

The MOSIX Cluster Operating System for High-Performance Computing on Linux Clusters, Multi-Clusters and Clouds

A White Paper

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

OVERVIEW

MOSIX¹ is a cluster operating system targeted for High-Performance Computing (HPC) on Linux platforms, including clusters, multi-clusters and clouds.

The unique features of MOSIX provide users and applications with the impression of running on a single computer with multiple processors, without changing the interface and the run-time environment of their respective login nodes. For example, in a MOSIX cluster users can run applications that create multiple processes, then let MOSIX seek resources and automatically distribute processes among nodes, e.g., to improve the overall performance, without changing the run-time environment of the migrated processes. As a result, users need not change or even link applications with any special library, they need not modify applications, login or copy files to remote nodes or even know where their programs run.

MOSIX Version 1 was originally developed to manage a single cluster [4]. MOSIX Version 2 (MOSIX2) for Linux-2.6 and Linux-3 was extended with a comprehensive set of new features for managing clusters, multi-clusters, e.g., among different groups in an organization [6] and clouds. For example, one multi-cluster feature allows owners of clusters to share their resources from time to time, while still preserving the autonomy of the owners to disconnect their clusters at any time, without sacrificing running guest processes from other clusters. Another feature, MOSIX Reach the Clouds (MRC), is a tool that allows applications to run in a hybrid environment on remote clusters such as clouds [2], without the need to pre-copy files to these clusters, see Sec. IV for details. MRC can run on both Linux and various MOSIX cluster configurations.

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. 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, e.g., for load-balancing, or to move processes from a disconnecting cluster.

In a MOSIX 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, where clusters can be shared (symmetrically or asymmetrically) among users of different clusters. Users need not know the details of the configuration nor the state of any resource.

Other features of MOSIX include migratable sockets - for direct communication between migrated processes; a secure run-time environment (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 as well as checkpoint and recovery.

MOSIX is implemented as a set of utilities that provide users and applications with a distributed Linux-like run-time environment. MOSIX supports most Linux features that are relevant to ordinary, non-threaded Linux applications, so that most Linux programs can run unchanged.

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 [6].

MOSIX should be used in trusted environments over secure networks, where only authorized nodes are allowed. These requirements are standard within private clusters and intra-organizational clouds, but usually not elsewhere. Other than these requirements, MOSIX could be used in any configuration with multiple computers.

A production campus multi-cluster, with 18 MOSIX clusters (about 1000 CPUs) can be seen at <http://www.MOSIX.org/webmon>. Most clusters are private, belonging to research groups in various departments. The

¹MOSIX[®] is a registered trademark of A. Barak and A. Shiloh. Copyright © A. Barak 2011. All rights reserved.

rest are shared clusters that are made of workstations in student labs. The features of MOSIX allow better utilization of resources, including idle workstations in student labs, by users who need to run HPC applications but cannot afford such a large private cluster.

II. BUILDING BLOCKS

This section describes the two building blocks of MOSIX: configurations and processes.

A. Configurations

1) *A MOSIX Cluster*: A MOSIX cluster is a set of connected computers (that may include servers and workstations), called "nodes", that are administrated by a single owner and run the same version of MOSIX. In a MOSIX cluster, each node maintains information about availability and the state of the resources of all the nodes, see Sec.III-A for details.

2) *A MOSIX Multi-cluster*: A MOSIX multi-cluster (sometimes also referred to as "an intra-organizational multi-cluster") is a collection of private MOSIX clusters that run the same version of MOSIX and are configured to work together.

A MOSIX multi-cluster usually belongs to the same organization, but each cluster may be administrated by a different owner or belongs to a different group.

The cluster-owners are willing to share their computing resources at least some of the time, but are still allowed to disconnect their clusters from the multi-cluster at any time.

In a MOSIX multi-cluster, each node maintains information about availability and the state of the resources of all the nodes in all the connected clusters. Different clusters may (or may not) have a shared environment such as a common NFS file-system. Nevertheless, MOSIX processes can run in remote clusters while still using the environment provided by their respective private home clusters. From the user's perspective, MOSIX transforms such a multi-cluster into a single cluster by preserving the user's local run-time environment.

In MOSIX multi-clusters there is usually a high degree of trust, i.e., a guarantee that applications are not viewed or tampered with when running in remote clusters. Other possible safety requirements are a secure network and that only authorized nodes, with identifiable IP addresses, are included.

3) *A MOSIX Cloud*: A MOSIX cloud is a collection of entities such as MOSIX clusters; MOSIX multi-clusters; Linux clusters (such as a group of Linux servers); individual workstations and Virtual Machines (VM). Each entity may possibly run a different version of Linux or MOSIX.

In a MOSIX cloud, different entities are usually administrated by different owners and rarely share any file systems (such as NFS). In this cloud, nodes in each entity are aware of one or more nodes in other entities, including their IP-addresses and services they are willing to provide, but there is no on-going automatic flow of information between entities.

In a MOSIX cloud, users can launch applications from their workstations or a private home-cluster, on target nodes of other entities. These applications have access to files on nodes of these entities, while still allowing the applications to access

files on their launching node. This is accomplished by the MOSIX Reach Clouds (MRC), described in Sec. IV, which allows applications to run in remote nodes, without the need to copy files to/from remote clusters.

B. Processes

MOSIX recognizes two types of processes: Linux processes and MOSIX processes. Linux processes are not affected by MOSIX - they run in native Linux mode and cannot 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 be assigned by the "-b" option of "mosrun" to the least loaded nodes in the cluster (but not to nodes in other clusters, for which the MRC tool can be used, see Sec. IV for details).

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 [4]. Child processes of MOSIX processes remain under the MOSIX discipline (with the exception of the **native** utility, that allows programs, mainly shells, already running under mosrun, to spawn children in native Linux mode). Below, all references to processes mean MOSIX processes.

III. UNIQUE FEATURES OF MOSIX

This section describes the unique features of MOSIX which are intended to provide users and applications with the impression of running on a single computer with multiple processors.

A. Automatic Resource Discovery

Resource discovery is performed by an on-line information dissemination algorithm, providing each node in all the clusters with the latest information about availability and the state of system-wide resources [1]. 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 randomly chosen nodes. 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 [1] we presented bounds for the age properties and the rates of propagation of the above algorithm.

B. Process Migration

MOSIX supports (preemptive) process migration [4] among nodes in a cluster and in a multi-cluster. Process migration can

be triggered 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 [9].

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 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 on-line algorithms [7] 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.

C. The Run-Time Environment

MOSIX is implemented as a software layer that allows applications to run in remote nodes, away from their respective home-nodes. This is accomplished by intercepting all system-calls, then if the process was migrated, most 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 processes seem to be running in their home-nodes. 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 traditional 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) *Overhead of migrated processes:* The following four real-life applications, each with a different amount of I/O, illustrate the overhead of running migrated processes. 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**ium (molecular dynamics), uses a larger amount of I/O. Finally, **BLAT** (bioinformatics) uses a moderate amount of I/O.

We used identical Xeon 3.06GHz servers that were connected by a 1Gb/s Ethernet and ran each program in three different ways:

- 1) As a local (non-migrated) **Linux** process.
- 2) As a migrated MOSIX process to another node in the **same cluster**.
- 3) As a migrated MOSIX process to a node in a **remote cluster**, located about 1 Km away.

The results (averaged over 5 runs) are shown in Table I. 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 by each program. 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 show the run-times and the slowdowns in the **remote cluster** across campus.

TABLE I
LOCAL VS. REMOTE RUN-TIMES (SEC.)

	RC	SW	JEL	BLAT
Linux	723.4	627.9	601.2	611.6
Total I/O (MB)	0	90	206	476
I/O block size	–	32KB	32KB	64KB
Syscalls	3,050	16,700	7,200	7,800
Same cluster	725.7	637.1	608.2	620.1
Slowdown	0.32%	1.47%	1.16%	1.39%
Remote cluster	727.0	639.5	608.3	621.8
Slowdown	0.50%	1.85%	1.18%	1.67%

Table I 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 campus, 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.

2) *Migratable sockets:* Migratable sockets allow 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 other clusters (that are willing to accept them).

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

3) *A secure run-time environment*: The MOSIX software layer guarantees that a migrated (guest) process cannot 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, cannot access resources in the hosting node, while 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.

D. 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 private clusters can be shared (symmetrically or asymmetrically) among users of each cluster. Public clusters can also be set to be shared among users of private clusters.

E. 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 other clusters are disconnected or reclaimed, causing a large number of processes to migrate back to their respective home-clusters.

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 secondary storage. 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, as more resources become available, the load-balancing algorithm migrates running processes away, thus allowing reactivation of more frozen processes.

F. Disruptive Configurations

In a multi-cluster configuration, authorized administrators of each physical cluster can connect (disconnect) it to (from) the

pool at any time. If the cluster is a Linux cluster then all open connections to other clusters are closed, which may result in losing running jobs. In the case of a MOSIX cluster, all guest processes (if any) are moved out and all local processes that were migrated to other MOSIX clusters are brought back. Note that guest processes can be migrated to any available node in MOSIX clusters - not necessarily to their respective home-nodes. It is therefore recommended that users do not login and/or initiate processes from remote MOSIX clusters, since if they did so, then their processes would have nowhere to return.

1) *Time to disconnect a cluster*: This test shows the times to move out guest processes from a hosting cluster in order to disconnect it from a multi-cluster. Two MOSIX clusters, with identical Xeon 3.06GHz servers that were connected by a 1Gb/s Ethernet were used: cluster A with 14 nodes and cluster B with 20 nodes.

First, a given set of identical CPU-bound processes were started on cluster A and were forced to migrate to cluster B. The test started by a *cluster-disconnect* command on cluster B, that forced all the guest processes out. The test ended when all the processes returned to cluster A.

TABLE II
TIMES TO DISCONNECT 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 II. The first two columns list the number of guest processes and the size of each process; Column 3 shows the average (over 4 runs) of the migration times and Column 4 shows the migration rates.

The 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.

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. As explained above, before a MOSIX cluster is disconnected, all guest processes are moved out. These processes are frozen in their respective home-nodes and are gradually reactivated when system-wide MOSIX 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.

G. Dynamic Queuing

The number and type of jobs that are released by the MOSIX queuing system depends on the current availability of the cluster/multi-cluster resources.

H. Live Queuing

Unlike other queuing systems, MOSIX 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.

IV. MOSIX REACH THE CLOUDS

MOSIX Reach the Clouds (MRC) is a tool that allows applications to run in remote computers in any MOSIX cloud entity (see Sec. II-A3), without pre-copying files to these computers. MRC users launch applications from their workstation (or private home-cluster) on target nodes of other entities. MRC applications run in a hybrid environment, where some of their files are on their launching node and the rest are on target nodes. As a result, MRC can be used in diverse ways to promote different modes of file-sharing among different computers and users.

MRC can run on both Linux computers and MOSIX clusters. The hybrid environment on target nodes can be used for remote file access; file-sharing among different computers and users (even more than two computers when MRC is used recursively); and for running applications that need to use both private and shared data. Standard I/O remains on the launching computer. All “mosrun” features can be used on clouds running MOSIX.

The rationale behind MRC is to harness resources in remote entities without copying data files there. As such, MRC can be useful to users that need to run applications but do not wish to store data on commercial clouds [2], or that cannot determine in advance which files (or file-parts) will be needed. MRC provides consistent access to files, even among multiple MRC jobs that run on different targets. Due to network latencies, MRC is more suited to run compute intensive and other applications with low to moderate amounts of I/O.

MRC consists of two parts: a launching program that can send jobs from a head-node to designated target nodes, and a run-time environment that provides file services to running jobs on target computers. The head-node could be the user’s workstation and any computer that has access to the user’s files, possibly even a virtual machine in a commercial cloud, if the user’s files are stored there. Note that the head-node should not be confused with the MOSIX home-node, which, in the case of using MRC to launch MOSIX processes, is the target node rather than the head-node.

The head-node and the target nodes can run Linux or MOSIX. If the target node is part of a MOSIX cluster, then MRC jobs can benefit from all the MOSIX features. In particular, MOSIX processes generated by MRC jobs can be automatically dispersed among the nodes of that cluster or even among other MOSIX clusters in a multi-cluster. If a target

node runs Linux (not MOSIX), then MRC jobs can only run there as native Linux jobs.

Launching a job (from the head-node) is done by the “mrc” command, see the mrc manual for details.

V. OTHER SERVICES

This section describes additional services that are provided to MOSIX processes.

A. 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 application can later be recovered from that file and continue to run from the point it was last checkpoint. Checkpoints can be triggered by the program itself, by a manual request or can automatically be taken periodically.

Some processes may not be checkpoint 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 cannot afford to lose signals; and processes that use system-V semaphores and messages.

B. Queuing

MOSIX 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.

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

In addition to the unique dynamic and live-queuing, described in the previous section, MOSIX also supports fair-share, urgent and out-of-order jobs.

1) *Fair-share*: An optional fair-share policy allows the interleaving of queued-jobs among users, thus protecting users against the possibility that other users that came first effectively do not allow their jobs to start at all. Fine-tuning among users of the fair-share policy is also available.

2) *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 on who is allowed to use this option and which jobs should be considered as “urgent”. It is the sys-admin’s responsibility to ensure that at any given time, even when running the maximum allowed number of those additional “urgent” jobs, there will be sufficient memory/swap-space to proceed reasonably.

3) *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 must be 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.

C. Batch Jobs

MOSIX supports batch jobs that can be sent to any node in the local cluster (as opposed to interactive 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, although 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 cannot 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.

D. Support for 32-bit and 64-bit

MOSIX 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 cannot 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 pool.

The installation script automatically detects and installs the appropriate binaries.

E. Running in a Virtual Machine

MOSIX can run in native Linux mode or in a Virtual Machine (VM). In native mode, performance is better [8], 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.

F. Monitors

Two monitors, **mon** and the optional **mmon**, provide information about resources in the multi-cluster and each cluster, e.g., CPU-speed, load, free vs. used memory, swap space, number of active and inactive nodes, guest processes, etc. Type “help” to see all the available options.

VI. DOCUMENTATION

The MOSIX web includes a wiki, answers to frequently asked questions (FAQ) and a list of selected MOSIX publications [5].

The wiki includes installation and configuration instructions, pointers to the latest release and the change-log, to the user’s and the administrator’s guide and the MOSIX manuals, an overview and tutorial presentations, a list of MOSIX related publications, a reference to the MOSIX version 1 book and description of the MOSIX history.

VII. HOW TO OBTAIN A COPY

A copy of MOSIX for academic, non-profit and evaluation is available via the MOSIX web [5]. Distributions are provided for use with native Linux, as RPMs for openSUSE and as a pre-installed virtual-disk image that can be used to create a MOSIX virtual cluster on Windows and/or Linux computers.

VIII. CONCLUSIONS

MOSIX is an operating system-like management system that consists of a comprehensive set of tools for sharing computational resources in Linux clusters, multi-clusters and clouds. Its main features are geared for ease of use by providing the impression of running on a single computer with multiple processors. This is accomplished by preserving the interface and the run-time environment of the login (home) node for applications that run in other nodes. As a result, users need not modify or link applications with any library, they need not login or copy files to remote nodes or even know where their programs run.

The unique features of MOSIX include automatic resource discovery, dynamic workload distribution by process migration, a priority method that allows processes to migrate among nodes in a multi-cluster, to take advantage of available resources beyond the allocated nodes in any private 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 migration of processes from disconnecting clusters, including long running processes when remote resources are no longer available. Other unique features include live queuing and a tool to run applications on clouds, without the need to pre-copy files to these clusters.

REFERENCES

- [1] Amar L., Barak A., Drezner Z. and Okun M., "Randomized Gossip Algorithms for Maintaining a Distributed Bulletin Board with Guaranteed Age Properties," *Concurrency and Computation: Practice and Experience*, Vol. 21, pp. 1907-1927, 2009.
- [2] Armbrust M., et al, "Above the Clouds: A Berkeley View of Cloud Computing," Technical Report EECS-2009-28, 2009.
- [3] Barak A., Ben-Nun T., Levy E. and Shiloh A., "A Package for OpenCL Based Heterogeneous Computing on Clusters with Many GPU Devices," *Proc. Workshop on Parallel Programming and Applications on Accelerator Clusters (PPAAC10), IEEE Cluster 2010*, Crete, 2010.
- [4] Barak A., La'adan O. and Shiloh A., "Scalable Cluster Computing with MOSIX for Linux," *Proc. 5th Annual Linux Expo*, Raleigh, NC, pp. 95-100, 1999.
- [5] <http://www.MOSIX.org>.
- [6] Barak A., Shiloh A. and Amar L., "An Organizational Grid of Federated MOSIX Clusters," *Proc. 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid05)*, Cardiff, 2005.
- [7] Keren A., and Barak A., "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.
- [8] Maoz T., Barak A. and Amar L., "Combining Virtual Machine Migration with Process Migration for HPC on Multi-Clusters and Grids," *Proc. IEEE Cluster 2008*, Tsukuba, 2008.
- [9] <http://www.lzop.de>.