# Comparative Study of Single System Image Clusters

Piotr Osiński [1], Ewa Niewiadomska-Szynkiewicz [1,2]

[1] Warsaw University of Technology, Institute of Control and Computation Engineering, Warsaw, Poland,
e-mail: p.osinski@elka.pw.edu.pl, ens@ia.pw.edu.pl
[2] Research and Academic Computer Network (NASK), Warsaw, Poland.

**Abstract.** Cluster computing has been identified as an important new technology that may be used to solve complex scientific and engineering problems as well as to tackle many projects in commerce and industry. In this paper* we present an overview of three Linux-based SSI cluster systems. We compare their stability, performance and efficiency.

## 1 Introduction to cluster systems

One of the biggest advantages of distributed systems over standalone computers is an ability to share the workload between the nodes. A cluster is a group of cooperating, usually homogeneous computers that serves as one virtual machine [8, 11]. The performance of a given cluster depends on the speed of processors of separate nodes and the efficiency of particular network technology. In advanced computing clusters simple local networks are substituted by complicated network graphs or very fast communication channels.

The most common operating systems used for building clusters are UNIX and Linux. Clusters should effectuate following features: scalability, transparency, reconfigurability, availability, reliability and high performance. There are many software tools for supporting cluster computing. In this paper we focus on three of them: Mosix [9] and its open source version – OpenMosix [12], OpenSSI [14] and Kerrighed [3].

One of the most important features of cluster systems is load balancing. The idea is to implement an efficient load balancing algorithm, which is triggered when loads of nodes are not balanced or local resources are limited. In general, processes are moved from higher to less loaded nodes. Many different load balancing techniques are described in literature [2, 10, 16].

## 2 SSI cluster systems

The idea of SSI (*Single System Image*) [7, 11] is to provide a view of one supercomputer for cluster built from a group of independent workstations. All workstations' resources such as disks, memory, processors are seen by the user as one unique machine. The whole cluster is identified from outside by one IP address. There are three basic features that modern SSI cluster system should implement:

* The paper was partially supported by Polish Ministry of Science and Higher Education grant N N514 416934.

1. Distributed file system usually mounted in one place as root file system. It enables the access to remote disks. A user see cluster hard disks as located on a single machine.
2. Load balancing. SSI clusters are high performance clusters. The idea of moving processes from higher to less loaded computers is implemented to improve performance.
3. Dynamic reconfiguration of a cluster. The system is robust to failure of workstations. This implies the ability of adding and removing nodes while cluster is running.

## 2.1 Mosix system

The Mosix system [1, 9] has been developed at the Hebrew University in Jerosolima by professor Amnon Barak group. In years 1981-1988 the system was known under MOS and NSMOS names. Since appearance of version 4 of the system working under VAX machines on AT&T Unix system it has been known under the present name. In 1998 Linux version was provided. At the end of 2001 Mosix became commercial software, which caused the appearance of the OpenMosix project [7] – the open source version of Mosix. In 2007, OpenMosix leader announced the end of the project. The Mosix team, however, are still working, new version of the Mosix system – Mosix2 was developed.

*System architecture and features*
Mosix was implemented at the kernel level as a loadable module. In this solution the kernel interface is not modified. Mosix is a cache-coherent cluster in which each process shares the execution environment of Unique Home Node (UHN) – a node on which it was initiated. Two resource sharing algorithms are provided: load balancing and memory ushering. The objective of load balancing is to reduce the differences in the load between pairs of cluster nodes. Processes are moved from higher loaded nodes to the less loaded ones. Algorithm is executed on each node and load is balanced independently between pairs of nodes. When memory of a node ran out the memory ushering algorithm is triggered. A given process is moved to a node, which has enough free memory. The remote process maintains interaction with its environment. The context of the process selected to migration is divided into two parts: deputy and remote. Deputy context remains on UHN and cannot be migrated. Remote part of a process is a user context and can be migrated. Hence, all processes that have migrated to other nodes interact with user's environment through the UHN and use the remote node resources when it is possible.

Mosix provides transparent process migration and automatic load balancing in the cluster. Migration of processes using system V semaphores, pipes and sockets is possible. Mosix does not provide full SSI. All the processes, which were launched on a given node are displayed, even if they have been moved to remote nodes. However, processes initiated on other nodes are not displayed. The cluster wide CPU usage and global memory statistics are not displayed. If a process is initiated on a given node the PID identifier for this node is assigned to it. The PID space is not unique cluster wide. Mosix supports hot node removal or addition. The checkpointing mechanism is not implemented.

## 2.2 OpenSSI system

The OpenSSI system [12] appeared in 2001 based on the NonStop Cluster project for Unix Ware – an operating system created by Novell and Unix System Laboratories in 90's. Solutions proposed in LVS (*Linux Virtual Server*) and CI (*Custer Infrastructure for Linux*) have been adopted to OpenSSI.

*System architecture and features*

The OpenSSI architecture can be divided into three parts: outside kernel extensions for high availability and management, kernel extensions and extensions that provide SSI view of a cluster. To enable transparent process migration a special extension to the kernel was introduced. It is called *Vproc* (*Virtual Processes*). The idea lies in adding a virtual layer to the Linux kernel, which is responsible for process management. This layer consists of two lists of structures: *pvproc* structure that points to the *task_struct* – the structure representing the process in the Linux system and *vproc* – structure that contains PID of a process. Node on which process was launched is called *origin node* whereas the node on which the process is currently running is called *local node*. Origin node is responsible for tracing the process state and localization (in case of migration). The local node stores all three process structures: *task_struct*, *vproc* and *pvproc*. In case of migration the origin node stores only virtual structures: *vproc* and *pvproc*. Thanks to the *Vproc* extension there is no need to leave the substitute process on the origin node while migration like in Mosix system.

OpenSSI allows transparent process migration and load balancing within the cluster. It also provides the migration of group of threads. The load balancing algorithm used in OpenSSI is the one derived from the Mosix system. There is possibility to migrate processes using system V memory segments, system V semaphores, pipes and sockets. In OpenSSI all processes running in the cluster are displayed. The devices of all nodes in the cluster are visible through the */dev* directory and can be accessed through every node. The unique space of processes PIDs is preserved. OpenSSI provides hot node adding and removal while the cluster is running. A checkpointing mechanism is not provided.

## 2.3 Kerrighed system

Kerrighed [3, 4] is another open source software for creating efficient SSI computing clusters. The biggest advantage over previously described systems is its speed and efficient process communication and also effective implementation of file system. It supports the SMP machines for building a cluster. The Kerrighed project was started in 1998 at the university IRISA in Paris by Christina Morin group. Since 2006 the project is developed by Kerlabs, INRIA, partners from the Xtreem consortium and many contributors.

*System architecture and features*

The Kerrighed system consists of seven modules described in [4]. Kerrighed implements its own library responsible for high performance communication. It offers user friendly programming interface. To enable transparent process migration the mechanisms, such as process ghosting, containers and migrable streams were implemented. The goal of the process ghosting mechanism is to extract the process state and store it on a given device (disk, network or memory). The container is used to data sharing across cluster nodes. The migrable stream mechanism is used to provide efficient migration of communicating processes.

Kerrighed provides transparent migration of processes as well as single threads. The same as in the OpenSSI system all processes created on any node in the cluster and all processes that migrated are displayed. The unique list of processes PIDs is preserved among the cluster. The biggest disadvantage is impossibility to reconfigure the cluster while it is running – adding and removing nodes has not been implemented. Also the node failure results in failure of a whole cluster. Kerrighed is still in progress, and in many cases the stability is not preserved.

## 3 Comparative study of SSI cluster systems

Many numerical tests were performed to present efficiency, availability and stability of described SSI cluster systems. The goal of the tests was to compare the systems performance in case of different types of applications.

### 3.1 Testing environment

Tests were carried out on three computers connected with 10/100 Mbps Ethernet switch as depicted in Fig. 1. The specification of cluster nodes is presented in Table 1.

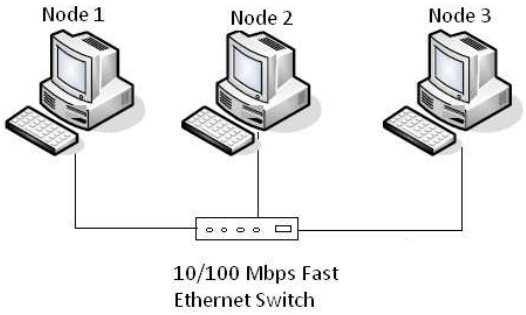


**Figure 1**. Testing environment

**Table 1**: Cluster nodes specification

| Node | CPU info | Memory info | Swap used |
|---|---|---|---|
| 1 | 1728 MHz AMD Duron™, 64 KB cache | 768 MB DDR SDRAM | 1500 MB |
| 2 | 2528 MHz AMD Sempron™, 128 KB cache | 1536 MB DDR SDRAM | 3200 MB |
| 3 | 1528 MHz Intel® Celeron®, 64 KB cache | 1024 MB SDRAM | 2100 MB |

The following versions of systems were considered: OpenMosix 2.4.26 kernel version [13], OpenSSI 2.0 [15] with 2.6.11 kernel version, Kerrighed 2.1.1 [5] with 2.6.20 kernel version. Testing programs were written in C, and compiled with gcc.

### 3.2 Testing examples

Two types of tests were performed for each system:
*Single node performance measurement*. The goal of these tests was to run 1, 2 and 3 instances of calculating program and measure execution time for each series. Measurements show how single node with specific cluster kernel enabled deals with execution of a user code.
*Cluster performance measurement*. These tests were divided into two parts:

- User code execution performance measurement. The objective was to show the performance of each system in case of complex (time consuming) calculations. For each system a specified number of calculating program instances were run. The processes were executed independently without any internode communication. During all tests the calculations were stopped after performing $5 \bullet 10^7$ iterations of the algorithm.

- System code execution performance measurement. The objective was to show the performance of each system when running a program strongly connected to the local resources. Tests were carried out on the cluster consisting of two nodes and were divided into two series. In the first series the program was run on the node 1 and was migrated to the node 3. In the second ones the program was run directly on node 3 (without migration).

For **user code execution** testing purposes a *usercode* program was written. It solves an optimization problem (1) using simple random search (Monte Carlo method).

$$\min_{x}\left[ f(x) = \frac{1}{40}\sum_{i=1}^{n} x_i^2 + 1 - \prod_{i=1}^{n} \cos(\frac{x_i}{i}) \right], \text{ where } D = \{x_i: -20 < x_i < 20\} \quad (1)$$

For **system code execution** testing purposes a *systemcode* program was written. The main feature of this program is long execution time of system code. Program extensively calls system commands.

The following measurements were considered during the experiments:

- *Execution time*: the time needed for the system to execute a task.
- *Load*: the load of each node was measured every second until the end of a test.
- *Bandwidth*: the incoming, outgoing and total bandwidth were measured on each node network interface during execution time of each test.

The goal of these measurements was to compare the efficiency of load balancing algorithms and migration mechanisms implemented in the considered cluster systems. All values presented in figures and tables are average results of five runs of both types of programs.

# 4 Test results

## 4.1 Single node performance evaluation

The performance evaluation of three SSI clusters is presented. In the first set of tests the execution times of calculations performed on a single node were compared. From Fig. 2 we see that the Kerrighed system is about 2 times faster then OpenMosix and OpenSSI. The difference in program execution times can be explained by the kernel versions, Kerrighed offers the efficient solution.

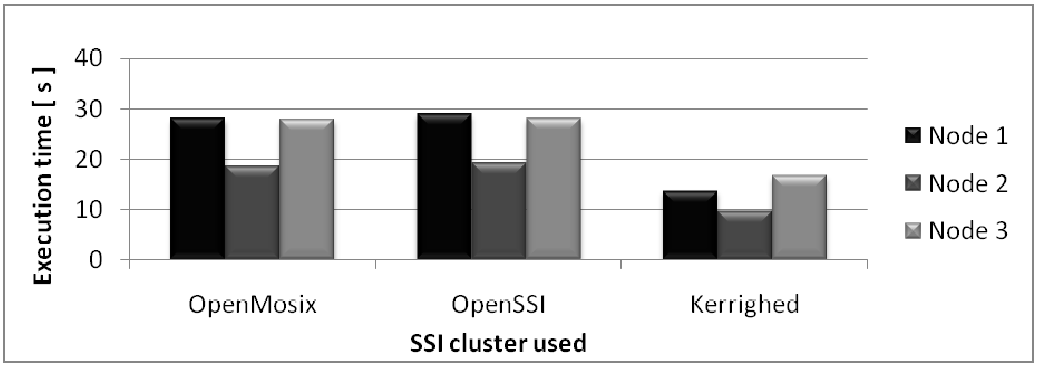


**Figure 2**. Single node performance evaluation

## 4.2 Cluster performance evaluation

*User code execution performance results*

Next we assumed that calculations were done by three machines in the cluster. All processes were launched on the node 1 and were migrated to balance the load. Fig. 3 shows the execution times of seven series of tests with number of processes from 3 to 192 for OpenMosix, OpenSSI and Kerrighed. The best results were obtained for Kerrighed. The execution times for OpenMosix and OpenSSI systems were similar. However in the series with bigger number of processes (Fig. 4) OpenSSI performance dropped significantly. It seems that OpenSSI wasn't able to efficiently distribute such big number of processes over the nodes in the cluster.

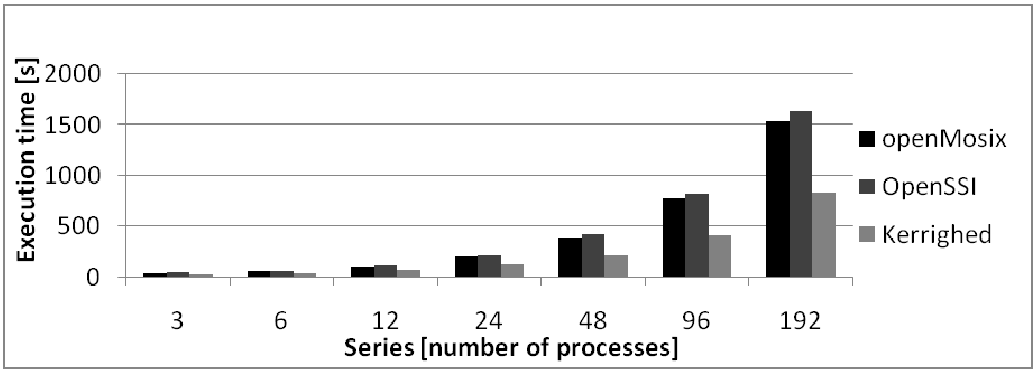


**Figure 3.** Execution time – OpenMosix, OpenSSI and Kerrighed

Tests for Kerrighed and the number of program instances equal 384, 768 and 1536 weren't completed due to the system failure. Kerrighed is not stable system.

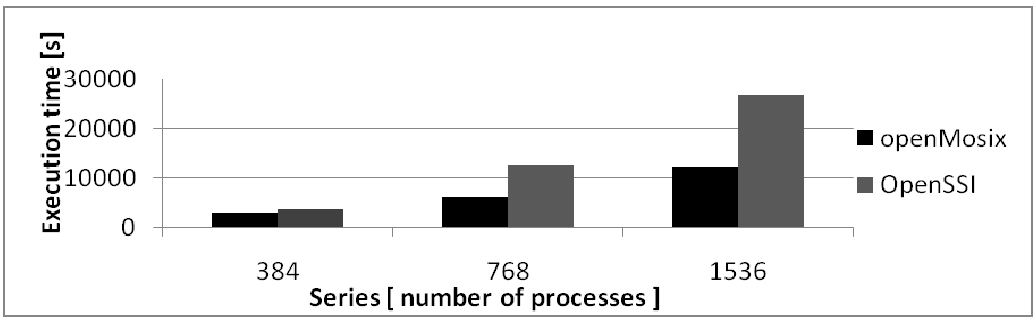


**Figure 4**. Execution time – OpenMosix and OpenSSI.

Next a load of separate node was measured. The goal was to compare the efficiency of load balancing algorithms and migration mechanisms. Figures 5 – 7 present measurements for OpenMosix and OpenSSI, and 768 processes. The architecture of Kerrighed assumes that all system commands provide the information about the whole cluster, so it was impossible to measure load of the separate nodes in this case.

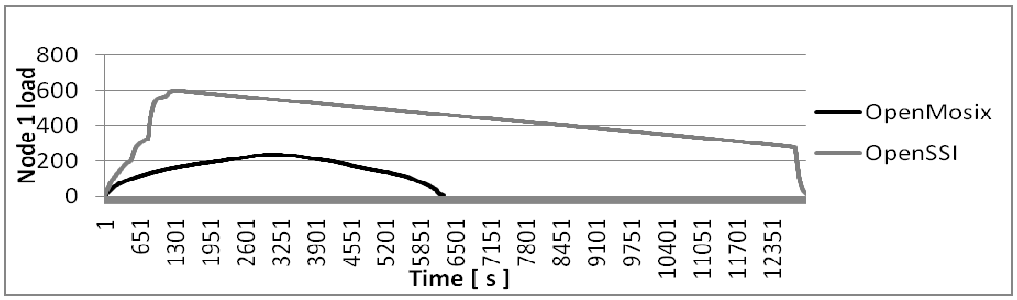


**Figure 5**. Load measurement, node 1

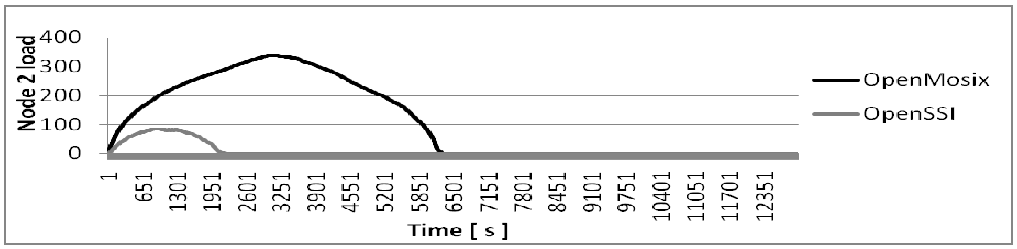


**Figure 6**. Load measurement, node 2

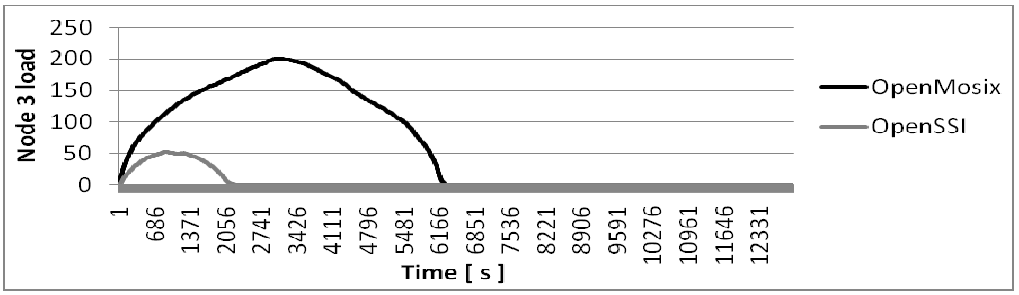


**Figure 7**. Load measurement, node 3

Results presented in Fig. 5 – 7 show clearly that OpenMosix is much efficient when big amount of processes has to be migrated. The load was balanced taking into account the resources and CPU of all computers in the cluster. Node 1 was much more loaded than other nodes in case of OpenSSI. It shows that the implementation of migration mechanism needs some improvements. It collapses in case of big number of processes migration. More detailed measurements of the overhead of process, stream and socket migration in case of the discussed cluster systems can be found in [7].

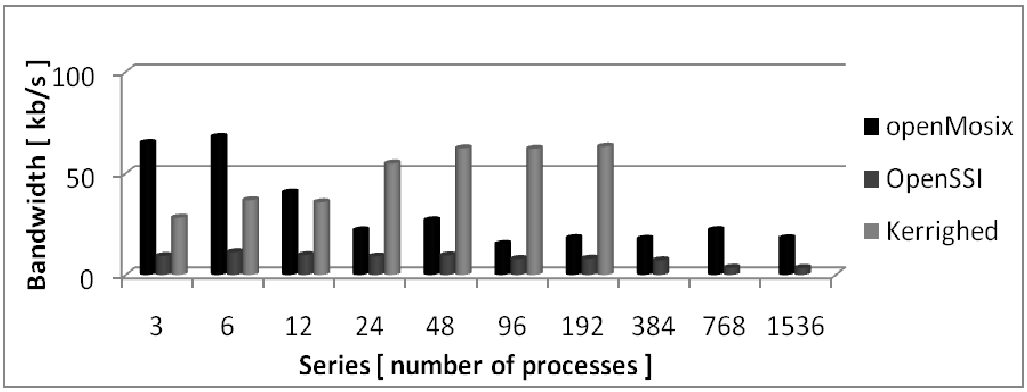


**Figure 8**. Network traffic measured on the node 1 interface

The interprocess communication in the context of process migration was considered. Fig. 8 presents the average traffic loads on the interface of node 1 for OpenMosix, OpenSSI and Kerrighed during each experiment. The tests were performed for number of processes from 3 to 1536. It can be observed that for small number of processes OpenMosix generates the biggest overhead traffic. The situation changes when number of processes increases - Kerrighed starts to generate a big traffic. The results for OpenSSI are similar for all tests.

In summary, OpenMosix was stable during all series of tests. The even distribution of load among the nodes according to their CPU speed was the main factor for increased performance. The bandwidth measurements indicate that when executing big amount of processes the system is focused on computation rather and reduces further migrations. Test of the OpenSSI system showed that for big amount of processes the system is unable to evenly distribute the load between the nodes. Constant and relatively low bandwidth for each series is caused by this defect. The shortest execution times were achieved for Kerrighed system. This comes out from the efficient kernel implementation. Unfortunately Kerrighed was unable to complete more complex tests.

*System code execution performance results*

The objective of this series of experiments was to compare the performance of cluster systems and show the overhead involved by process migration in case of programs with extensive usage of system commands. From Table 2 we see that execution times of migrated *syscode* program are almost the same as without migration in case of Kerrighed and OpenSSI. The performance of OpenMosix is very low when migrating a system-related processes.

**Table 2**: The *syscode* program, execution times in seconds

| | OpenMosix | OpenSSI | Kerrighed |
|---|---|---|---|
| Without migration | 26,23 | 25,26 | 21,57 |
| With migration | 12541,99 | 25,76 | 21,92 |

**Table 3**: Bandwidth measurements – migration of *syscode* program

| | OpenMosix | OpenSSI | Kerrighed |
|---|---|---|---|
| Bandwidth [kB/s] | 1852,77 | 11,59 | 9,18 |

Table 3 presents the bandwidth measurements after *syscode* program migration. All processes were launched on the node 1 and migrated to the node 3. For Kerrighed and OpenSSI the average bandwidth is not very high. The same test for OpenMosix gives very high average bandwidth. It is obvious that the migration of processes using system calls and interprocess communication should be minimized in case of OpenMosix. In case of *syscode* program every system call is associated with data transfer between kernel and user space, hence the communication between two parts of the process: remote - located on the current node, and deputy - located on UHN is very frequent.

The loads of node 1 and 3 measured for *syscode* program execution and the OpenMosix system are presented in Fig. 9. It can be observed that OpenMosix reduces the migration of programs that extensively use the communication to the kernel. It is sensible behavior.

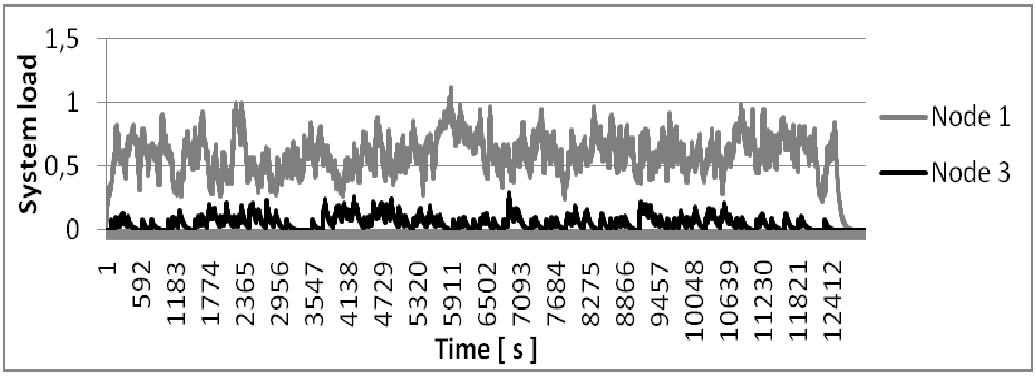


**Figure 9.** The nodes' loads – *syscode* program, the OpenMosix system

## 5. Summary and conclusions

When choosing a cluster software one has to take into consideration four important factors: reliability, simplicity of installation and administration, performance and future development. Results presented in this paper should help to make decision, which of three SSI cluster systems is the best for a given application. All considered systems allow to dynamically balanced load of cluster CPU. Mosix and its open source version OpenMosix systems are most reliable and easy in administration in comparison to their competitors. They are robust to failure of nodes and provide the biggest number of system commands. The installation is simple and user-friendly. The performance of the systems is not bad. Unfortunately, from Tables 2 and 3 we can see that the process migration mechanism implemented in Mosix is not efficient when migrated processes extensively use interprocesses communication or system calls. It results the dramatic extra overheads in communication. Another drawback is that Mosix does not cover all SSI features.

OpenSSI implements nearly all SSI features that a user can expect. In general, it offers very reliable tool for clustering. However in case of multiple processes the performance of OpenSSI is below the one offered by Mosix and Kerrighed. The system is easy to install. Administration is supported by many useful commands, however the system provides less functionality in comparison to Mosix.

The main advantage of Kerrighed in comparison to others is its high performance and efficient process communication. Similarly to OpenSSI it covers nearly all SSI features. The

main disadvantage is the worst reliability. It is not stable software for creating clusters – it is still in early development. Installation of a system and its administration are troublesome.

The paper presents only several features of given systems. We focused on comparison of load balancing and migration mechanisms. To perform more complete set of tests concerning for example systems scalability we need to curry out the experiments on the cluster consisting of many machines.

## Bibliography


[1] A. Barak, O. La'adan, A. Shiloh, Scalable Cluster Computing with Mosix for Linux, *Proceedings of Linux Expo*, pp. 95-100, Raleigh, 1999.

[2] A. Chhabra, G. Singh, S.S. Waraich, B. Sidhu, and G. Kumar, Qualitative Parametric Comparison of Load Balancing Algorithms in Parallel and Distributed Computing Environment, *Proc. of World Academy of Science, Engineering and Technology*, Vol. 16, pp. 58-61, 2006.

[3] Kerrighed project home page: http://www.kerrighed.org/wiki/index.php/Main_Page

[4] Kerrighed reference manual. http://www.kerrighed.org/docs/PI-1576.pdf

[5] Kerrighed 2.1.1 download. http://rpm.pbone.net/index.php3

[6] R. Lottiaux and P. Gallard, Kerrighed Internal, *Technical Report*, Irisa / Inria University, Paris, 2005.

[7] R. Lottiaux, B. Boissinot, P. Gallard, G. Vallee, C. Morin, OpenMosix, OpenSSI and Kerrighed: A Comparative Study, *Proceeding of IEEE International Symposium on Cluster Computing and the Grid* (*CCGrid '05*), Cardiff, UK, 2005.

[8] P. L. McEntire, J. G. O'Reilly, and R. E. Larson, Distributed Computing: Concepts and Implementations, *IEEE Press*, New York, 1984.

[9] Mosix project home page: http://www.mosix.org

[10] R. Motwani and P. Raghavan, Randomized algorithms, *ACM Computing Surveys* (CSUR), 28(1), pp. 33-37, 1996.

[11] E. Niewiadomska-Szynkiewicz, A. Kozakiewicz, A. Karbowski, Distributed Computation in Clusters and Grids, ICCS WUT, Warsaw 2007 (in Polish).

[12] OpenMosix project home page: http://openmosix.sourceforge.net/

[13] OpenMosix 2.4.26 kernel rpm download. http://sourceforge.net/project/showfiles.php?group_id=46729&package_id=137967

[14] OpenSSI project home page: http://openssi.org/cgi-bin/view?page=openssi.html

[15] OpenSSI 2.0 download. http://sourceforge.net/project/showfiles.php?group_id=32541&package_id=154064&release_id=565029

[16] S. Sharma, S. Singh, and M. Sharma, Performance Analysis of Load Balancing Algorithms, *Proc. of World Academy of Science, Eng. and Technology*, Vol. 28, pp. 269-271, 2008.