The MOSIX Cluster Management System for Parallel Computing on Linux Clusters and Multi-Cluster Private Clouds

White Paper

http://www.MOSIX.org

OVERVIEW

MOSIX[1] is a cluster management system that provides users and applications with a single-system image [1]. In a MOSIX cluster, users can run multiple processes on different nodes by allowing MOSIX to seek resources and automatically allocate (and migrate) processes among nodes, without changing the interface and the run-time environment of their respective login nodes. As a result, users need not change or 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.

Originally, MOSIX distributions for Linux-2.2 and Linux-2.4 were designed to manage a single cluster [4]. Starting from Linux-2.6, MOSIX was extended with a comprehensive set of features for managing clusters, multi-clusters, e.g., among different groups in an organization [6] and clouds. Owners of clusters can, for example, share their resources from time to time, while still preserving their right to disconnect their clusters at any time, doing so without sacrificing running guest processes from other clusters. The latest distribution is MOSIX release 4 (MOSIX-4).

MOSIX supports both interactive processes and batch jobs. It incorporates dynamic resource discovery and automatic workload distribution. The resource discovery algorithm provides each node with the latest information about resource availability and the state of other 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 multi-cluster private cloud, assigning priorities ensures that local processes and processes with a higher priority can force out guest (migrated) processes with a lower priority. Flexible configurations are supported, 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; as well as checkpoint and restart.

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 such programs can run unchanged. The latest distribution, MOSIX-4, is implemented in user-mode and no longer requires a kernel patch.

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 clusters and private clouds. Other than these requirements, MOSIX could be used in any configuration with multiple computers.

A production campus private cloud, with 7 MOSIX clusters (about 1300 cores) can be seen at http://www.MOSIX.org/webmon. Most clusters are private, belonging to different research groups. The rest are shared clusters 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 parallel applications but cannot afford such a large private cluster.

[1]MOSIX® is a registered trademark.

II. Building Blocks

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

A. Configurations

A MOSIX *cluster* is a set of connected servers and workstations (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 resources in the other nodes, see Sec.III-A for details.

A MOSIX multi-cluster private *cloud* is a collection of MOSIX clusters that run the same version of MOSIX and are configured to work together. A MOSIX cloud usually belongs to the same organization, but each cluster may be administrated by a different owner or belong 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 cloud at any time.

In a MOSIX cloud, each node maintains information about the availability and state of resources of the other 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 cloud into a single cluster by preserving the user's local run-time environment.

In MOSIX clouds 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.

B. Processes

MOSIX processes are usually user applications that are suitable and can benefit from migration. 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. The node in which a process was created is called its home-node. [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

The unique features of MOSIX are intended to provide users and applications with the impression of running on a single system.

A. 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 [2]. 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. A study of bounds for the age properties and the rates of propagation of the above algorithm was presented in [2].

B. Process migration

MOSIX supports (preemptive) process migration [4] among nodes in a cluster and in a cloud. 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 the 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**lium (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) 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 **Local** 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 Local 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**
Local	**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 Local 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. When this limit is reached, additional processes 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 MOSIX cloud, authorized administrators of each private cluster can connect (disconnect) it to (from) the cloud at any time. In a Linux cluster, 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 the cloud - 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) 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.

IV. 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 checkpointed. Checkpoints can be triggered by the program itself, by a manual request or can automatically be taken periodically.

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

B. Running in a virtual machine

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

but a VM can run on top of any operating system that supports virtualization, including OS-X and Windows.

C. Monitors

The monitor, **mosmon** provides information about resources in the cluster, e.g., CPU-speed, load, free vs. used memory, swap space, number of active nodes, etc. Type "help" to see the available options.

V. Documentation

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

The wiki includes installation and configuration instructions, information about the latest release; the change-log; the administrator's, user's and programmer's guides and the MOSIX manuals; overview and tutorial presentations; a list of MOSIX related publications and the MOSIX history.

VI. Obtaining a Copy

Subject to the MOSIX software license agreement, a copy of MOSIX is available via the MOSIX web [5].

VII. Conclusions

MOSIX is a runtime management system that consists of a comprehensive set of tools for sharing computational resources in Linux clusters and clouds. Its main features are geared for ease of use by providing the impression of running on a single computer with many cores. 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 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 cloud, 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] http://en.wikipedia.org/wiki/Single_system_image

[2] 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.

[3] Barak A., Guday G. and Wheeler R., The MOSIX Distributed Operating System, Load Balancing for UNIX. LNCS Vol. 672, Springer-Verlag, ISBN 0-387-56663-5, New York, May 1993.

[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. 5-th 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.