INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

OpenMosix, OpenSSI and Kerrighed: A Comparative Study

Renaud Lottiaux, Benoit Boissinot, Pascal Gallard, Geoffroy Vallée, Christine Morin

N°5399

November 2004

Systèmes communicants

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5399--FR+ENG

OpenMosix, OpenSSI and Kerrighed: A Comparative Study

Renaud Lottiaux*, Benoit Boissinot†, Pascal Gallard‡, Geoffroy Vallée§, Christine Morin¶

Systèmes communicants
Projets PARIS

Rapport de recherche n° 5399 — November 2004 — 20 pages

Abstract: This paper presents a comparative study of Kerrighed, openMosix and OpenSSI, three Single System Image (SSI) operating systems for clusters. This experimental study gives an overview of SSI features offered by these SSI and evaluates performance of such features.

(Résumé : tsvp)

* IRISA/INRIA {Renaud.Lottiaux@irisa.fr}
† Ecole Normale Supérieur de Lyon {Benoit.Boissinot@ens-lyon.org}
‡ IRISA/INRIA {Pascal.Gallard@irisa.fr}
§ IRISA/INRIA/EDF {Geoffroy.Vallee@irisa.fr}
¶ IRISA/INRIA {Christine.Morin@irisa.fr}

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

OpenMosix, OpenSSI et Kerrighed:
Une étude comparative

Résumé :
Ce rapport de recherche présente une étude comparative de Kerrighed, openMosix et OpenSSI, trois systèmes d'exploitation à image unique pour grappe de PCs. Cette étude experimentale présente un panorama des fonctionnalités offertes par ces systèmes et évalue les performance des fonctionnalités présentes.

1 Introduction

A Single System Image (SSI) is a physical or logical mechanism giving the illusion that a set of distributed resources (memory, disk or CPU) forms a unique and shared resource. At the present time, the term SSI is no more restricted to one given resource but is being more and more extended to all cluster resources: an SSI should offer the view of one unique machine on top of a cluster. SSI can be implemented at several levels : hardware, operating system, middleware or application.

SSI operating systems for clusters are attractive as they ease cluster programming and use. Different groups currently target the development of SSI operating systems, such as openMosix [1] resulting from Mosix [6], OpenSSI [2] derived from previous systems [16], Kerrighed [11], DragonflyBSD [3], Genesis [9] or Plurix [8].

This paper focuses on Linux-based SSI and more precisely on Kerrighed, openMosix and OpenSSI. These systems start to be mature enough to be used outside of research centers. Some of them, such as openMosix are already in use in the industry. However, no real comparative study of these systems has been carried out up to now.

This experimental study aims at giving an overview of SSI features offered by openMosix, OpenSSI and Kerrighed and evaluate performance of such features.

The paper is organized as follows. In a first part, we briefly describe Kerrighed, openMosix and OpenSSI. In a second part, we present features offered by these three systems. In Section 4, we present a performance evaluation of several mechanisms such as process migration, Inter Process Communications (IPC), file accesses and global memory sharing. Finally, we conclude in Section 5.

2 Systems Description

In this section, we briefly describe Kerrighed, openMosix and OpenSSI.

2.1 Overview of Kerrighed

Kerrighed [11] results from a research project started from scratch in 1999. Its aims at providing the view of a single SMP machine on top of a cluster. Kerrighed is made up of a set of kernel distributed services in charge of the global management of cluster resources.

Kerrighed offers a configurable global process scheduler. Using the Kerrighed scheduler builder tool [14], dedicated scheduling policies can be easily written and hot plugged in the cluster. Kerrighed comes with a default scheduling policy which allows to dynamically balance the cluster CPU load by using a receiver initiated preemptive process migration scheme. When a node is under-loaded, the system detects the unbalance and migrates a process from a high-loaded node to an under-loaded node.

The Kerrighed migration mechanism is based on several mechanims, such as process ghosting [13], containers [10], migrable streams [7] and distributed file system.

Process ghosting is used to extract process state information and store corresponding data on a given device. This device can be a disk (process checkpointing), a network (process migration or remote process creation) or a memory (process duplication or memory checkpointing).

The container mechanism is used to share data across nodes while ensuring data coherency. This mechanism is used to implement memory sharing, a cooperative file cache and the Kerrighed distributed file system called KerFS.

The migrable stream mechanism is used to efficiently handle communicating process migration. Processes using pipes or sockets can be migrated with no penalty on latency or bandwidth after migration.

2.2 Overview of openMosix

The openMosix system is based on Mosix [6] a research project started in 1981 at the Hebrew university of Jerusalem.

Mosix uses a sender initiated preemptive process migration scheme to balance the cluster CPU load. The Mosix's migration mechanism is based on a deputy mechanism: when a process is transfered from one node to another, a dependency is preserved on the original node through a deputy process. This deputy allows to solve system calls (network communications, file accesses) of the migrated process that Mosix cannot solve locally after a migration.

Some form of global memory management is provided in Mosix through the memory ushering algorithm [5]. This algorithm is activated when a node's free memory falls below a threshold value and attempts to migrate processes to other nodes which have sufficient free memory. Thus, process migration is decided not only based on processor load criterion but also taking into account memory usage.

Mosix implements its own cluster file system, called Mosix FS (MFS) [4], to allows access to remote disks. To enhance file system performance after migration, a Direct File System Access (DFSA) algorithm is used. This algorithm reduces the need of I/O-bound processes to communicate with their home node.

Mosix was basically developed on a BSD based system. It has been ported on Linux in 1999 and the openMosix *fork* finally appeared in 2002.

2.3 Overview of OpenSSI

OpenSSI appeared in 2001 based on the previous NonStop Cluster for UnixWare project [16], itself based on Locus [15].

The OpenSSI design goal is to provide a platform in which other open source cluster technologies can be integrated. The current version of OpenSSI integrates several open source file systems and disk management systems (GFS, OpenGFS, Lustre, OCFS, DRBD), a distributed lock mechanism (OpenDLM) and a migration mechanism derived from Mosix.

OpenSSI allows to dynamically balance the cluster CPU load by using a process migration scheme derived from Mosix and called load leveling. The OpenSSI migration mechanism uses deputation to maintain IPC and some system calls after process migration and Cluster File System (OCFS) to maintain access to opened files.

3 Comparison of Covered SSI Features

In this section, we present the features offered by the different SSI on a qualitative point of view.

3.1 View of a Unique Machine

In the openMosix system, the non standard *mps* and *mtop* commands shows all processes launched from a given node, even if they have been migrated to remote nodes. However, processes launched from other nodes are not displayed with these commands. The *mtop* command does not display global memory statistics nor cluster wide CPU usage. A unique process identifier (PID) space is offered for every process launched from a given home node. Thus, the PID space is not unique cluster wide.

In the OpenSSI system, the standard *ps* and *top* commands show all processes running in the cluster: processes running locally or remotely, whatever their initial execution node. The *top* command does not display global memory statistics nor cluster wide CPU usage. All cluster devices are listed in the */dev* directory and can be reached from any node. For instance, it is possible to mount a partition from a disk located on a remote node. A unique and cluster wide process identifier (PID) space is present.

In the Kerrighed system, the standard *ps* and *top* commands show all processes running in the cluster: processes running locally or remotely, whatever their initial execution node. The *top* command displays global memory statistics and cluster wide CPU usage just like on a SMP machine. A unique and cluster wide process identifier (PID) space is present.

Table 1 summarizes covered features regarding the vision of a unique machine.

	Kerrighed	openMosix	OpenSSI
Cluster wide unique PID space	Yes	Incomplete	Yes
Cluster wide process view (global ps)	Yes	Incomplete	Yes
Global memory and process load (global top)	Yes	-	-
Global device view (global /dev)	-	-	Yes
Unique file system tree	Yes	Incomplete	Yes

Table 1: Vision of a unique machine features

3.2 Global Process Management

OpenMosix allows to dynamically balance the cluster CPU load by using a sender initiated preemptive process migration scheme. Any process can be migrated except for processes using system V segments or processes strongly connected to a node (direct access to video or network card memory for instance). Threads cannot be migrated. An extension to openMosix called *migshm* enables to migrate processes using system V memory segments. However, this extension does not appear to be really functional.

OpenSSI allows to dynamically balance the cluster CPU load by using a process migration scheme derived from Mosix. Any process can be migrated except for processes strongly connected to a node (direct access to video or network card memory for instance). Processes using system V memory segment and group of threads can be migrated. However, individual threads cannot be migrated.

Kerrighed allows to dynamically balance the cluster CPU load by using the default Kerrighed scheduling policy or with any other policy which can be written and hot-loaded. Any process can be migrated except processes strongly connected to a node. Processes using system V memory segment, individual and group of threads can be migrated.

Table 2 summarizes covered features regarding global process management.

	Kerrighed	openMosix	OpenSSI
Process migration	Yes	Yes	Yes
Individual thread migration	Yes	-	-
Threaded application migration	Yes	-	Yes
Global process scheduler	Yes	Yes	Yes
Customizable global process scheduler	Yes	-	-

Table 2: Global process management features

3.3 Global IPC Management

OpenMosix supports migration of processes using sockets, pipes and system V semaphores. Processes using system V memory segments cannot be migrated.

OpenSSI supports migration of processes using sockets, pipes, system V semaphores and system V memory segments.

Kerrighed supports migration of processes using sockets, pipes and system V memory segments. Processes using system V semaphores cannot be migrated.

Table 3 summarizes covered features regarding global IPC management.

3.4 Fault Tolerance and Checkpointing

It is possible with OpenMosix to add or remove a node to a running cluster. If a node fails, the remaining nodes are not affected. Processes which were running on the faulty node are lost. Processes launched from the faulty node and migrated to remaining nodes before the failure are likely to crash. No checkpointing is provided.

	Kerrighed	openMosix	OpenSSI
Migrate process using System V memory segment	Yes	-	Yes
Migrate process using System V semaphores	-	Yes	Yes
Migrate process using Pipes	Yes	Yes	Yes
Migrate process using UNIX sockets	Yes	Yes	Yes
Migrate process using INET sockets	Yes	Yes	Yes

Table 3: Global IPC management features

It is possible with openSSI to add or remove a node to a running cluster. If a node fails, the remaining nodes are not affected except if the faulty node was hosting the global file system manager. Processes which were running on the faulty node are lost. Processes launched from the faulty node and migrated to remaining nodes before the failure are likely to crash. No checkpointing is provided.

Kerrighed does not yet offers the ability to add or remove a node to a running cluster. If a node fails, the remaining nodes are likely to be affected resulting in a whole cluster dead-lock or crash. Processes which were running on the faulty node are lost. Processes launched from the faulty node and migrated to remaining nodes before the failure are likely to crash. A checkpointing mechanism is offered and allows to restart a process after a node or a cluster crash.

Table 4 summarizes covered features regarding fault tolerance and checkpointing.

	Kerrighed	openMosix	OpenSSI
Hot node addition	-	Yes	Yes
Hot node removal	-	Yes	Yes
Tolerance to node failure	-	Yes	Yes
Process Checkpoint	Yes	-	-
Threaded application Checkpoint	-	-	-
Parallel application Checkpoint	-	-	-

Table 4: Fault tolerance and checkpointing features

3.5 Hardware Support

64 bits processors are supported by openMosix (Itanium) and OpenSSI (Alpha, Itanium). Both systems also support SMP mode on multiprocessor machines. Kerrighed does not support yet 64 bits architectures nor SMP mode. Table 5 summarizes supported hardware architecture for each system.

	Kerrighed	openMosix	OpenSSI
64 bits	-	Yes	Yes
SMP nodes	-	Yes	Yes

Table 5: Supported hardware architectures

4 Performance Evaluation

In this section we present a performance evaluation of Kerrighed, openMosix and OpenSSI. We made several sets of experiments to evaluate the performance of different SSI features. The first set of experiments aims at measuring the overhead of process migration. The second set of experiments aims at measuring the performance of IPC (sockets, pipes, etc) in the context of process migration. The third set of experiments aims at measuring file operations performance. Finally, the last set of experiments aims at measuring the performance of cluster wide shared memory.

4.1 Experimental Platform

The experimental platform consists of four nodes based on one-way Intel Pentium III 1GHz processor with 512 MB physical memory and interconnected with a Fast Ethernet network. We tested Kerrighed version 1.0-rc7 based on Linux 2.4.24, openMosix version 2.4.22-3 based on Linux 2.4.22 and OpenSSI version 1.0.0-rc5 based on Linux 2.4.20.

4.2 Process Migration Evaluation

We measured the process migration overhead by measuring the execution time of a simple sequential application (vector addition) with and without migration. For the migration case, we started the application on an initial node A and migrated it to a destination node B.

The measurement has been done at four different times during the process execution: (1) before initialization of vectors, (2) after initialization of vectors, (3) in the middle of the computation and (4) at the end of the computation. The measurement has been done using the shell command */usr/bin/time*. Figures 1 and 2 show experiment results with vector sizes 16 KB and 64 MB.

With a 16 KB vectors, the computation time is negligible compared to migration time. OpenMosix appears to be much slower than other systems. However, this test does not measure the only migration overhead but also the time needed to send the application termination signal to the *time* command on the initial node. The short migration time and imprecision of the *time* command makes this test difficult to analyze.

With a 64 MB vectors, when the migration occurs before vectors initialization, the overhead is very low and nearly equivalent for the three systems. In this case, the process

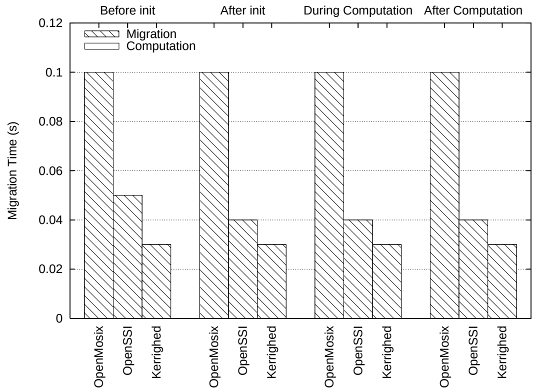

Figure 1: Process migration overhead using a vectors addition with 16 KB data)

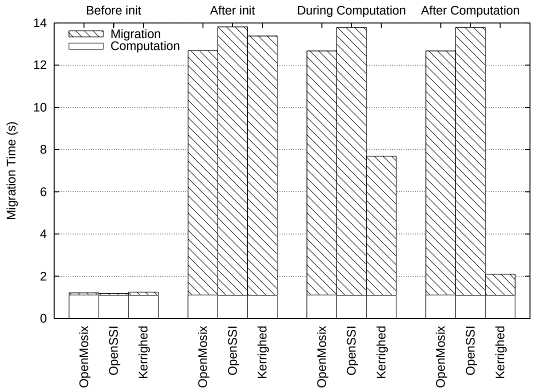

Figure 2: Process migration overhead using a vectors addition with 64 MB data)

memory print mainly consists of code, global variables and stack, which represent less than 1 MB. Migrating this data is not a burden to the migration mechanism.

After vectors initialization, the overhead increases dramatically. Every vector data is needed for the computation on the destination node and has to be transfer through the network. The overheads measured are very close for the three systems. However, openMosix performs a little better than other systems.

During and after the computation, the migration overhead for openMosix and OpenSSI are the same than in the previous case. Both systems migrate every vector data on the destination node, even if they are not required for the computation.

With Kerrighed, the closer to the end the application is when we migrate it, the lower the overhead is. Only needed data is migrated to perform the computation. Computing an element for a vectors addition, does not require the previously computed data. Thus, the more the computation is achieved when we migrate the process, the less we need to transfer data and the lower the overhead is.

4.3 Stream Migration Evaluation

In this section, we present results from Inter Process Communication (IPC) performance evaluation in the context of process migration. Measurements have been carried out using NetPipe [12] for sockets (INET and Unix) and pipes.

4.3.1 Inet Socket

We measured the bandwidth and latency of *inet* sockets in three different cases:

- **No migration**: two processes communicating through a *inet* socket are launched on 2 different nodes. Then, the measurement is performed.

- **One end migration**: two processes communicating through an *inet* socket are launched on 2 different nodes. Then, one process is migrated to a third node and the measurement is performed.
- **Both ends migration**: two processes communicating through a *inet* socket are launched on 2 different nodes. One process is migrated to a third node, the other process on a fourth node. Then the measurement is performed.

Figure 3 presents the bandwidth of *inet* sockets without migration. The bandwidth is nearly the same for the three systems and reaches 90 Mb/s.

Figure 4 presents the bandwidth of *inet* sockets after the migration of one process. For Kerrighed, there is no performance degradation. The bandwidth does not change and still reaches 90 Mb/s. With openMosix, the bandwidth decreases and does not exceed 52 Mb/s. Finally, with OpenSSI, the bandwidth collapses and reaches a maximum at 23 Mb/s. This performance loss with openMosix and OpenSSI can be easily explained by the use of deputation. Since every communication between 2 migrated processes has to go through the deputy located on the home node, an extra network overhead is added to all these communications.

Figure 5 presents the bandwidth of *inet* sockets after the migration of both processes on 2 different nodes. For Kerrighed, the bandwidth still not changes and reaches 90 Mb/s. With openMosix, the bandwidth decreases even more and does not exceed 40 Mb/s. Finally, with OpenSSI, the bandwidth still collapses and reaches a maximum at 21 Mb/s. Here the deputation adds a new extra overhead since 2 indirections (1 for each deputy to reach) are needed for any communication between the 2 migrated processes.

Figure 6 presents the latency of *inet* sockets for each previously described situations and for each SSI. Before migration, the latencies measured for each SSI are close and below 70 μs. OpenMosix shows a latency of 64 μs, OpenSSI shows a latency of 52 μs and Kerrighed shows a latency of 45 μs. After the migration of 1 process the latency increases dramatically for openMosix and OpenSSI, while it remains very low for Kerrighed. This effect is even worth when both processes are migrated.

4.3.2 Unix Socket

We measured the bandwidth and latency of *unix* sockets in four different cases:

- **No migration**: two processes communicating through a *unix* socket are launched on 1 node. Then, the measurement is performed.
- **One end migration**: two processes communicating through a *unix* socket are launched on 1 node. Then, one process is migrated to a second node and the measurement is performed.
- **Both ends migration with different destinations**: two processes communicating through a *unix* socket are launched on 1 node. One process is migrated to a second node, the other process on a third node. Then the measurement is performed.

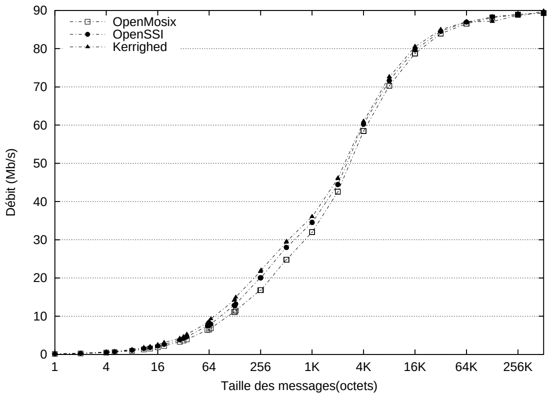

Figure 3: Socket INET bandwidth without migration

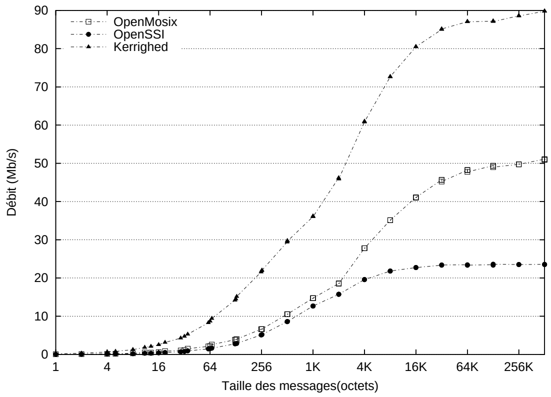

Figure 4: Socket INET bandwidth after 1 end migration

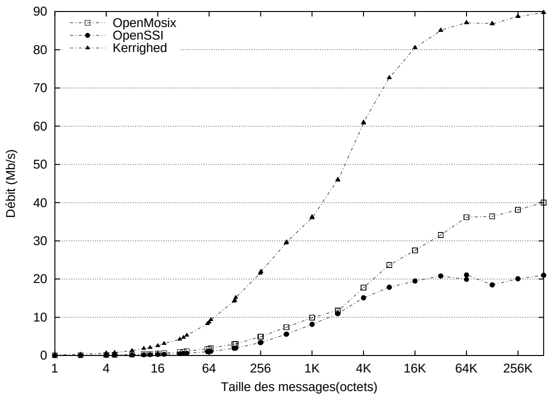

Figure 5: Socket INET bandwidth after both ends migrated on 2 different nodes

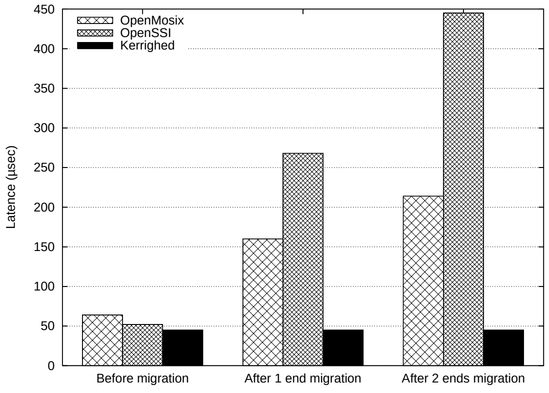

Figure 6: Socket INET latency for each presented scenario

- **Both ends migration with same destination**: two processes communicating through a *unix* socket are launched on 1 node. Both processes are migrated to same destination node. Then the measurement is performed.

Figure 7 presents the bandwidth of *unix* sockets without migration. The bandwidth reaches 7000 Mb/s for Kerrighed, 6000 Mb/s for openSSI and 1500 Mb/s for openMosix.

Figure 8 presents the bandwidth of *unix* sockets after the migration of one process. For Kerrighed, the bandwidth falls to 90 Mb/s. By moving a process to a remote node, the memory bandwidth is replaced by a network bandwidth. With openMosix, the bandwidth falls to 72 Mb/s. Finally, with OpenSSI, the bandwidth falls to 59 Mb/s.

Figure 9 presents the bandwidth of *unix* sockets after the migration of both processes on 2 different nodes. For Kerrighed, the bandwidth does not change and still reaches 90 Mb/s. With openMosix, the bandwidth decreases and does not exceed 56 Mb/s. Finally, with OpenSSI, the bandwidth collapses and reaches a maximum at 33 Mb/s.

Figure 10 presents the bandwidth of *unix* sockets after the migration of both processes on the same destination node. With openMosix and OpenSSI, bandwidth does not change and remains mostly limited by the network bandwidth (56 Mb/s for openMosix and 33 Mb/s for OpenSSI). However, both processes have been migrated on the same destination node, making possible to recover a memory bandwidth. This is what Kerrighed does by offering a bandwidth of 7000 Mb/s.

Figure 11 presents the latency of *unix* sockets for each previously described situations and for each SSI. Before migration, the latencies measured for each SSI are close to 4 μs. After the migration of 1 process, the latency increases dramatically for every systems since we move from a memory latency to a network latency. However, we can observe a significant difference between systems. Kerrighed has a latency of 45 ms, openMosix a latency of 152 μs and OpenSSI has a latency of 264 μs. After migration of both socket ends, the Kerrighed latency does not change, since the latency of other systems still increases. Finally, when both processes are sent to the same destination node, latency for openMosix and OpenSSI does not change and remains very high, since the Kerrighed latency falls back to a memory latency, *i.e.* 4 μs.

4.3.3 Pipe

We measured the bandwidth and latency of pipes in four different cases:

- **No migration**: two processes communicating through a pipe are launched on 1 node. Then, the measurement is performed.
- **One end migration**: two processes communicating through a pipe are launched on 1 node. Then, one process is migrated to a second node and the measurement is performed.
- **Both ends migration with different destinations**: two processes communicating through a pipe are launched on 1 node. One process is migrated to a second node, the other process on a third node. Then the measurement is performed.

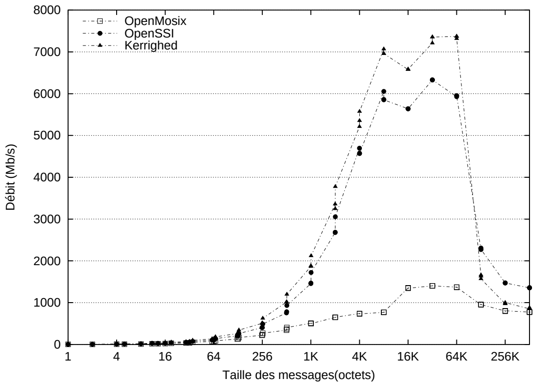

Figure 7: Socket UNIX Bandwidth without migration

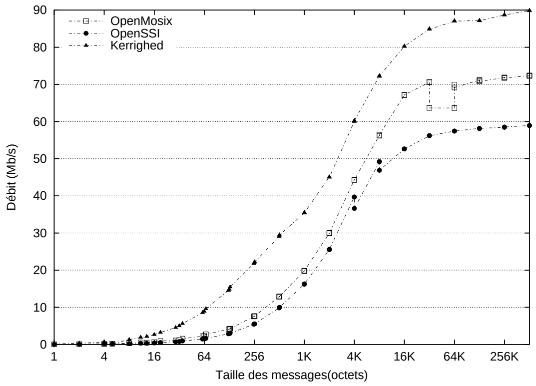

Figure 8: Socket UNIX Bandwidth after migration of 1 end

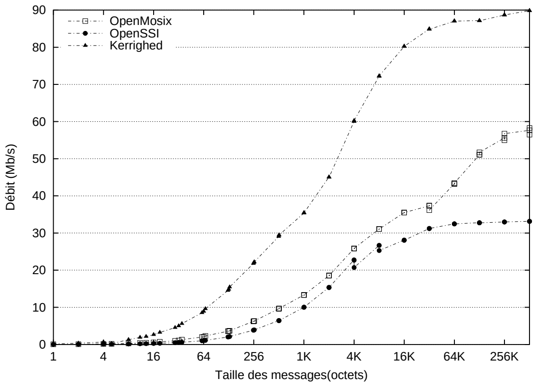

Figure 9: Socket UNIX Bandwidth after both ends migrated on different nodes

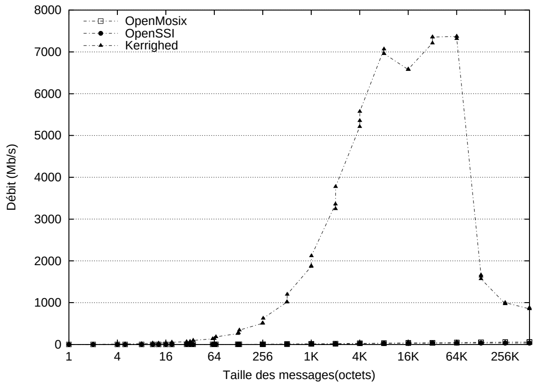

Figure 10: Socket UNIX Bandwidth after both ends migrated on the same node

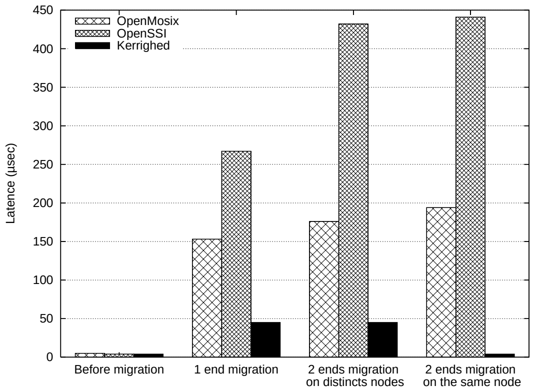

Figure 11: Socket UNIX Latency

- **Both ends migration with same destination**: two processes communicating through a pipe are launched on 1 node. Both processes are migrated to the same destination node. Then the measurement is performed.

Figure 12 presents the bandwidth of pipes without migration. The bandwidth is nearly the same for the three systems and reaches 6500 Mb/s.

Figure 13 presents the bandwidth of pipes after the migration of one process. For Kerrighed, the bandwidth falls to 83 Mb/s. As for *unix* sockets, by moving a process using a pipe to a remote node, the memory bandwidth is replaced by a network bandwidth. With openMosix, the bandwidth falls to 79 Mb/s. Finally, with OpenSSI, the bandwidth falls to 50 Mb/s.

Figure 14 presents the bandwidth of pipes after the migration of both processes on 2 different nodes. For Kerrighed, the bandwidth does not change and still reaches 83 Mb/s. With openMosix, the bandwidth decreases and does not exceed 39 Mb/s. Finally, with OpenSSI, the bandwidth collapses and reaches a maximum at 25 Mb/s.

Figure 15 presents the bandwidth of pipes after the migration of both processes to the same destination node. We observe the same phenomenon as the one with *Unix* sockets: with openMosix and OpenSSI, the bandwidth does not change and remains mainly limited by the network bandwidth while Kerrighed falls back to a memory bandwidth of 7000 Mb/s.

Figure 16 presents the latencies of pipes for each previously described situations and for each SSI. With pipes, the measured latencies are nearly identical to the ones measured with *unix* sockets.

4.4 File System Evaluation

We measured the impact of process migration on file access bandwidth. For each SSI, we used its dedicated file system, *i.e.* MFS with DFSA activated for openMosix, CFS for OpenSSI and KERFS for Kerrighed. The cluster has been rebooted between each test to highlight cache effect: first time file accesses are performed with an empty cache and following file accesses are performed with a heat cache.

4.4.1 Read Access

For each system we did the following tests:

- **Read (cold cache)**: a process P1 started on a node A reads sequentially a 30 MB file after a fresh cluster reboot, to ensure that file caches are empty.

- **Read Again**: the same process P1 still on node A, reads again the same file from the beginning.

- **Read (after migration)**: the process P1 is migrated on a remote node B. Then the same file is again red from the beginning.

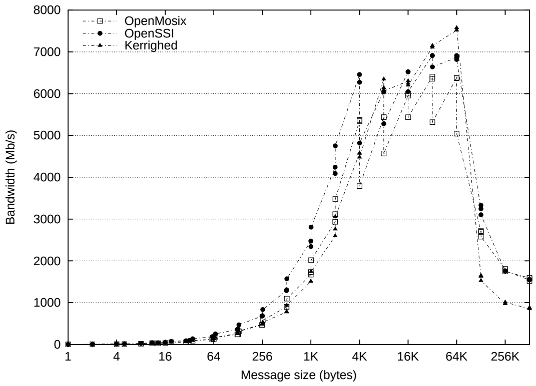

Figure 12: Pipe Bandwidth without process migration

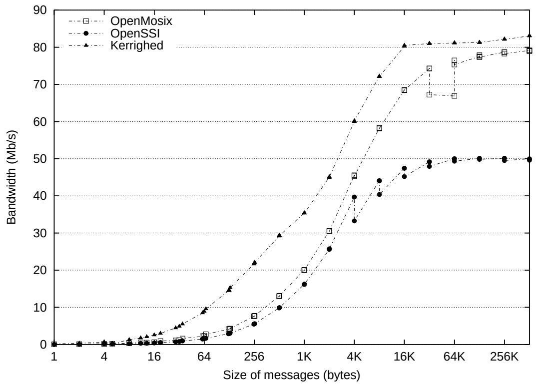

Figure 13: Pipe Bandwidth after 1 end migration

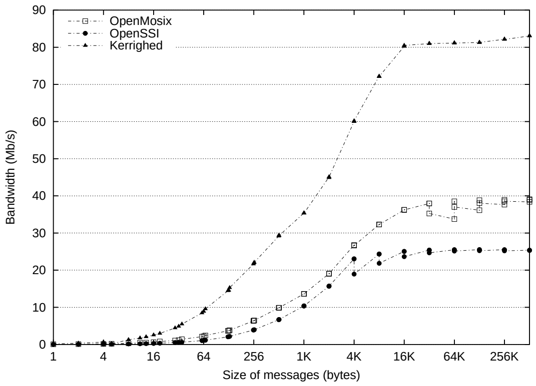

Figure 14: Pipe Bandwidth after both ends migrated on different nodes

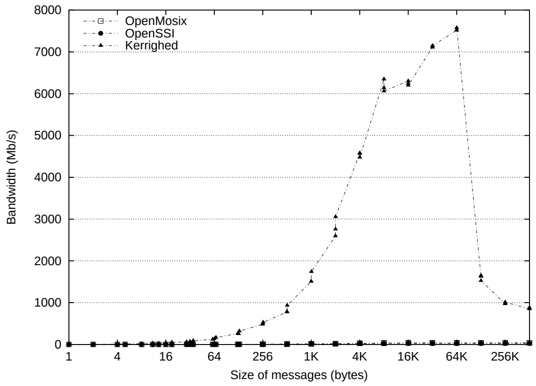

Figure 15: Pipe Bandwidth after both ends migrated on the same node

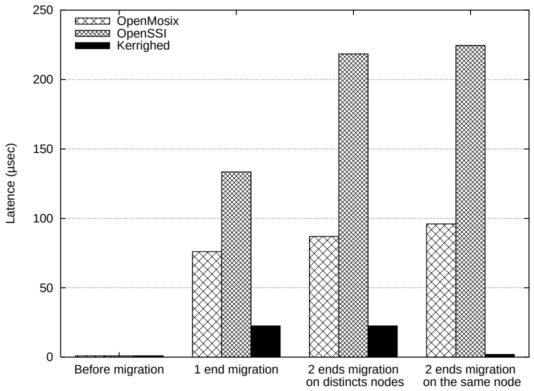

Figure 16: Pipe Latency

- **Read Again (after migration)**: the process P1 still on its destination node B, reads again the file from the beginning.

Figure 17 presents the bandwidth measured for each system and for each previously described situation.

During the first read with an empty cache, openMosix and OpenSSI reach a bandwidth of 25 MB/s, which corresponds to the disk bandwidth. Kerrighed does a counter performance, with a maximum bandwidth of 17 MB/s. We have no explanation at the moment for this poor result with Kerrighed.

During the second read, with a heat cache, all systems exhibit a bandwidth close to 450 MB/s, indicating that the file cache is correctly used.

After migration, for the first read with an empty cache on the destination node, openMosix and OpenSSI reach a bandwidth of 5.5 MB/s, since Kerrighed reaches a bandwidth of 9.6 MB/s. This bandwidth is mainly limited by the network bandwidth. However, the bandwidth measured is far from the theoretical bandwidth of a fast Ethernet network (12 MB/s), especially for openMosix and OpenSSI.

Finally, during the second read on the destination node, with a heat cache, OpenSSI and Kerrighed exhibit a bandwidth close to 450 MB/s, since openMosix is still limited to 5.5 MB/s. This indicates that openMosix does not allow caching on *client* or destination nodes which is surprising if we consider the use of MFS and DFSA.

4.4.2 Write Access

For each system we did the following tests:

- **Write (cold cache)**: a process P1 started on a node A writes sequentially a 30 MB file after a fresh cluster reboot, to ensure that file caches are empty.
- **Read**: the same process P1 still on node A, reads the freshly written file from the beginning.
- **Write (after migration)**: the process P1 is migrated on a remote node B. Then the process overwrites 30 MB on the same file.
- **Read (after migration)**: the process P1 still on its destination node B, reads the file from the beginning.

Figure 18 presents the bandwidth measured for each system and for each previously described situation.

During the first write with an empty cache, Kerrighed reaches a write bandwidth of 229 MB/s, since openMosix and OpenSSI reach respectively a bandwidth of 200 MB/s and 95 MB/s.

During the following read with a heat cache, openMosix and Kerrighed exhibit a read bandwidth close to 450 MB/s, indicating that the file cache is correctly used. For OpenSSI, the bandwidth is limited to 200 MB/s.

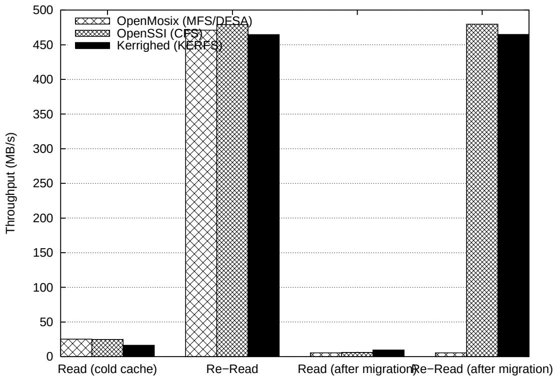

Figure 17: Read throughput before and after process migration

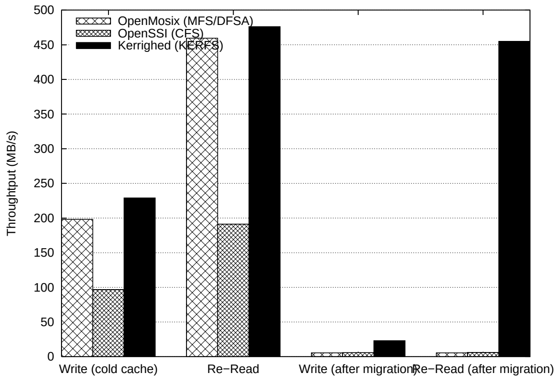

Figure 18: Write throughput before and after process migration

After migration, during a file write with an empty cache on the destination node, openMosix and OpenSSI reach a write bandwidth of 5.7 MB/s. This bandwidth is mainly limited by the network bandwidth. With Kerrighed, the write bandwidth reaches 23 MB/s.

Finally, during the following read on the destination node, with a heat cache, openMosix and OpenSSI exhibit a read bandwidth still limited to 5.7 MB/s since Kerrighed exhibit a bandwidth close to 450 MB/s. This result is not surprising for openMosix since we got the same result for the read access test. This is more surprising for openSSI which has proved in the previous read test that caching was enabled after migration. It seems that writing after a migration kills the cache on an OpenSSI system.

4.5 Shared Memory Evaluation

Performance evaluation of memory sharing for the three given SSI is difficult since only Kerrighed offers a usable cluster wide memory sharing mechanism. OpenMosix does not offer any mechanism to allow memory sharing between nodes. OpenSSI offers some support for cluster wide system V memory segments, however performance observed is dramatically low has shown on figures 19 and 20. These figures present speed-up obtained with simple shared memory parallel applications.

The first application is a Modified Gram-Schmidt (MGS) algorithm used to orthonormalize a vector basis, the second one is a matrix multiply. These applications use system V memory segments to share applications data. We have compared the sequential time to the parallel time using 2, 3 and 4 tasks, each task running on a different node.

With OpenSSI, we have measured very high slow down making this system completely unusable for shared memory programming. With Kerrighed, good speed-ups can be obtained as we can see of figures 19 and 20. However, the speed-up is highly dependent from the application access patterns as usual in software page based DSM systems.

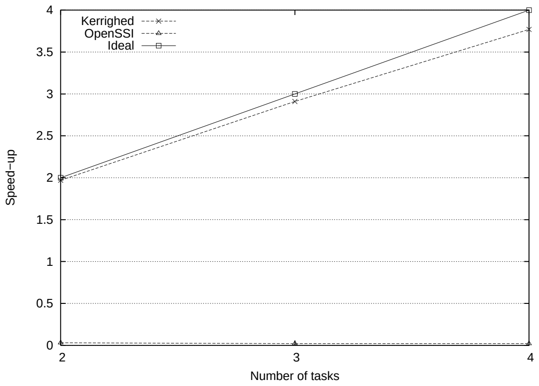

Figure 19: Speed-up with MGS

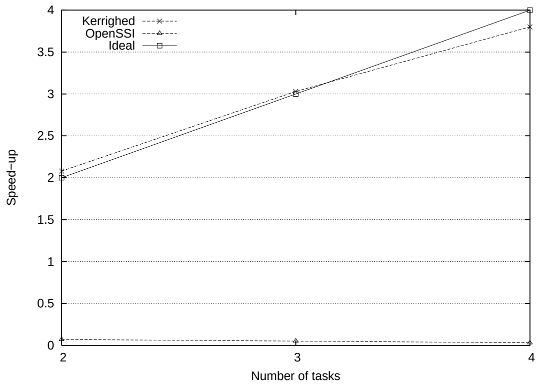

Figure 20: Speed-up with matrix multiply

5 Conclusion

We have presented in this paper a comparison of three single system image cluster operating systems: Kerrighed, openMosix and OpenSSI, based on experimentations conducted on the same cluster.

OpenSSI is the most robust system and covers nearly all SSI features a user could expect. However performance exhibited by this system is often greatly below the one offered by other systems. The deputation mechanism used by openMosix and OpenSSI leads to dramatic extra overheads for IPC after a process migration.

OpenMosix, which is probably the most popular system, offers a good compromise between performance and covered SSI features. However, the openMosix stability is not as good as the one offered by OpenSSI.

Up to now, Kerrighed is still a research prototype, less robust the 2 other systems. Kerrighed does not support hot node addition and removal. Moreover a node crash often leads to a complete cluster crash. However, Kerrighed offers the best performance, specially regarding IPC and file system. Kerrighed is also the only system offering highly customizable features, efficient cluster wide memory sharing, process checkpointing and able to migrate and schedule threads. Finally, Kerrighed still being in development, stability issues are likely to be fixed in a near future.

A more complete set of experiments will be carried out during the next months using more recent machines and networks (gigabit Ethernet, Myrinet). Scalability has not been tested neither. This will be done in our future experiments.

References

[1] http://openmosix.sourceforge.net/.

[2] http://openssi.org/index.shtml.

[3] http://www.dragonflybsd.org.

[4] Loir Amar, Amnon Barak, and Amnon Shiloh. The mosix direct file system access method for supporting scalable cluster file systems. *Cluster Computing*, 7(2):141–150, April 2004.

[5] A. Barak and A. Braverman. Memory ushering in a scalable computing cluster. In *Proc. of IEEE third Int. Conf. on Algorithms and Architecture for Parallel Processing*. IEEE, December 1997.

[6] Amnon Barak, Shai Guday, and Richard G. Wheeler. *The MOSIX Distributed Operating System, Load Balancing for UNIX*, volume 672 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[7] Pascal Gallard and Christine Morin. Dynamic streams for efficient communications between migrating processes in a cluster. *Parallel Processing Letters*, 13(4), December 2003.

[8] Ralph Goeckelmann, Mickael Schoettner, Stefan Frenz, and Peter Schulthess. A kernel running in a dsm - design aspects of a distributed operating system. In *Proceedings of the IEEE International Conference on Cluster Computing*, 2003.

[9] Andrzej Goscinski, Mickael Hobbs, and Jackie Silcock. GENESIS: an efficient, transparent and easy to use cluster operating system. *Parallel Computing*, 28(4):557–606, April 2002.

[10] Renaud Lottiaux and Christine Morin. Containers: A sound basis for a true single system image. In *Proceeding of IEEE International Symposium on Cluster Computing and the Grid (CCGrid '01)*, pages 66–73, May 2001.

[11] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, David Margery, Jean-Yves Berthou, and Isaac Scherson. Kerrighed and data parallelism: Cluster computing on single system image operating systems. In *Proceedings of Cluster 2004*. IEEE, September 2004.

[12] Quinn O. Snell, Armin R. Mikler, and John L. Gustafson. Netpipe: A network protocol independent performance evaluator. In *IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[13] Geoffroy Vallée, Christine Morin, Jean-Yves Berthou, Ivan Dutka Malen, and Renaud Lottiaux. Process migration based on gobelins distributed shared memory. In *Proceedings of the workshop on Distributed Shared Memory (DSM'02) in CCGRID 2002*, pages 325–330. IEEE Computer Society, May 2002.

[14] Geoffroy Vallée, Christine Morin, Jean-Yves Berthou, and Louis Rilling. A new approach to configurable dynamic scheduling in clusters based on single system image technologies. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*. IEEE, April 2003.

[15] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The locus distributed operating system. In *Proceedings of the ninth ACM symposium on Operating systems principles*, pages 49–70. ACM Press, 1983.

[16] Bruce Walker and Douglas Steel. Implementing a full single system image unixware cluster: Middleware vs underware. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'99*, 1999.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399