Architectural Review of Load Balancing Single System Image

Bestoun S. Ahmed, Khairulmizam Samsudin and Abdul Rahman Ramli
Department of Computer and Communication Systems Engineering,
University Putra Malaysia, 43400 Serdang, Selangor, Malaysia

Abstract: Problem statement: With the growing popularity of clustering application combined with apparent usability, the single system image is in the limelight and actively studied as an alternative solution for computational intensive applications as well as the platform for next evolutionary grid computing era. **Approach:** Existing researches in this field concentrated on various features of Single System Images like file system and memory management. However, an important design consideration for this environment is load allocation and balancing that is usually handled by an automatic process migration daemon. Literature shows that the design concepts and factors that affect the load balancing feature in an SSI system are not clear. **Result:** This study will review some of the most popular architecture and algorithms used in load balancing single system image. Various implementations from the past to present will be presented while focusing on the factors that affect the performance of such system. **Conclusion:** The study showed that although there are some successful open source systems, the wide range of implemented systems investigated that research activity should concentrate on the systems that have already been proposed and proved effectiveness to achieve a high quality load balancing system.

Key words: Single system image, NOWs (network of workstations), load balancing algorithm, distributed systems, openMosix, MOSIX

INTRODUCTION

Cluster of computers has become an efficient platform for computational intensive applications. Nowadays, the usage of clusters is mainly based on batch scheduler and Single System Image (SSI). In the former case, the scheduling of the applications is managed by a supervisor "batch" regarding the available resources in the cluster. Whereas in SSI, the application scheduling is handled transparently by the operating system, to give the appearance of SMP.

Since few years, batch scheduling is preferred because of its simplicity of usage, configuration and implementation. Latest contributions in SSI systems showed the abilities of the system in deferent fields and directions. Among these contributions, the load allocation and balancing which is usually handled by an automatic process migration daemon, performed better especially for reducing the application execution capability.

The single system image architecture was developed to provide a unified system view and globalize processor, file system and network. The characteristics of SSI allow user to access system resources transparently irrespective of where they are available[1].The load balancing single system image clusters dominate research work in this environment.

In this study, we will elaborate briefly the types of SSI clusters and concentrate on load balancing type as the main aim. Such concentration leads to illustrate the load balancing strategies and the architectures of implemented systems. We then stress on two important and successful types of implemented systems from the architecture, design, behaviour and work mechanism as a main points of view. From that view, we will provide new ideas especially how to develop and investigate the weakness of the systems.

We have structured the study in the following way. First we clarify the SSI organizations and structures to justify the types of SSI and how it structured. Then, the load balancing and scheduling mechanism has been declared to know the main components of load balancing in SSI. According to these components, we will declare the developments of varies systems to know the evolution of SSI load balancing systems in addition to the current developments and researches. Finally we will clarify the problems in the implemented systems that have to be solved and stressed on in the

Corresponding Author: Khairulmizam Samsudin, Department of Computer and Communication Systems Engineering, University Putra Malaysia, 43400 Serdang, Selangor, Malaysia, Tel: +60 (3) 89464320

future researches as well as we declared the future directions of SSI systems.

The SSI organizations and structures: In classical cluster systems like Beowulf, a programmer has to write an explicit program by Message Passing Interface (MPI) or by Parallel Virtual Machine (PVM). However, in contrast to high performance Beowulf cluster, Single System Image (SSI) clusters free the end user from such task. According to[2], Single System Image (SSI) is a property of a cluster system to hide the distributed and heterogeneous nature of the resource around the cluster and to present them to applications and users as a single resource. Single system image can be classified in different ways depending on its abstraction layer[3]. The available layers of SSI cluster are:

- Hardware layer
- Middle ware layer
- Application layer
- Operating system layer

In spite of the other types, in the operating system layer, most of the mechanisms are transparent to the user; in other words, the user does not interact with the system and the complexity of its implementation. Therefore, the real benefit of this system is its ease of use; by means, program can use the system resources and availability without modification to the source code. A full SSI can achieve more using OS layer[4] through cooperation among nodes operating system to present same view of the system. However, in practice, it is difficult to combine all characteristics of OS layer together although there is some preliminary work towards such initiative[5]. These characteristics are cluster wide system management, cluster wide device management, cluster file system, cluster wide process management and cluster wide load balancing [4]. To achieve the main purpose of SSI, the load balancing feature becomes most important to reduce execution time and to gain high performance case. Since the main feature of OS layer SSI is the ease of use and transparency, the dynamic load balancing become the main part of implementation.

With dynamic load balancing, the distribution of the workload among the workstation can change at the run time by using current or recent information of the nodes when making the decision [6]. There are two predominant organizations by which dynamic load balancing algorithms are implemented: centralized and decentralized [7]. In centralized structure, a central node plays the major part in the process placement decision of the cluster. Whereas in a decentralized structure, the process placement decision can be made by any of the nodes around the cluster. From the system point of view, each node in the cluster manages the algorithm independently. As a result, any node around the cluster can make decisions.

Though dynamic load balancing policies offer a high degree of adjustment to the fluctuated load, they still suffer from imbalances. That is because when the task is assigned to the execution site by the load balancing algorithm, it will not change through its life.

Pre-emptive load balancing is an improvement of the dynamic policy. The difference is that the decision of the load placing and scheduling is made during run-time continuously. As a result, there is repeated decision of the system scheduling. In this way, a task may begin its execution at its original site and, due to load fluctuation, be reassigned to another site. Such assignment is accomplished by process migration mechanism. It appears from [8] that the benefits of pre-emptive load balancing may cause it to be used extensively in distributed systems.

Scheduling and load balancing mechanisms in SSI: In a cluster of workstations, the main component of the load balancing SSI implementation is the scheduler mechanism. When a given workload is applied on any cluster's node, this given load can be efficiently executed if the available resources are efficiently used. So that, there must be a mechanism for choosing the nodes that have these resources. Scheduling is a component or a mechanism, which is responsible for the selection of a cluster node, to which a particular process will be placed. This mechanism will investigate the load balancing state[9,10]. Hence, scheduling needs algorithms to solve such problems.

In real world, load balancing affected by 3 factors mainly [11]:

- The environment in which one wishes to balance the load.
- The nature of the load itself.
- The load balancing tools available.

The environment defined to include the architecture of the processors that belong to the system, the type of resources that are to be shared among the processors and the form and type of connections between the processors. These factors can be distinguished practically by identifying the nature of the system itself in heterogeneity, resource allocation, or data transfer facilities. In the work load case, tasks in general, tend to be classified as either I/O-bound, CPU-bound, or as mixed tasks. The load balancing policy

decides which site is eligible for the execution of a task and invokes the proper process tool to execute the task at that site.

Load balancing tools represent the main and important part of load balancing systems. It can be represented as the procedures and programs that is responsible for balancing the load. For such reason, two main tools needed: information and process tools. Information tools determine the placement of the process whereas process tools transfer processes between processors in a distributed environment and provide access to the various resources of the distributed system. For the purpose of this study, we will stress more on the load balancing tools or systems.

Any load balancing system must address at least three stages of algorithms in some form[12,13]. These stages described as follows:

Load calculation: Load calculation deals with computer load and it is responsible for calculation of this load. To address a better load calculation, it is required to calculate the load of each single node individually. As a result each node around the cluster must handle this algorithm independently.

Load can be described in different ways, but this algorithm calculates the number of process, which are ready to run but waiting for the CPU and the number of load currently running on CPU. The problem arises when this load is fluctuated rapidly each small period of time. Although there are some trends towards overcome this problem, the most important, successful and popular method until now is by taking the average every period of time. In such case, the load values are accumulated and the averaged every period of time T, where T is a unit of time for load balancing.

For better understanding of how the load calculated, consider a processor with the number of processes ready for execution (W_i) during a time interval (t_{i-1}, t_i), $i = 1,2…,\mu$. Assume that T is an integer product of t. As a result, $T = \mu \times t$. Then, the approximation of the load each time interval T is given by:

$$L_T = \sum_{i=1}^{\mu} W_i$$

This method has been implemented and used successfully in MOSIX[14].

Load information dissemination: When the local load calculated, information collection and dissemination algorithm manages how the load information

communicates to global task schedule. In some implementation [15], single node is responsible for this decision as in Fig. 1, while in other implementations [5], each node in the cluster exchange these information with each other.
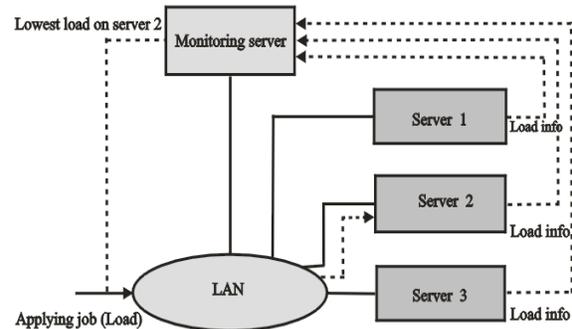

Fig. 1: Example of a load balancing cluster with information collection and dissemination algorithm management

In the cluster of workstations, the load balancing becomes efficient when there is an accurate knowledge about the state of individual node around the cluster. In other words, it is been used to make an optimal balancing and distribution decision of the load. The information collection and dissemination algorithm manages how this load information is distributed to global task scheduler. The purpose of this data collection and information dissemination is to aid the real time decision making by collecting data and reviewing as quickly as possible. Furthermore, to facilitate an efficient and effective decision making and to keep activities on track[16]. For this purpose, broadcast, multicast or probabilistic mechanism is been used for information exchange within decentralized systems.

In broadcast mechanism, each node periodically broadcasts its load information to each node in the cluster. In this way, each node receives and processes a number of messages that is equal to the number of nodes in the cluster. Whereas in multicast mechanism, load information messages are sent to members of certain multicast group to limit the drawback of the traffic in the former one[17]. The probabilistic mechanism tries to minimize the information messages among nodes in the decentralized algorithm by sending messages to a specific number of nodes randomly in the cluster instead of sending messages to all.

Migration consideration: Migration consideration algorithm is responsible for the decision about which

process needs to migrate and where to migrate. It is an important mechanism for load balancing due to its ability to distribute the load.

In a cluster, the process migration represents an important mechanism for balancing the load on the nodes [18]. The mechanism will select candidate a processor for a given process to be executed. Ideally, the chosen processor should have a minimal cost of execution to ensure that the candidate processor will result in an improvement in the load balancing system[19].

To perform such decision, different factors should take into consideration such as response time, process table capacity, amount of free memory, type of processes and migration time[14].

As previously mentioned, the aim of load balancing process migration is to improve the response time of the migrated process. As a result, the migration mechanism should migrate the process to the processor that has fewer loads. This migration happened when the process table inside the operating system is nearly full. In such a case, a process wishes to create a new process named "child" that is almost migrate. In addition to the process table capacity, if there is no enough free memory in the source node for creating new processes, then the algorithm attempt to crate these processes on another processor. However the type of process itself plays a major role. In most implementations, the fork( ) system call create the child processes that can migrate to any other node while there is some tend to migrate thread processes too. However the time and size play a major role for any processes to migrate. That is, if the size or the time of the child processes is small there is no justification for their migration. When these factors considered, the migration time must conceder also. Such time has a direct effect on the migration and its usefulness.

In most of the load balancing single system image implementations, this algorithm is distributed completely. While in some implementations a central node is responsible for this mechanism.

Development of SSI clusters: Researchers has developed and designed many prototypes that each of them differ from each other in the feature and performance and also in use and design. The migration of active entity (process) from one node to another is the main focus of any load balancing system especially in the implementation stage. In most cases, dynamic load balancing mechanism will be implemented and the aim is to gain increased performance from a group of processors or connected nodes.

The implementation and evolution of load balancing SSI starting by developing different systems

with different feature before going to the development of SSI itself. From the literature, two different implementation of the dynamic load balancing and process migration has been proposed: process migration without any modification to the OS itself and process migration by modifying the OS or writing the system from scratch.

One of the earlier research by Freedman described a simple process migration by migrating a process' memory image only. The system runs over UNIX without any modification to the kernel and the program code needs to be modified in order to use the service[20].

Condor is another successful system that provides a process scheduling and migration[21]. The system implemented entirely in the user space to run on top of an operating system without modification on the kernel. The primary aim is to identify the idle nodes in the environment and schedule jobs on them for load balancing. The program code needs to be modified and linked to condor migration library routines during compilation in order to use the service. It was developed in University of Wisconsin-Madison and ported to numerous machines like IBM and Sun[21].

With the development of the systems in user space over the OS, there was a much research concentrating in modifying the OS or writing the system from scratch early. This is to achieve SSI for a unified operating system with high performance feature and for the need to provide ease of use for the normal user. To this end, Accent has developed with a new kernel without compatibility with UNIX with a capability of process migration. In 1990, Amoeba has been developed at Vrije University from scratch with a microkernel later on with UNIX compatibilities but with incomplete compatibility and with the ability of process migration. It was reported to run on Motorola, Intel 80386 and MicroVAX II processors[22,23]. In 1991, Angel was developed in University of London and it is based on microkernel without any compatibility with UNIX. Around 1994, GLUnix started in University of California and it was implemented on top of Solari.

The problem with the systems that started from scratch is that the system does not take any standard form and it is designed for a specific task. The developments of these kinds of systems continued rapidly by designing and developing many other systems like ChorusOS, Guide, Hurricane and Spring. These developments separated the direction of research and development of these systems depending on global resource management. These directions are global memory management and global processor management mainly[5].

In global memory management model, the system consists of several clients and one or more dedicated machines with a communicating channel connecting them. The client machines share the memory resources that are located in a server or dedicated machine. When the local memory of the client machines is exhausted, they move portions of their address space to the dedicated server and regain pieces as needed. This general model was proposed earlier by[24].The work described in[25].goes a step further by designing and implementing a global memory management in a cluster of workstations at the lowest level of the operating system. In addition to the mechanisms that depend on remote memory paging, the Distributed Shared Memory System (DSM) is another way for memory management. In this mechanism, a global shared memory is provided on the top of distributed memory. In this case, the nodes' local memory is used as a local catch of a shared data space [26].

Global processor management allows processor resources of the cluster to be managed in a way that is supervised by load distributing algorithm to allow the process to move from a node to another [5] as in pre-emptive process migration algorithm[14].Several systems were implemented in this direction; one of the first systems was Sprite[27].The Sprite network operating system was developed in University of California in form of a kernel written from scratch. It was designed to run on Sun-2 and Sun-3 workstation. The kernel call interface was very similar to BSD UNIX[29].

Interestingly, some of the earlier load balancing systems used initial placement for supporting the application. It was investigated in the experiments of Concert[29] and other researches that systems, which make intelligent initial process placement, are performed efficiently. This showed that the key to perform better load balancing is to utilize prior information about cluster nodes and processes. This is the feature that the general purpose operating system does not have. MOSIX was one of the systems that provide such a mechanism.

MOSIX[30] is one of the successful systems for SSI used commodity off the shelf computers to gain high performance environment. It consists of a set of additions and modifications to UNIX/Linux kernel. The primary features are the automatic load balancing and the transparent process migration. The pre-emptive process migration represents the main component of the load balancing. Load balancing is achieved continuously during the run time. If the process requirement exceeds the threshold, the process will migrate to another less loaded node. A load vector is kept in each node that contains the information about other random nodes load. This process is done in decentralized organization. The process with the history of forking other processes is better for migration [11].

Although there are many systems have been proposed like UnixWare Non-Stop Cluster [4], Nomad[31] and Plurix[32], none of these systems gives the assurance of every resources global management. Furthermore, the processor global management and load balancing appear to be the classical techniques in this area [33]. On the other hand, the wide range of implemented systems investigated that research activity should concentrate on the systems that have already been proposed and proved effectiveness to achieve a high quality load balancing system.

Nowadays the direction of the implementation and development goes to the systems that implement Linux. There are more than one reason behind this choice. The most important reasons are the Linux feature of free and open source that make the development of such a system more flexible since the source code is available. In contrast, the commercial solutions do not provide the source code. Nowadays OpenSSI[34] Kerrighed[35] and openMosix[36] provide a better load balancing strategy for SSI based on Linux kernel. In addition, these systems are mature enough for use[37].

OpenMosix: OpenMosix is an open source project forked from MOSIX. Most of its designs are similar to that of MOSIX. In 2001, it was decided that future releases of MOSIX would be proprietary for commercial use. OpenMosix becomes a real alternate for such projects for research usage. OpenMosix like MOSIX has a hard limitation of not allowing the migration of shared memory processes. Consequently, applications that use shared memory do not gain benefits from its use. This fact constitutes a significant obstacle to the objectives of research largely based on user applications. In fact, openMosix was designed for HPC, where shared memory process is not necessarily relevant. On July 15, 2007, Bar (openMosix cofounder) announced that the openMosix project will reach its end of life on March 1, 2008. The LinuxPMI [38] project is continuing development of the former openMosix code. Since LinuxPMI is still a prototype and the 2.6 kernel is still under development, we concentrate on openMosix main design components. In the following sections, a brief review is introduced in two directions openMosix architecture and the load balancing mechanism in openMosix.

OpenMosix Architecture: OpenMosix was implemented on a Vanilla Linux kernel. The load balancing mechanism tends to balance the processes on processors around the nodes by migrating extra processes [19]. In this case, a deputation was 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 is 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 [20].Whenever possible a process uses local resources, but often it 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 intercepts all site dependent system calls and forwards them to its deputy from remote node as shown in Fig. 3.

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 a certain threshold point, all user processes are restricted to their home node. When these requirements exceed the CPU threshold levels, some processes will be migrated transparently to other nodes [20].

Memory management is provided in openMosix through a memory ushering algorithm, similar to MOSIX [20].This algorithm will be active when the memory of the node falls below a threshold value. OpenMosix attempts to transfer processes to other nodes, which have sufficient free memory. Thus, process migration is decided not only based on processor load criterion but also by taking into account memory usage [40]. A better understanding of this architecture and mechanism is shown in Fig. 2.

Load Balancing in OpenMosix: As in any SSI system, the major component of openMosix system is the load balancing. The main load balancing components of openMosix are the information dissemination and process migration.

The information dissemination daemon disseminates load balancing information to other nodes in the cluster. The daemon runs on each node and it is responsible for sending and receiving load information to/from other nodes. The sending part of this daemon will periodically send load information each second to two randomly selected nodes.
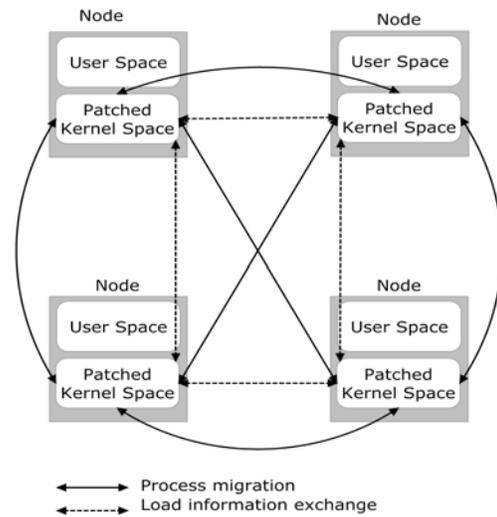

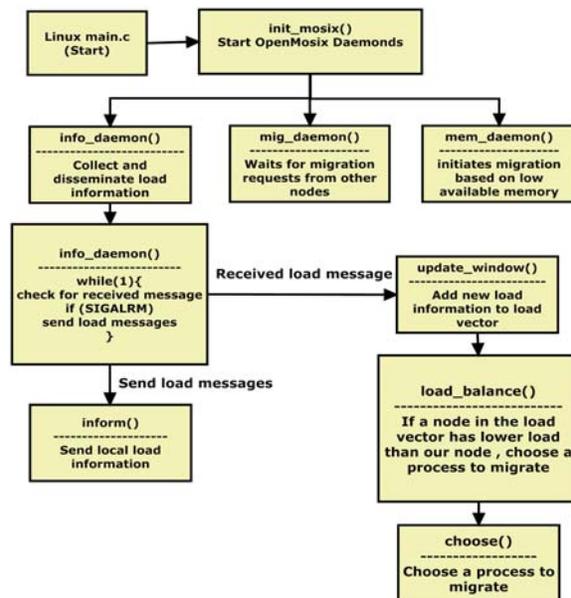Fig. 2: OpenMosix architecture and migration mechanism

Fig. 3: Load information dissemination and collection management

The first node is selected from all nodes that have contacted the node "recently" with their load information. The second node is chosen from other nodes in the cluster [11].The receiving portion of the information dissemination daemon receives the load

information and attempts to replace information in the local load vector[40]. 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[40]. The following flow chart demonstrates the load balancing and information dissemination algorithms briefly.

Kerrighed: Kerrighed is a result of the Gobelins project [41]. It is an SSI system that provides high-level services to high performance applications on clusters of computer in an operating system layer. It is made up from a set of modules that merge with a standard Linux kernel to enable the cluster feature in such a kernel by applying a patch file. In Kerrighed, all nods' resources (processors, memories and disks) are dynamically and globally managed. As mentioned before, the global resources management feature enables resources to be distributed transparently and dynamically throughout the cluster's nods. As a result, better usage of whole cluster resources was carried out[35]. In Kerrighed, the checkpointing is also implemented to avoid the restarting of the application when any node fails. This is where a snapshot of an executing program's state is saved and can be used to restart the running program from the same point at a later time.

When threads and processes are started on any node in Kerrighed system, they can migrate during its execution to any other node or staying in the current node. This migration is based on the scheduling mechanism that is used to migrate the jobs dynamically. For supporting global memory management, Container is used as a new idea to provide a high-level service of a standard operating system to provide shared virtual memory and remote paging[35]. In the global process management perspective, pre-emptive process migration scheme acts as a default scheduler that is responsible for dynamic balancing of CPU load[42]. This scheduler detects the unbalanced and overloads nodes, then migrate the load from higher loaded node to lower loaded nodes [5]. Kerrighed seems very promising research prototype nowadays and for the future.

Open problem in SSI: The information dissemination must take the significance of research and study. The key to the effectiveness of the MOSIX information dissemination algorithm is keeping the number of load messages on the network to a minimum. The information dissemination has to be made efficiently to enable efficient decision making by the scheduler.

There are different aspects of the load balancing algorithms that provide the best opportunity for study by looking at the three general types of load balancing algorithms discussed (load calculation, information dissemination, and migration consideration). A special method can be used and proposed to improve the information dissemination and to address better performance. In openMosix implementation UDP messages are used to disseminate load information and TCP for all other communications such as migrating processes and communicating with the UHN. Therefore, it would be possible to attach node load information to any TCP message and incorporate the extra load information into the load vector. This could potentially increase the accuracy of the load vector information held by each node. As a result, migration decisions could be improved.

Another direction of research appears nowadays with the increase of Multi-core technology. There is an urgent need for a new load balancing and scheduling strategy for such technology in case of using it in SSI clusters. This load balancing strategy should be combined with new and efficient information dissemination that must give good information about the cores and the nodes in the network. However, such information dissemination should represent the Multi-core node load as a one unified representative to disseminate to other nodes. This can be achieved by providing a good scheduling mechanism design.

Future of SSI: Currently, MOSIX and Kerrighed became the base of bigger projects for grid management systems to providing a virtual organization. MOSIX announced MOSIX2 release for 2.6 Linux kernel as a single system image system for clustering and grid management system [43]. Whereas, Kerrighed research group announced XtreemOS [44] as open source Grid operating system. MOSIX2 was extended with a comprehensive set of new features that can manage a cluster and a multi-cluster Grid. The features of MOSIX2 allow better utilization of Grid resources by users who need to run demanding applications but cannot obtain such a large cluster. A production organizational Grid with 15 MOSIX clusters is operational at Hebrew university. XtreemOS is a Grid operating system that will provide native support for Virtual Organizations. Based on Linux, XtreemOS will have 3 different versions capable of running on single PCs, clusters and mobile devices. The cluster flavour of XtreemOS-F relies on the Kerrighed single system image.

CONCLUSION

In this study, we have presented an overview of single system image types, structure and mechanism of work including load balancing and scheduling. Then the

implementations evolution and steps illustrated from past to present. In addition, the new directions of implementation have been declared. We attempted to provide a brief review of implemented technologies as well as the features in each implementation. Depending on the review and from load balancing view point, openMosix and kerrighed represents important and successful opensource implementations till now.

REFERENCES

1. Buyya, R., 1997. A study on HPC systems supporting single system image. Processing in Parallel and Distributed Processing Techniques and Application International Conference, July 3-3, Las Vegas, Nevada, USA., pp: 1106. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.8156.

2. Buyya, R., 1997. Single system image: Need, approaches and supporting HPC systems. Proceedings of the 4th International Conference on Parallel and Distributed Processing, Technique and Applications, June 1997, CSREA Publishers, USA.

3. Buyya, R. T. Cortes, H. Jin, 2001. Single system image. Int. J. High Performance Comput. Appli., 15: 124-135. http://portal.acm.org/citation.cfm?id=1080643.1080652.

4. Walker, B. and D. Steel, 1999. Implementing a full single system image unixware cluster: Middle ware vs under ware. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), June 1999, Monte Carlo Resort, Las Vegas, Nevada, USA., pp: 1-7. http://66.102.1.104/scholar?hl=en&lr=&q=cache:uV9CP3gOsh0J:www.buyya.com/pdpta99/douglass.ps.gz+related:uV9CP3gOsh0J:scholar.google.com/

5. Morin, C., P. Gallard, R. Lottiaux and G. Vall´ee, 2004. Towards an efficient single system image cluster operating system. Future Generation Comput. Sys. J., 20: 505-521. DOI: 10.1016/S0167-739x(03)00170-5.

6. Piotrowski, A. and S. Dandamudi, June 1997. A comparative study of load sharing on networks of workstations. Proceedings of the International Conference on Parallel and Distributed Computing Systems, Oct. 1997, New Orleans, pp: 1-8. http://66.102.1.104/scholar?hl=en&lr=&q=cache:AmfnXnjiM5EJ:www.cs.carleton.ca/research/tech_reports/1997/TR-97-14.ps+.

7. Zaki, M.J., W. Li and S. Parthasarathy, 1997. Customized dynamic load balancing for a network of workstations. J. Parallel Distributed Comput., 43: 156-162. http://www.ingentaconnect.com/content/ap/pc/1997/00000043/00000002/art01339.

8. Harchol-Balter, M. and A.B. Downey, 1997. Exploiting process lifetime distributions for dynamic load balancing. ACM Transact. Comput. Sys. (TOCS)., 15: 253-285. http://DOI.acm.org/10.1145/263326.263344

9. Rao, C.S., M. Naidu, K. Subbaiah and N.R. Reddy, 2007. Process migration in network of linux systems. Int. J. Comput. Sci. Network Security, 7: 213-219. http://paper.ijcsns.org/07_book/html/200705/200705032.html.

10. Du, C., X.-H. Sun and M. Wu, May 2007. Dynamic scheduling with process migration. Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid. May 14-17, IEEE Computer Society, Washington, DC, USA., pp: 92-99. DOI: 10.1109/CCGRID.2007.46

11. Barak, S.G.A. and R. Wheeler, 1993. The MOSIX Distributed Operating System: Load Balancing for UNIX. 1st Edn., Springer-Verlag, Inc., New York, pp: 221. ISBN-10: 0387566635

12. Barak, A. and O. La'adan, 1998. The MOSIX multicomputer operating system for high performance cluster computing. Future Generation Comput. Sys., 13: 361-372. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.6599.

13. Chen, J.C., G.X. Liao, J.S. Hsie and C.H. Liao, 2008. A study of the contribution made by evolutionary learning on dynamic load-balancing problems in distributed computing systems. Int. J. Expert Sys. Appli., 34: 357-365. DOI: 10.1016/j.eswa.2006.09.036

14. Keren, A. and Barak, A., Jan 2003. Opportunity cost algorithms for reduction of I/O and interprocess communication overhead in a computing cluster. IEEE Trans. Parallel Distribut. Syst., 14: 39-50. DOI: 10.1109/TPDS.2003.1167369

15. Zhu, W. and C.F. Steketee, 1995. An experimental study of load balancing on amoeba. Proceedings of the First Aizu International Symposium on Parallel Algorithms/Architecture Synthesis, Mar. 15-17, IEEE Computer Society, Washington, DC, USA., pp:220-226. http://portal.acm.org/citation.cfm?id=527214.826050.

16. Van Albada, G.D., J. Clinckmaillie, A.H.L. Emmen, J. Gehring, O. Heinz, F. van der Linden, B. J. Overeinder, A. Reinefeld and P.M.A. Sloot, 1999. Dynamite - blasting obstacles to parallel cluster computing. Proceedings of the 7th International Conference on High-Performance Computing and Networking, Apr. 12-14, Springer-Verlag, London, UK., pp: 300-310. http://portal.acm.org/citation.cfm?id=645563.6603 42&coll=GUIDE&dl=GUIDE

17. Wills, C. and D. Finkel, 1995. Scalable approaches to load sharing in the presence of multicasting. J. Comput. Communicat., 18: 620-630. DOI: 10.1016/0140-3664(95)99805-M

18. Ho, R.S., C.L. Wang and F.C. Lau, 2008. Lightweight process migration and memory prefetching in openMosix. Proceeding of IEEE International Symposium on Parallel and Distributed Processing, Apr. 14-18, IEEE Computer Society, Rome, Italy, pp: 1-12. DOI: 10.1109/IPDPS.2008.4536329.

19. Malik, K., O. Khan, T. Mobashir and M. Sarwar, 2005. Migratable sockets in cluster computing. J. Sys. Software, 75: 171-177. DOI: 10.1016/J.Jss.2004.03.023

20. Nuttall, M., 1994. A brief survey of systems providing process or object migration facilities. ACM SIGOPS Operat. Sys. Rev., 28: 64-80. http://DOI.acm.org/10.1145/191525.191541.

21. Litzkow, M., M. Livny and M. Mutka, 1988. Condor-a hunter of idle workstations. Proceeding of 8th International Conference on Distributed Computing Systems. June 13-17, IEEE Computer Society, San Jose, CA., USA., pp: 104-111. DOI: 10.1109/DCS.1988.12507.

22. Mullender, S.J., G. van Rossum, A.S. Tanenbaum, R. van Renesse and H. van Staveren, 1990. Amoeba: A distributed operating system for the 1990s. J. Comput., 23: 44-53. DOI: 10.1109/2.53354.

23. Tanenbaum, A.S., R. van Renesse, H. van Staveren, G.J. Sharp and S.J. Mullender, 1990. Experiences with the amoeba distributed operating system. Commun. ACM., 33: 46-63. http://DOI.acm.org/10.1145/96267.96281.

24. Comer, D. and J. Griffioen, June 1990. A new design for distributed systems: The remote memory model. In USENIX Summer Technical Conference. (Citation).

25. Feeley, M.J., W.E. Morgan, E.P. Pighin, A.R. Karlin, H.M. Levy and C.A. Thekkath, 1995. Implementing global memory management in a workstation cluster. Proceedings of the 15th ACM Symposium on Operating Systems Principles, Dec. 03-06, ACM, New York, USA., pp: 201-212. http://doi.acm.org/10.1145/224056.224072

26. Li, K. and P. Hudak, 1989. Memory coherence in shared virtual memory systems. ACM Trans. Comput. Sys., 7: 321-359. http://portal.acm.org/citation.cfm?id=75105

27. Ousterhout, J., A. Cherenson, F. Douglis, M. Nelson and B. Welch, 1988. The sprite network operating system. IEEE J. Comput., 21: 23-36. http://citeseerx.ist.psu.edu/viewdoc/summary?doi= 10.1.1.54.7773.

28. Douglis, F. and J. Ousterhout, 1991. Transparent process migration: Design alternatives and the sprite implementation. J. Software Practice Experience, 21: 757-785. DOI: 10.1002/spe.4380210802

29. Anane, R. and R.J. Anthony, 2003. Implementation of a proactive load sharing scheme. Proceedings of the 2003 ACM symposium on Applied Computing, Mar. 09-12, Melbourne, Florida, pp: 1038-1045. http://doi.acm.org/10.1145/952532.952735.

30. Barak, A. and O. La'adan, 1998. The MOSIX multicomputer operating system for high performance cluster computing. Future Generat. Comput. Syst., 13: 361-372. DOI: 10.1016/S0167-739X(97)00037-X

31. Pinheiro, E. and R. Bianchini, December 1999. Nomad: A scalable operating system for clusters of uni and multiprocessors. Proceedings of the 1st IEEE Computer Society International Workshop on Cluster Computing, Dec. 02-03, IEEE Computer Society, Washington, DC, USA., pp: 247-254. DOI: 10.1109/IWCC.1999.810831

32. Goeckelmann, R., M. Schoettner, S. Frenz and P. Schulthess, 2003. A kernel running in dsm-design aspects of a distributed operating system. Proceedings of IEEE International Conference on Cluster Computing, IEEE Computer Society, Dec. 1-4, Los Alamitos, CA, USA., pp: 478-482. http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5 /8878/28041/01253353.pdf?arnumber=1253353

33. Hajmahmoud, Y., P. Sens and B. Folliot, December 1999. Performance evaluation of a load sharing system on a cluster of workstations. Proceedings of the 6th International Conference on High Performance Computing, Dece. 17-20, Springer-Verlag, London, UK., pp: 71-76. http://portal.acm.org/citation.cfm?id=645445.6531 70.

34. Bestoun, S. Ahmad, Khairulmizam Samsudin and Abdul Rehman Ramli, 2008. Benchmark framwork for a load balacing singla system image. Int. J. Compt. Sci. Network Swcurity, 8: 320-333. http://search.ijcsns.org/02_search/02_search_03.ph p?number=200805048.

35. Morin, C., R. Lottiaux, G. Vallee, P. Gallard, D. Margery, J.Y. Berthou and I.D. Scherson, 2004. Kerrighed and data parallelism: Cluster computing on single system image operating systems. Proceedings of the 2004 IEEE International Conference on Cluster Computing, Sept. 20-23, IEEE Computer Society, Washington, DC, USA., pp: 277-286. DOI: 10.1109/CLUSTR.2004.1392625.

36. Esposito, G.T.R., P. Mastroserio and F. Taurino, 2003. Openmosix approach to build scalable HPC farms with an easy management infrastructure. Proceeding of International Conference of Computing in High Energy and Nuclear Physics, Mar. 24-28, La Jolla, California, pp: 1-2. http://arxiv.org/ftp/hep-ex/papers/0305/0305077.pdf

37. Lottiaux, G.V.R., P. Gallard and C. Morin, 2005. OpenMosix, openssi and kerrighed: A comparative study. Proceeding of IEEE International Symposium on Cluster Computing and the Grid, May 9-12, IEEE Computer Society, Washington, DC, USA., pp: 1016-1023. DOI: 10.1109/CCGRID.2005.1558672

38. LinuxPMI, 2008. http://linuxpmi.org/trac/.

39. Barak, A. and A. Braverman, 1997. Memory ushering in a scalable computing cluster. Proceeding of 3rd International Conference on Algorithms and Architectures for Parallel Processing, Dec. 10-12, IEEE Computer Society, Melbourne, Vic., Australia, pp: 211-224. DOI: 10.1109/ICAPP.1997.651492.

40. Meehan, M. and A. Ritter, 2006. Load balancing experiments in openMosix. Proceeding of International Conference on Computers and Their Applications, March Seattle, Washington, USA, pp: 314-319.

41. Vall´ee, G., C. Morin, R. Lottiaux, J.Y. Berthou and I.D. Malen, 2002. Process migration based on gobelins distributed shared memory. Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, May 21-24, IEEE Computer Society, Washington, DC, USA., pp: 325. http://portal.acm.org/citation.cfm?id=873266.

42. Vall´ee, G., C. Morin, J.Y. Berthou and L. Rilling, 2003. A new approach to configurable dynamic scheduling in clusters based on single system image technologies. Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Apr. 22-26, IEEE Computer Society, Washington, DC, USA., pp: 91. http://portal.acm.org/citation.cfm?id=838553.

43. MOSIX H, 2008. http://www.mosix.org.

44. XtreemOS, 2008. http://www.xtreemos.eu/.