shown in fig .10. The increasing is going to be constant at 5 nodes and above since the load cannot exceed 4 nodes as shown in fig.5, in such case the system calls made between 4 nodes only and the other nodes will be idle. In the beginning at two nodes, the overhead is said to be large. This overhead is because of the processing time the CPU takes for DKG, so the processor took more than the exact time to process dummy cycle and DKG concurrently and this is called CPU overhead. The important case that we want to stress on is the case of after four nodes when there are enough nodes for processing DKG. We can note from fig.10 that the overhead going to be

constant but with a slight different with the case of no load. This caused by the transferring of memory pages and data stacks through the network to the destination node that will make extra overhead. This traffic will eliminate the data gathering from the network. Such elimination leads the processes that are running on a processor to go to ready queue since the system calls need the home node. This will be accepted with what H. Justin and F. Wu-chun said in their research [33]. This caused by an interrupt that make the CPU to handle extra job before continuing its original load. In another hands, when the process needs to access data on the home node, it must wait for the request of data.

In addition, the time that the system takes to balance the load on all the participating node around the cluster will be very important as well as the factors that affect this time. This time is the time when equal number of processes started in the home node and migrates freely to other nodes till the load on these nodes become equal. This load balancing time was measured by using a special method benefits from [34]. As we note from fig.11, the load balancing time increase by adding more nodes.

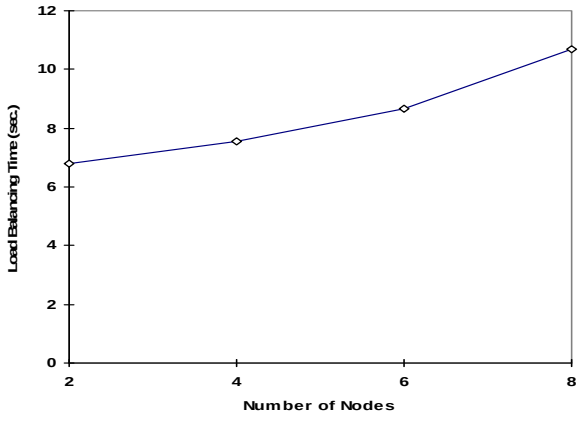

Fig.11 Load balancing Time

This increasing of time caused by the time the nodes take to exchange information about their state and the time of decision and choosing suitable nodes as shown in the flow chart in fig.2.

From the above result, we can note several important factors that could improve the performance of the cluster one of these factors is the time that the cluster takes to balance the load between all nodes. As stated in section three, due to the probabilistic and decentralized nature of load message dissemination, OpenMosix would always update the local load vector information (LVM) without considering the advantages to the current node. This leads us to modify the existing load vector information management in the kernel. Our approach is to update the load vector only when the load information from the remote node is beneficial (Lremote < Llocal) to the existing node. This load vector management modification shown in the following simplified flow chart. This modification will allow the information about the nodes load to be disseminating around the cluster more efficiently as compared with the original one.

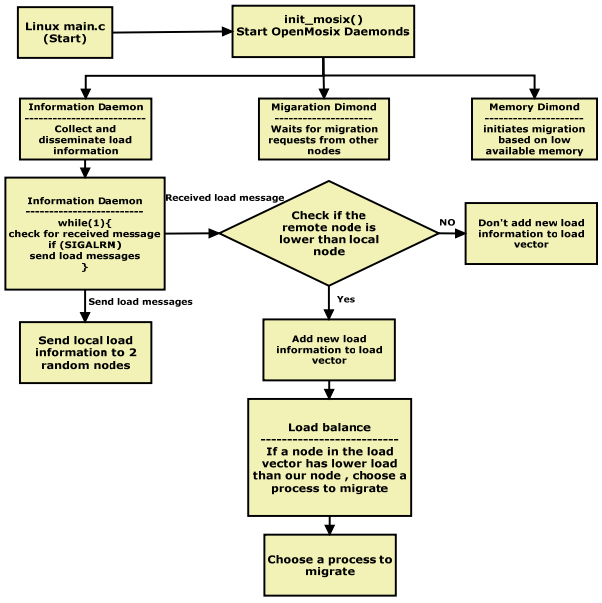

Fig.12 Modified Load information Dissemination and Collection Management

This modification derives us to repeat all the experiments of our framework to know the exact modification and its effect on the various costs.

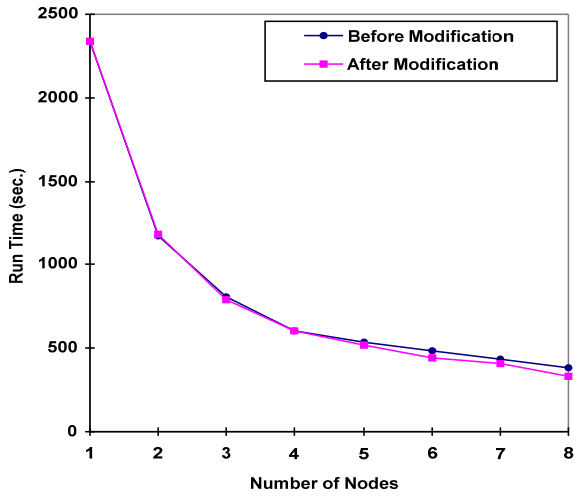

Fig.13 Comparison of run time for standard and improved LVM

Fig.13. shows the run time comparison between the standard OpenMosix LVM policy and the improved LVM. There is visible additional performance gain after five nodes. With eight nodes, a gain of 50 second of run-time performance is achieved. However, we cannot expect more improvement in performance because of our small-scale cluster.

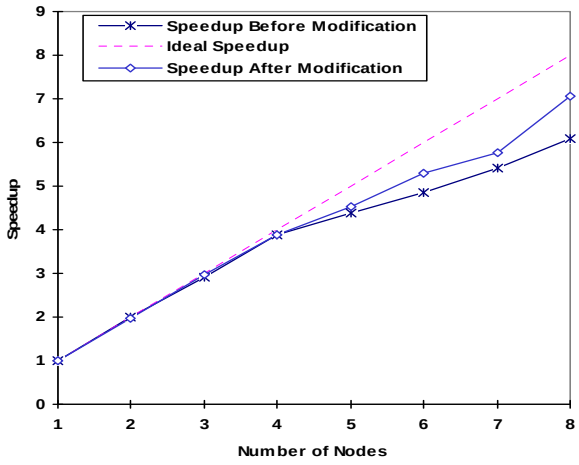

Fig.14 Comparison of speedup for standard and improved LVM

Fig.14 shows the speed up of running DKG using standard OpenMosix LVM compared to the improved LVM policy. The modified LVM has a significant performance gain over the standard LVM. This is due to the efficient information dissemination around the cluster that leads to better load balancing strategy.

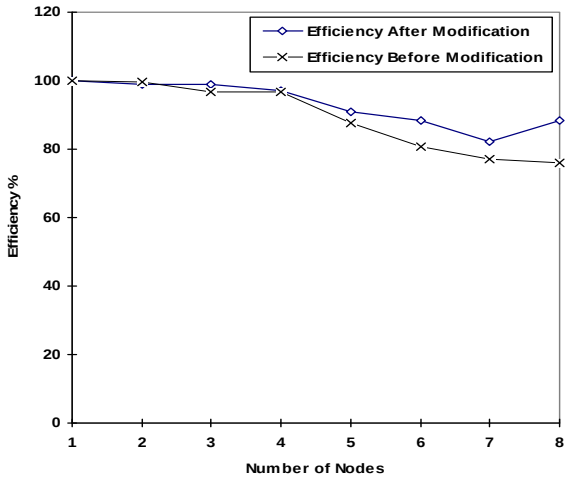

Fig.15 Comparison of Efficiency for standard and improved LVM

In the same way, the efficiency, as in fig 15, show us a noticeable better change. These changes lead us to expect a decreasing in some factor. This expectation leads us also to repeat the cost level of the framework. When we measure the traffic and overheads, we could not recognize noticeable changes in these factors. Except a very small decreasing in traffic as compared with the original one that is caused by the difference in the execution time. The only changes can be noted in the load balancing time, which is said to be an extra cost.

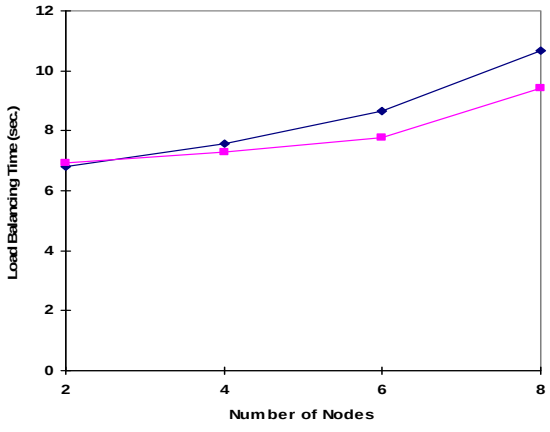

Fig.16 Comparison of load balancing time for standard and improved LVM

Fig.16 illustrates a comparison of load balancing time for standard and improved LVM. As mentioned before, the load balancing efficiency depends on the information dissemination and decision to migrate the processes that is depends on the load vector management. This modified load vector management leads the load exchange more efficiently and leads the load to balance around the cluster quickly.

8. Summary and Conclusions

In the current work, a flexible benchmark framework for examination and experiments of certain type of load balancing single system image is implemented as a methodology of performance evaluation and benchmark. Performance metrics and costs for performance degrade are formulated. Furthermore, the information dissemination algorithm has been modified. Due to this framework, we make an analyzing and empirical study of an existing and successful opensource load balancing SSI. We validate the data of the improved algorithm by comparing with the original system. Finally, we combine results from the tests into single figure of performance behaviour.

The interpretation of the experiments results and the framework reveals to the following points.

The number of nodes affect the performance of SSI cluster and can regarded as an important factor of performance decaying. This number of nodes can affect the performance by adding extra costs in a way for decreasing that performance. Each cost is related to each other in a way that cannot separate them. The main costs that affect the performance are traffic, load balancing time and overhead. The performance of SSI can be enhanced and improved by enhancing any of the above factors or all together.

References

[1] Morin, Christine, et al., "Towards an efficient single system image cluster operating system." Amsterdam, The Netherlands, The Netherlands : Future Gener. Comput. Syst.,Elsevier Science Publishers B. V., 2004, Issue 4, Vol. 20. 0167-739X.

[2] Hwang, Kai, et al., "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space." Los Alamitos, CA, USA : IEEE Journal of Concurrency, 1999, Issue 1, Vol. 7. 1092-3063.

[3] Buyya, Rajkumar, Cortes, Toni and Jin, Hai., "SINGLE SYSTEM IMAGE (SSI)." s.l. : The International Journal of High Performance Computing Applications, 2001, Issue 2, Vol. 15.

[4] OpenSSI home page . [Online] http://openssi.eu/.

[5] Barak, Amnon and La'adan, Oren., "The MOSIX multicomputer operating system for high performance cluster computing." Amsterdam, The Netherlands, The Netherlands : Future Gener. Comput. Syst.,Elsevier Science Publishers B. V., 1998, Issue 4-5, Vol. 13. 0167-739X.

[6] B., Moshe, K., Maya and B., Krushna., "openMosix, a Linux Kernel Extension for Single System Image Clustering." s.l. : Proceedings ofthe 10th International Linux SystemTechnology Conference, 2003.

[7] Ahmed, Bestoun S., et al., "A Descriptive Performance Model of a Load Balancing Single System Image." Los Alamitos, CA, USA : Proceeding of Second Asia International Conference on Simulation and Modelling,IEEE Computer Society, 2008. 978-0-7695-3136-6.

[8] Pacheco, Peter., *Parallel Programming With MPI.* s.l. : Morgan Kaufmann; 1st edition , 1996. 1558603395.

[9] Geist, Al, et al., PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing. s.l. : The MIT Press, 1994. 0262571080.

[10] Keren, Arie and Barak, Amnon., "Opportunity Cost Algorithms for Reduction of I/O and Interprocess Communication Overhead in a Computing Cluster." Piscataway, NJ, USA : IEEE Trans. Parallel Distrib. Syst., 2003, Issue 1, Vol. 14. 1045-9219.

[11] Shivaratri, Niranjan G., Krueger, Phillip and Singhal, Mukesh., "Load Distributing for Locally Distributed Systems." s.l. : Journal of Computer ,IEEE Computer Society, 1992, Issue 12, Vol. 25. 0018-9162.

[12] Wills, Craig E. and Finkel, David., "Scalable approaches to load sharing in the presence of multicasting." s.l. : Journal of Computer Communication, 1995, Issue 9, Vol. 18.

[13] Zhou, S., "A Trace-Driven Simulation Study of Dynamic Load Balancing." s.l. : IEEE Transactions on Software Engineering, 1988, Issue 9, Vol. 14. 0098-5589.

[14] Eager, D. L., Lazowska, E. D. and Zahorjan, J., "The limited performance benefits of migrating active processes for load sharing." s.l. : SIGMETRICS Perform. Eval. Rev., 1988, Issue 1, Vol. 16. 0163-5999.

[15] Harchol-Balter, Mor and Downey, Allen B., "Exploiting process lifetime distributions for dynamic load balancing." s.l. : ACM Trans. Comput. Syst., 1997, Issue 3, Vol. 15. 0734-2071.

[16] Lottiaux, Renaud, et al., "OpenMosix, OpenSSI and Kerrighed: a comparative study." s.l. : IEEE International Symposium on Cluster Computing and the Grid, 2005. 0-7803-9074-1.

[17] Malik, Kamran, et al., "Migratable sockets in cluster computing." New York, NY, USA : Elsevier Science Inc., 2005, Issue 1-2, Vol. 75. 0164-1212.
[18] Barak, Amnon and Braverman, Avner., "Memory ushering in a scalable computing cluster." Melbourne, Vic., Australia : 3rd International Conference on Algorithms and Architectures for Parallel ProcessingICAPP 97, Dec 1997. 0-7803-4229-1.
[19] Barak, Amnon, Guday, Shai and Wheeler, Richard G., *The MOSIX Distributed Operating System: Load Balancing for UNIX*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1993. 0387566635.
[20] Collier, Nigel., Evaluation of Enigma: an OpenMOSIX Cluster for Text Mining. Tokyo,Japan : NII, 2003. 1346-5597.
[21] Damelio, Robert., *The Basics of Benchmarking* . USA : Productivity Press, 1995. 0527763012.
[22] Harbaugh, Sam and Forakis, John A., "Timing studies using a synthetic Whetstone benchmark." New York, NY, USA : ACM,Ada Lett, 1984, Issue 2, Vol. IV. 1094-3641.
[23] Krishnaswamy, Umesh and Scherson, Isaac D., "A Framework for Computer Performance Evaluation Using Benchmark Sets." Washington, DC, USA : IEEE Transactions on Computers,, 2000, Issue 12, Vol. 49. 0018-9340.
[24] Hwang, Kai, et al., "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space." Los Alamitos, CA, USA : IEEE Concurrency, 1999, Issue 1, Vol. 7. 1092-3063.
[25] H., Ying., [Online] http://ying.yingternet.com/mosix.
[26] Vall´ee, Geoffroy, et al., "A New Approach to Configurable Dynamic Scheduling in Clusters Based on Single System Image Technologies." Washington, DC, USA : Proceedings of the 17th International Symposium on Parallel and Distributed Processing,IEEE Computer Society, 2003. 0-7695-1926-1.
[27] Morin, Christine, et al., "Towards an efficient single system image cluster operating system." Amsterdam, The Netherlands, The Netherlands : Future Gener. Comput. Syst.,Elsevier Science Publishers, 2004, Issue 4, Vol. 20. 0167-739X.
[28] Psoroulas, Ioannis, et al., "A Study of the Parameters Concerning Load Balancing Algorithms." s.l. : International Journal of Computer Science and Network Security, 2007, Issue 4, Vol. 7. 1738-7906.
[29] [IPtraf home page. [Online] http://iptraf.seul.org.
[30] Abiona, O. O., et al., "Development of a non Intrusive Network Traffic Monitoring and Analysis System." s.l. : African Journal of Science and Technology, 2006, Issue 2, Vol. 7. 16079949.
[31] Lottiaux, Renaud, et al., "OpenMosix, OpenSSI and Kerrighed: a comparative study." Cardiff, UK : IEEE International Symposium on Cluster Computing and the Grid, 2005. 0-7803-9074-1.
[32] Russo, Ruggero, Lamanna, Davide and Baldoni, Roberto., "Distributed Software Platforms for Rehabilitating Obsolete Hardware." Genova : Proceedings of the First International Conference on Open Source Systems, 2005. 88-7544-048-4.
[33] Hurwitz, Justin (Gus) and Feng, Wu-chun., "End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems." Los Alamitos, CA, USA : IEEE Micro, 2004, Issue 1, Vol. 24. 0272-1732.
[34] Andresen, Robert., "Monitoring Linux with native tools." Las Vegas, Nevada USA : International Conference of The Computer Measurement Group, Inc, 2004.
[35] Tanenbaum, Andrew S and Street, Maarten Van., *Distributed Systems Principles and Paradigm.* s.l. : Pearson Prentice Hall, 2007. 0132392275.

Bestoun S. Ahmed received the B.E. degree in electrical and electronics engineering from University of Salahaddin Erbil in 2004. He is currently a master student and research fellow in Department of Computer and Communication Systems Engineering, University Putra Malaysia. He has been attended in many national and international communication companies. His research interest includes distributed operating system, high performance computing, performance evaluation and modeling, clustering

Khairulmizam Samsudin received the B.E. degree from University Putra Malaysia in 2001. He received the Ph.D. degree in electrical and electronics engineering from University of Glasgow in 2006. He is a faculty member in the Department of Computer and Communication Systems Engineering and leads the Computer Systems Research Group. His research interest includes distributed operating system, high performance computer architecture, biologically inspired computing and mobile-robot agents.

Abd Rahman Ramli received MSc degree in Information Technology System from University of Strathclyde, United Kingdom in 1985 and PhD Degree in Image Processing from University of Bradford, United Kingdom in 1995. He is currently an Associate Professor and Head of Intelligent Systems and Robotics Laboratory in Institute of Advanced Technology Universiti Putra Malaysia. His research interests include image processing and intelligent systems.