The MOSIX2 Management System for Linux Clusters and Multi-Cluster Organizational Grids

A White Paper

A. Barak and A. Shiloh
http://www.MOSIX.org

January 2009

Overview

MOSIX[1] Version 2 (MOSIX2) for Linux-2.6 is an operating system like management system targeted for high performance computing on x86 based (32-bit and 64-bit) Linux clusters and multi-cluster (organizational) Grids [5]. MOSIX supports both interactive processes and batch jobs. It incorporates dynamic resource discovery and automatic workload distribution, commonly found on single computers with multiple processors. In a MOSIX system, users can run applications by creating multiple processes, then let MOSIX seek resources and automatically migrate processes among nodes to improve the overall performance, without changing the run-time environment of migrated processes.

MOSIX is implemented as a software layer that provides applications with an unmodified Linux run-time environment. Therefore, there is no need to change or even link applications with any special library. Moreover, MOSIX supports most additional Linux features that are relevant to ordinary, non-threaded Linux applications, so that most Linux programs can run unchanged.

MOSIX Version 1 was originally developed to manage a single cluster. MOSIX2 was extended with a comprehensive set of new features for managing a cluster and a multi-cluster Grid, e.g., in different departments of an organization, as well as shared servers and independent workstations [1]. For example, a new feature allows owners of clusters to share their computational resources from time to time, while still preserving the autonomy of the owners to disconnect their clusters from the Grid at any time, without sacrificing guest processes from other clusters.

In a MOSIX configuration, nodes can be partitioned into logical (virtual) clusters that are allocated to specific user(s) or purpose(s). Logical clusters do not necessarily correspond to physical clusters.

Resources are managed by the automatic resource discovery and the process migration algorithms. The resource discovery algorithm provides each node with the latest information about resource availability and the state of the nodes. Based on this information and subject to priorities, the process migration algorithms can initiate reallocation of processes among nodes, such as to improve the performance, e.g., by load-balancing, or to move processes from a disconnecting cluster.

A priority method ensures that local processes and processes with a higher priority can always move in and force out guest (migrated) processes with a lower priority. The priority method can be used to guarantee fair access to users. It can also be used to support flexible configurations in which clusters can be shared (symmetrically or asymmetrically) among users of different clusters. By proper setting of priorities, processes from a higher priority cluster can move to other clusters and if necessary, even to all the Grid nodes. Users need not know the details of the configuration nor the state of any resource.

Other features of MOSIX include migratable sockets by direct communication between migrated processes; a secure run-time environment

[1] MOSIX® is a registered trademark of A. Barak and A. Shiloh.

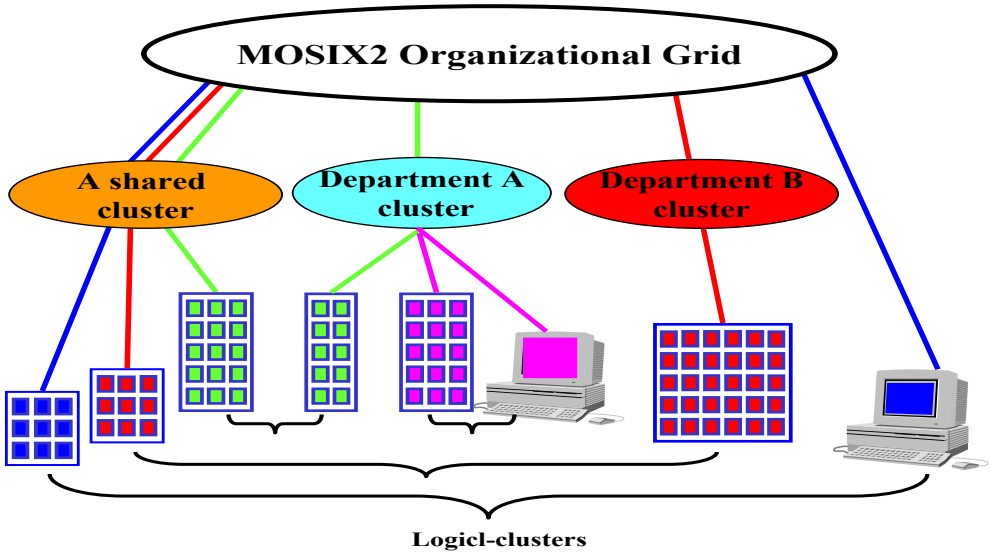

Figure 1: A multi-cluster Grid with 3 clusters and 2 workstations, forming 4 logical clusters.

(sandbox) that prevents guest processes from accessing local resources in hosting nodes; "live-queuing" that preserves the full generic Linux environment of queued jobs; gradual release of queued jobs, to prevent flooding of any cluster; checkpoint and recovery and support of batch jobs.

Due to networking and management overheads, MOSIX is particularly suited to run compute intensive and other applications with low to moderate amounts of I/O. Tests of MOSIX show that the performance of several such applications over a 1Gb/s campus backbone is nearly identical to that within the same cluster [1].

A key requirement for safe Grid computing is trust, i.e., a guarantee that applications are not viewed or tempered when running in remote clusters. Other safety requirements include a secure network and that the Grid includes only authorized nodes with identifiable IPs. Since nowadays these requirements are standard within clusters and intra-organizational Grids, but usually not elsewhere, we recommend using MOSIX in such configurations. Other than the above requirements, MOSIX could be used in any Grid.

Figure 1 illustrates the flexibility of partitions of nodes in a multi-cluster Grid with two departmental clusters and a shared cluster. In this Grid, the green (red) logical cluster consist of sets of nodes from Department A (B) and the shared cluster. Another logical cluster is formed by the (blue) workstation and the Blue nodes in the shared cluster. In the above configuration, when the shared cluster is disconnected from the Grid, all guest processes are either moved out to other clusters, or to their respective home-clusters or to the workstation, where they continue to run or are frozen until other nodes become available.

The ability to form logical clusters provides greater flexibility in the allocation of nodes to users. For example, nodes from the shared cluster can, without being stopped, be disconnected from one logical cluster and be moved to another, without losing running processes - this feature can be useful for groups that need to partition a large cluster among different users. Further flexibility can be obtained by proper setting of the priorities. For example, if the processes of the user in the red cluster are given a higher priority, then this user can get all the nodes in the shared cluster, and if necessary, even all the Grid nodes.

A production campus Grid with 15 MOSIX clusters ($\sim$ 650 CPUs) in different departments is operational at our university (see http://www.MOSIX.org/webmon). The features of MOSIX2 allow better utilization of Grid resources by users who need to run demanding applications but can not afford such a large cluster.

1 Main Features of MOSIX2

This section describes the main features of MOSIX2.

1.1 Logical Clusters

A cluster or a multi-cluster (Grid) can be partitioned into several logical clusters, each allocated to a user, a group of users or a general pool for shared use. A logical cluster may span nodes from the same cluster or from different physical clusters. Normally, each user is expected to login and create processes only in one logical cluster, called the home-cluster. The section on the priority method explains how partitions to logical clusters can provide fair-share access among different users.

Allocation of nodes to logical clusters is done by administrators, who can modify the allocations from time to time to reflect changing demands. Below, all references to clusters mean logical clusters.

1.2 Automatic Resource Discovery

Resource discovery is performed by an on-line information dissemination algorithm, providing each node in the cluster (Grid) with the latest information about availability and the state of system-wide resources. The algorithm is based on a randomized gossip dissemination, in which each node regularly monitors the state of its resources, including the CPU speed, current load, free and used memory, etc. This information, along with similar information that has been recently received by that node is routinely sent to a randomly chosen node, where a higher probability is given to choosing target nodes in the local cluster.

Information about newly available resources, e.g., nodes that have just joined, is gradually disseminated across the active nodes, while information about disconnected nodes is quickly phased out. In [6] we presented bounds for the age properties and the rates of propagation of the above information dissemination algorithm.

1.3 Types of Processes

MOSIX2 recognizes two types of processes: Linux processes and MOSIX processes. Linux processes are not affected by MOSIX, they run in native Linux mode and can not be migrated. MOSIX processes can be migrated.

Linux processes usually include administrative tasks and processes that are not suitable for migration. Another class of Linux processes is those created by the "mosrun -E" command. These processes can not be migrated, but can be assigned to the least loaded nodes in the cluster by the "mosrun -b" option.

MOSIX processes are usually user applications that are suitable and can benefit from migration. All MOSIX processes are created by the "mosrun" command. MOSIX processes are started from standard Linux executables, but run in an environment that allows each process to migrate from one node to another. Each MOSIX process has a unique home-node, which is usually the node in which the process was created [3]. Child processes of MOSIX processes remain under the MOSIX discipline (with the exception of the **native** utility, which allows programs, mainly shells, already running under mosrun, to spawn children in native Linux mode). Below, all references to processes mean MOSIX processes.

1.4 Process Migration

MOSIX2 supports cluster and Grid-wide (preemptive) process migration [3]. Process migration can be done either automatically or manually. The migration itself amounts to copying the memory image of the process and setting its run-time environment. To reduce network occupancy, the memory image is often compressed using LZOP [4].

Automatic migrations are supervised by on-line algorithms that continuously attempt to improve the performance, e.g., by load-balancing; by migrating processes that requested more than available free memory (assuming that there is another node with sufficient free memory); or by migrating processes from slower to faster nodes. These algorithms are particularly useful for applications with unpredictable or changing resource requirements and when several users run simultaneously.

Automatic migration decisions are based on (run-time) process profiling and the latest information on availability of Grid resources, as provided by the information dissemination algorithm. Process profiling is performed by continuously collecting information about its characteristics, e.g.,

size, rates of system-calls, volume of IPC and I/O. This information is then used by competitive online algorithms [2] to determine the best location for the process. These algorithms take into account the respective speed and current load of the nodes, the size of the migrated process vs. the free memory available in different nodes, and the characteristics of the processes. This way, when the profile of a process changes or when new resources become available, the algorithm automatically responds by considering reassignment of processes to better locations.

1.5 The Run-Time Environment

MOSIX is implemented as a software layer that allows applications to run in remote nodes, away from its home-node, as if they run locally. This is accomplished by intercepting all system-calls, then if the process was migrated, the majority of its system-calls are forwarded to its home-node, where they are performed on behalf of the process as if it was running in the home-node, then the results are sent back to the process.

In MOSIX, applications run in an environment where even migrated process seem to be running in their home-node. As a result, users do not need to know where their programs run, they need not modify applications, link applications with any library, login or copy files to remote nodes. Furthermore, file and data consistency, as well as most normal IPC mechanisms such as signals, semaphores and process-ID's are intact.

The outcome is a run-time environment where each user gets the impression of running on a single computer. The drawback of this approach is increased overheads, including management of migrated processes and networking.

1.5.1 Overhead of Migrated Processes

To illustrate the overhead of running migrated processes we ran four real-life applications, each with a different amount of I/O. The first application, **RC**, is an intensive CPU (satisfiability) program. The second application, **SW** (proteins sequences), uses a small amount of I/O. The third program, **JEL**lium (molecular dynamics), uses a larger amount of I/O. Finally, **BLAT** (bioinformatics) uses a moderate amount of I/O.

We ran each program in three different ways:

1. As a local **Linux** process.
2. As a migrated MOSIX process to another node in the same **cluster**.
3. As a migrated MOSIX process to a cluster located about 1 Km away, on a campus **Grid**.

In all cases we used identical Xeon 3.06GHz servers that were connected by a 1Gb/s Ethernet.

Table 1: Local vs. Remote Run-times (Sec.)

	RC	**SW**	**JEL**	**BLAT**
Linux times	**723.4**	**627.9**	**601.2**	**611.6**
Total I/O	0MB	90MB	206MB	476MB
Block size	–	32KB	32KB	64KB
#-Syscalls	3,050	16,700	7,200	7,800
Cluster times	**725.7**	**637.1**	**608.2**	**620.1**
Slowdown	0.32%	1.47%	1.16%	1.39%
Grid times	**727.0**	**639.5**	**608.3**	**621.8**
Slowdown	0.50%	1.85%	1.18%	1.67%

The results (averaged over 5 runs) are shown in Table 1. The first four rows show the **Linux** run-times (Sec.), the total amounts of **I/O** (MB), the I/O **block size** (KB) and the number of **system-calls** performed. The next two rows list the run-times of migrated MOSIX processes and the slowdowns (vs. the Linux times) in the **same cluster**. The last two rows list the run-times and the slowdowns across the **campus Grid**.

Table 1 shows that with a 1Gb/s Ethernet, the average slowdown (vs. the Linux times) of all the tested programs was 1.085% in the same cluster, and 1.3% across a campus Grid, an increase of only 0.215%. These results confirm the claim that MOSIX is suitable to run compute bound and applications with moderate amounts of I/O over fast networks.

1.5.2 Migratable Sockets

Migratable sockets allows processes to exchange messages by direct communication, bypassing their respective home-nodes.

For example, if process X whose home-node is A and runs on node B wishes to send a message over a socket to process Y whose home-node is C and

runs on node D, then without a migratable socket, the message has to pass over the network from B to A to C to D. Using direct communication, the message will pass directly from B to D. Moreover, if X and Y run on the same node, then the network will not be used at all.

To facilitate migratable sockets, each MOSIX process can own a "mailbox". MOSIX Processes can send messages to mailboxes of other processes anywhere in the Grid (that are willing to accept them).

Migratable sockets makes the location of processes transparent, so the senders do not need to know where the receivers run, but only to identify them by their home-node and process-ID (PID) in their home-node.

Migratable sockets guarantees that the order of messages per receiver is preserved, even when the sender(s) and receiver migrate several times.

1.5.3 A Secure Run-Time Environment

The MOSIX software layer guarantees that a migrated (guest) process can not modify or even access local resources other than CPU and memory in a remote (hosting) node. Due care is taken to ensure that those few system-calls that are performed locally, can not access resources in the hosting node. The majority are forwarded to the home-node of the process. The net result is a secure run-time environment (sandbox), protecting the host from stray guest processes.

1.6 The Priority Method

The priority method ensures that local processes and processes with a higher priority can always move in and push out all processes with a lower priority. The priority method allows flexible use of nodes within and among groups. By default, guest processes are automatically moved out whenever processes of the cluster's owner or other more privileged processes are moved in.

Owners of clusters can determine from which other cluster they are willing to accept processes and which clusters to block. Processes from unrecognized clusters are not allowed to move in. Note that the priority applies to the home-node of each process rather than to where it happens to arrive from.

By proper setting of the priority, two or more clusters could be shared (symmetrically or asymmetrically) among users of each cluster; a cluster can be shared among users or owners (with the same priority processes) from other clusters, or migration from one cluster could be blocked to other clusters.

Within a cluster, the priority method can be used to guarantee fair-share access to users, e.g., when some users run many jobs, thus depriving other users from their share. The priority method can also be helpful when some users run long jobs while other users need to run (from time to time) short jobs. In both of these scenarios, the sysadmin can partition the cluster to several logical sub-clusters, then allow each user to login to only one sub-cluster. As explained above, processes of local users (in each sub-cluster) has higher priority over guest processes from other sub-clusters. Note that users of each sub-cluster can still benefit from idle nodes in the other sub-clusters.

1.7 Flood Control

Flooding can occur when a user creates a large number of processes, either unintentionally or with the hope that somehow the system will run it. Flooding can also occur when a large number of processes migrate back to their respective home-clusters, when other clusters are disconnected or are reclaimed.

MOSIX has several built-in features to prevent flooding. For example, the load-balancing algorithm does not permit migration of a process to a node with insufficient free memory. Another example is the ability to limit the number of guest processes per node.

To prevent flooding by a large number of processes, including returning processes, each node can set a limit on the number of local processes of certain classes. When this limit is reached, additional processes of those classes are automatically frozen and their memory images are stored in regular files. This method ensures that a large number of processes can be handled without exhausting the CPU and memory.

Frozen processes are reactivated in a circular fashion, to allow some work to be done without overloading the owner's nodes. Later, when more resources become available, the load-balancing al-

gorithm migrates running processes away, thus allowing reactivation of more frozen processes.

1.8 Disruptive Configurations

In a MOSIX2 based multi-cluster, authorized administrators of each physical cluster can connect (disconnect) it to (from) the Grid at any time. After a request is issued to disconnect a cluster, all guest processes, if any, are moved out and all local processes that were migrated to other clusters are brought back. Note that guest processes can be migrated to any available node - not necessarily to their respective home-nodes. For this reason, users are not expected to login and/or initiate processes from remote clusters, since if that was allowed and those clusters were disconnected, the processes would have nowhere to return.

1.8.1 Time to Evacuate a Cluster

We measured the time to move out (evacuate) guest processes from a hosting cluster that is about to be disconnected from the Grid. We used 2 clusters, cluster A with 14 nodes and cluster B with 20 nodes. All the nodes were Xeon 3.06GHz servers that were connected by a 1Gb/s Ethernet.

The test started with a given set of identical CPU-bound processes from cluster A running on cluster B. We issued a *cluster-disconnect* command on cluster B that forced all the guest processes out. The test ended when all the processes were running in cluster A.

Table 2: Time to Evacuate a 20 Node Cluster

No. of Processes	Process Size	Migration	
		Time	Rate
40	512 MB	198 Sec	103 MB/Sec
40	1024 MB	397 Sec	103 MB/Sec
80	256 MB	192 Sec	106 MB/Sec
80	512 MB	388 Sec	105 MB/Sec

The results are presented in Table 2. Column 1 lists the total number of guest processes; Column 2 lists the size of each process; Column 3 shows the average (over 4 runs) of the measured migration times, and Column 4 shows the migration rates (MB/s).

The obtained results show that MOSIX can migrate a set of processes at an average (weighted over all cases) rate of 102.6 MB/s, which is about 93% of the maximal TCP/IP rate over a 1Gb/s Ethernet.

1.8.2 Long-Running Processes

The process migration, the freezing and the gradual reactivation mechanisms provide support to applications that need to run for a long time, e.g., days or even weeks, in different clusters across the Grid. As explained above, before a remote cluster is disconnected, all guest processes are moved out. These processes are frozen in their respective home-nodes and are gradually reactivated when Grid nodes become available again. For example, long processes from one department migrate at night to unused nodes in another department. During the day most of these processes are frozen in their home-cluster until the next evening.

1.9 Live Queuing

MOSIX2 incorporates a First-Come-First-Serve (FCFS) dynamic queuing that allows users to dispatch a large number of jobs, to run once sufficient resources are available.

Unlike other queuing systems, MOSIX2 uses "live-queuing" that allows queued jobs to preserve their full connection with their Linux environment (such as the controlling terminal, parent-process, signals, pipes, sockets, shared file-descriptors, etc.).

The MOSIX2 queuing system includes tools for tracing queued jobs, changing their priorities or the order of execution and for running parallel, e.g., MPI jobs.

1.9.1 Urgent Jobs

Despite the FCFS queuing policy, MOSIX allows to assign an additional number of "urgent" jobs to run regardless of the available resources and other limitations. Obviously, there are restrictions who is allowed to use this option and which jobs should be considered as "urgent". It is the sysadmin responsibility to ensure that at any given time, running those additional "urgent" jobs will in fact have sufficient memory/swap-space to proceed reasonably.

1.9.2 Out-of-order Jobs

MOSIX can be configured to guarantee a minimal (usually, small) number of jobs per user to start out of order, even when resources are insufficient. This, for example, allows users to run short jobs while very long jobs of other users are already running or are placed in the queue.

The only restriction on out of order jobs is that there is sufficient free memory, so that jobs that require much memory are not started. Jobs (per user) above this number and jobs that require more memory, are queued.

1.10 Checkpoint and Recovery

Checkpoint and recovery are supported for most computational MOSIX processes. When a process is checkpointed, its image is saved to a file. If necessary, the process can later be recovered from that file and continue to run from the point it was last checkpointed. Checkpoints can be triggered by the program itself, by a manual request or can automatically be taken on a time basis.

Some processes may not be checkpointed and other processes may not run correctly after recovery. For example, for security reasons checkpoint of processes with setuid/setgid privileges is not permitted. In general, checkpoint and recovery are not supported for processes that depend heavily on their Linux environment, such as processes with open pipes or sockets.

Processes that can be checkpointed but may not run correctly after being recovered include processes that rely on process-ID's of either themselves or other processes; processes that rely on parent-child relations; processes that rely on terminal job-control; processes that coordinate their input/output with other running processes; processes that rely on timers and alarms; processes that can not afford to lose signals; and processes that use system-V semaphores and messages.

1.11 Batch Jobs

MOSIX2 supports batch jobs that can be sent to any node in the local cluster (as opposed to non-batch jobs that require the specific environment of their dispatching node).

There are two types of batch jobs: Linux and MOSIX. Linux batch processes do not migrate, while MOSIX batch processes can migrate, but their home-node can be different than their dispatching node. MOSIX can assist both types by: (a) Queuing the job until resources are available (using "mosrun -q", "mosrun -S" or both); and (b) Selecting the best initial assignment for the job.

Batch jobs are started from binaries in another node and preserve only some of the caller's environment: they receive the environment variables; they can read from their standard-input and write to their standard output and error, but not from/to other open files; they receive signals, but if they fork, signals are delivered to the whole process-group rather than just the parent; they can not communicate with other processes on the calling node using pipes and sockets (other than standard input/output/error), semaphores, messages, etc. and can only receive signals, but not send them to processes on the calling node.

The main advantage of batch jobs is that they save time by not needing to refer to the dispatching-node to perform system-calls, and that temporary files can be created on the node where they start, preventing the dispatching node from becoming a bottleneck. This approach is therefore recommended for programs that perform a significant amount of I/O.

1.12 The Monitors

Two monitors, **mon** and **mmon**, provide information about resources in the Grid and each cluster, e.g., CPU-speed, load, free vs. used memory, swap space, number of active and inactive nodes, guest processes, etc.

2 Support for 32-bit and 64-bit

MOSIX2 supports both 32-bit (i386) and 64-bit (x86_64) architectures. 32-bit programs can run on 64-bit nodes and migrate as needed between 32-bit and 64-bit nodes. 32-bit programs must have a 32-bit home-node. 64-bit programs can not run on 32-bit computers.

It is possible to mix 32-bit and 64-bit nodes in the same cluster, but performance can be better when 32-bit and 64-bit nodes are kept as separate clusters within a multi-cluster Grid.

The installation script automatically detects and installs the appropriate binaries.

3 Running in a Virtual Machine

MOSIX can run in native Linux mode or in a Virtual Machine (VM). In native mode, performance is better, but it requires some modifications to the base Linux kernel, whereas a VM can run on top of any unmodified operating system that supports virtualization, including Linux, OS-X and Windows.

4 How to Request a Copy

The MOSIX web provides a free, limited evaluation copy for non-profit use. Distributions are provided as RPMs for openSUSE, for use in native Linux mode and as a pre-installed virtual-disk image that can be used to create a MOSIX virtual cluster on Windows and/or Linux computers.

Faculty and research staff can obtain an unlimited trial copy for use in academic and research organizations. For details see http://www.MOSIX.org/txt_grid.html.

Non-academic users can apply for a copy from http://www.mosix.com.au.

5 Conclusions

MOSIX2 is an operating system like management system that includes a comprehensive set of tools for sharing computational resources in a Linux cluster and a multi-cluster organizational Grid.

Logical clusters allow the sys-admin(s) to divide nodes into private partitions. Automatic resource discovery along with process migration and the priority method allow processes to migrate among nodes in the same cluster as well as among nodes in different clusters, to take advantage of remote nodes beyond the fixed set of allocated nodes in any cluster. This is particularly useful in shared clusters or when it is necessary to allocate a large number of nodes to one group, e.g., to meet a deadline. The flood prevention and the disruptive configuration provisions allow an orderly evacuation of disconnecting clusters as well as migrating away long running processes when remote resources are no longer available.

The home-node model along with the MOSIX software layer provide an environment in which users need not modify applications, link applications with any library, login or copy files to remote nodes. It also provides a secure run time environment to hosting nodes by preventing migrated processes from accessing or modifying local resources. In particular, this means that even a user with one workstation can use Grid resources by migrating processes to available nodes, and also store such processes locally when these nodes are no longer available. Other supported features include batch jobs, checkpoint and recovery, live queuing and an on-line monitor.

A production campus Grid with 15 MOSIX clusters is operational in our university. The features of MOSIX2 allow better utilization of Grid resources, including idle workstations in student labs. Performance of user's applications across different clusters in this Grid was found to be nearly identical to that of a single cluster [1].

References

[1] A. Barak, A. Shiloh and L. Amar, " An Organizational Grid of Federated MOSIX Clusters," *Proc. 5-th IEEE International Symposium on Cluster Computing and the Grid (CCGrid05)*, Cardiff, May 2005.

[2] A. Keren, and A. Barak, "Opportunity Cost Algorithms for Reduction of I/O and Interprocess Communication Overhead in a Computing Cluster," *IEEE Tran. Parallel and Dist. Systems*, 14(1), pp. 39–50, 2003.

[3] A. Barak, O. La'adan and A. Shiloh, "Scalable Cluster Computing with MOSIX for Linux," *Proc. 5th Annual Linux Expo*, Raleigh, NC, pp. 95–100, 1999.

[4] LZOP, http://www.lzop.de, 2009.

[5] MOSIX, http://www.MOSIX.org, 2009.

[6] Amar L., Barak A., Drezner Z. and Okun M., "Randomized Gossip Algorithms for Maintaining a Distributed Bulletin Board with Guaranteed Age Properties," 2008.