MOSIX2

Cluster and Multi-Cluster Management

Administrator's Guide

Revised for MOSIX-2.25.0.0

January 2009

Preface

This is an administrator guide of MOSIX* Version 2 **(MOSIX2)**, a management system targeted for high performance computing on x86 based (32-bit and 64-bit) Linux clusters and multi-clusters, such as intra-organizational Grids.

MOSIX can be viewed as a multi-cluster operating system that 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. Users do not need to modify or link applications with any library, login to remote nodes or even copy files to remote nodes.

MOSIX Version 1 was originally developed to manage a single cluster. MOSIX2 for Linux-2.6 was extended with a comprehensive set of new features that can manage clusters and multi-clusters.

Further information is available in the MOSIX web at $http://www.mosix.org$.

*MOSIX® is a registered trademark of Amnon Barak and Amnon Shiloh.

Contents

Preface **iii**

Part I Administrator's Guide **1**

1 Terminology **3**

2 What MOSIX is and is not **5**
2.1 What MOSIX is . . . 5
2.1.1 The main cluster features of MOSIX . . . 5
2.1.2 Additional multi-cluster features . . . 6
2.2 What MOSIX is not . . . 6

3 System requirements **7**

4 Configuration **9**
4.1 General . . . 9
4.2 Configuring the single cluster . . . 10
4.2.1 Participating nodes . . . 10
4.2.2 Advanced options . . . 10
4.3 Configuring the multi-cluster . . . 11
4.3.1 Partner-clusters . . . 11
4.3.2 Which nodes are in a partner-cluster . . . 12
4.3.3 Partner-cluster relationship . . . 12
4.3.4 Priorities . . . 13
4.3.5 Priority stabilization . . . 13
4.3.6 Maximum number of guests . . . 13
4.4 Configuring the queuing system . . . 13
4.4.1 Queuing is an option . . . 13
4.4.2 Selecting queue-managers . . . 13
4.4.3 Advanced . . . 14
4.5 Configuring the freezing policies . . . 15
4.5.1 Overview . . . 15
4.5.2 Process classes . . . 16
4.5.3 Freezing-policy details . . . 16
4.5.4 Disk-space for freezing . . . 17
4.5.5 Ownership of freezing-files . . . 18
4.6 Configuring the processor-speed . . . 18

4.6.1 Standard processor 18
4.6.2 No one benchmark for all 18
4.6.3 Setting the CPU speed manually 18

5 Storage allocation **19**
5.1 Swap space 19
5.2 MOSIX files 19
5.3 Freezing space 19
5.4 Private-file space 20

6 Managing jobs **21**
6.1 Monitoring (mon) 21
6.2 Listing MOSIX processes (mosps) 21
6.3 Controlling running processes (migrate) 21
6.4 Viewing and controlling queued processes (mosq) 21
6.5 Controlling the MOSIX node (mosctl) 21

7 Security **23**
7.1 Abuse by gaining control of a node 23
7.2 Abuse by connecting hostile computers 23
7.3 Multi-cluster password 23
7.4 Organizational Multi-cluster Grid 23
7.5 Batch password 24

Part II Manuals **25**

8 Manuals **27**

Part I
Administrator's Guide

Chapter 1

Terminology

The following terms are used throughout this guide:

- **Node** - a participating computer (physical or virtual), whose unique IP address is configured to be part of a MOSIX cluster or multi-cluster.
- **Processor** - a CPU (Central Processing Unit or a Core): most recent computers have several processors. (Hyper-Threads do not constitute different processors).
- **Process** - a unit of computation that is started by the "fork" (or "vfork") system call and maintains a unique identifier (PID) throughout its life-time (for the purpose of this document, units of computation that are started by the "clone" system call are called "threads" and are not included in this definition).
- **Job** - an instance of running an executable program (with given parameters and environment): a job can result in one or more processes.
- **Home-node** - the node to which a process "belongs": a process sees the world (file-systems, network, other processes, etc.) from the perspective of this node. The home-node is usually the node from where the user started their job. Note that the home-node should not be confused with the common concept of a "head-node": while the system-administrator can choose to assign head-nodes, MOSIX does not require it and in not aware of that.
- **Home-cluster** - the cluster to which the home-node of a process belongs.
- **Local process** - a process that runs in its home-node.
- **Guest process** - a process whose home-node is elsewhere, but is currently running here (on the node being administered).
- **Cluster** - one or more computers (nodes) that are owned and managed by the same entity (a person, a group of people or a project). Each MOSIX cluster may range from a single workstation to a large combination of computers - workstations, servers, blades, multi-core computers, etc. possibly of different speeds and number of processors and possibly in different locations. Note that a MOSIX cluster can at times be different than hardware clusters. For example, it can consist of several hardware-clusters or just part of a hardware-cluster.

- **Multi-cluster** - a collection of clusters whose owners trust each other and wish to share some computational resources among them.
- **Partition** - a subset of the nodes of a MOSIX cluster. This is an optional (configurable) feature that allows splitting of a cluster into several parts. Each node may only belong to one partition. Within a cluster, each partition views other partitions nearly as a different cluster, but other clusters perceive all the partitions as belonging to one cluster.

Chapter 2

What MOSIX is and is not

2.1 What MOSIX is

MOSIX is an extension of the operating system for managing clusters and multi-clusters efficiently.

MOSIX is intended primarily for High Performance Computing (HPC).

The main tool employed by MOSIX is **preemptive process migration** (a process may start on one node, then move smoothly to other nodes, repeated as necessary, possibly even returning to its first node). Process migration occurs automatically and transparently, in response to resource availability.

Process migration is utilized to optimize the overall performance.

2.1.1 The main cluster features of MOSIX

- Provides aspects of a single-system image.
  - Users can login on any node and do not need to know where their programs run.
  - No need to modify or link applications with special libraries.
  - No need to copy files to remote nodes.
- Automatic resource discovery and workload distribution:
  - Load-balancing by process migration.
  - Migrating processes from slower to faster nodes and from nodes that run out of free memory.
- Migratable sockets for direct communication between migrated processes.
- Provides a secure run time environment (sandbox) for guest processes.
- Supports live queuing - queued jobs preserve their full generic Linux environment.
- Supports batch jobs.
- Supports checkpoint and recovery.
- Supports both 32-bit and 64-bit x86 architectures.
- Includes tools for automatic installation and configuration, and on-line monitors.

2.1.2 Additional multi-cluster features

- Supports disruptive configurations:
  - Clusters can join or leave the multi-cluster at any time.
  - Guest processes move out before disconnecting a cluster.
- Clusters can be shared symmetrically or asymmetrically.
- Cluster owner can assign different priorities to guest processes from other clusters.

2.2 What MOSIX is not

MOSIX is not:

- A Linux distribution.
- A Linux kernel.
- A cluster set-up and installation tool.

MOSIX does not:

- Improve performance of intensive I/O jobs.
- Improve performance of non-computational server-applications, such as web or mail servers.
- Support High-Availability.
- Support shared-memory and threaded jobs.

Chapter 3

System requirements

Any combination of 32-bit and 64-bit computers of the x86 architecture, including both Intel or AMD, can be used.

Multiprocessor computers (SMP, dual-core, quad-core or multi-core) are supported, but all the processors of each node must be of the same speed.

All the nodes must be connected to a network that supports TCP/IP and UDP/IP. Each node should have a unique IP address in the range 0.1.0.0 to 255.255.254.255 that is accessible to all the other nodes.

TCP/IP ports 249-253 and UDP/IP ports 249-250 should be reserved for MOSIX (not used by other applications or blocked by a firewall).

MOSIX can be installed on top of any Linux distribution: mixing of different Linux distributions on different nodes is allowed.

Chapter 4

Configuration

4.1 General

The script "mosconf" will lead you step-by-step through the MOSIX configuration.

Mosconf can be used in two ways:

1. You can configure (or re-configure) MOSIX on the cluster-node where it should run. If so, just press <Enter> at the first question that "mosconf" presents.

2. In clusters (or parts of clusters) that have a central repository of system-files, containing their root image(s), you can make changes in the central repository instead of having to manually update each node separately.

   This repository can for example be NFS-mounted by the cluster as the root file-system, or it can be copied to the cluster at boot time, or perhaps you have some cluster-installation package that uses other methods to reflect those files to the cluster. Whichever method is used, you must have a directory on one of your servers, where you can find the hierarchy of system-files for the clusters (in it you should find subdirectories such as /etc, /bin, /sbin, /usr, "'lib", "mnt", "proc" and so on).

   At the first question of "mosconf", enter the full pathname to this repository.

When modifying the configuration there is no need to stop MOSIX - most changes will take effect within a minute. However, after modifying any of the following:

- The list of nodes in the cluster (/etc/mosix/mosix.map).
- The IP address used for MOSIX (/etc/mosix/mosip).
- The node's topological features (/etc/mosix/myfeatures),

you must commit your changes by running the command "setpe" - however, this is not necessary when you are using "mosconf" locally (option 1 above).

The MOSIX configuration is maintained in the directory "etc/mosix".

4.2 Configuring the single cluster

4.2.1 Participating nodes

The most important configuration task is to inform MOSIX which nodes participate in your cluster. In "mosconf" you do this by selecting "Which nodes are in this cluster".

Nodes are identified by their IP address (see the advanced options below if they have more than one): commonly the nodes in a cluster have consecutive IP addresses, so it is easy to define them using the IP address of the first node followed by the number of nodes in the range, for example, if you have 10 nodes starting from 192.168.3.1 to 192.168.3.10, type "192.168.3.1" followed by "10". If there are several such ranges, you need to specify all of them and if there are nodes with an isolated IP address, you need to specify them as ranges of 1.

If your IP addresses are mostly consecutive, but there are a few "holes" due to some missing computers, it is not a big deal - you can still specify the full range, including the missing computers (so long as the IP addresses of the "holes" do not belong to other computers elsewhere).

Specifying too many nodes that do not actually exist (or are down) has been known to produce excessive ARP broadcasts on some networks due to attempts to contact the missing nodes. This was found to be due to a bug in some routers, but unfortunately many routers have this bug.

It is always possible to add or delete nodes without stopping MOSIX: if you do it from a central repository, you need to run "setpe" on All your cluster nodes for the changes to take effect.

4.2.2 Advanced options

The following are advanced options (if no advanced options were previously configured, type "+" in "mosconf"). As above, it is not necessary to stop MOSIX for modifying advanced options, just run "setpe" after making the changes from a central repository.

Nearby or distant nodes

To optimize process migration, for each range of nodes, you can define whether they are "distant" or "near" the nodes that you are configuring. The reason is that when networking is slow, it is better to compress the memory image of migrating processes: it takes CPU time, but saves on network transfer time and volume. If however the nodes are near, it is better not to compress. As a general guideline, specify "distant" if the network is slower than 1GB/sec, or is 1GB/sec and the nodes are in different buildings, or if the nodes are several kilometers away.

Outsider nodes

For each range of nodes, you can define whether they are "outsider"s. Only processes that are allowed to migrate to other clusters in the multi-cluster are allowed to migrate to "outsider" nodes. This option was intended to allow users to prevent certain programs from migrating to unsuitable computers, such as computers that do not support the full machine instruction-set of their home-node. This option is almost completely replaced by cluster-partitions.

Cluster partitions

Clusters can be divided into partitions, for example in order to divide the nodes among several users. It is also recommended (though not required), on mixed clusters, to place 32-bit nodes in one partition and 64-nodes in another. The main feature of partitions is that processes will only migrate to other partitions if they are allowed to migrate to other clusters in the multi-cluster (using the "mosrun -G" flag), yet other clusters see all your partitions as one cluster, so you can change your cluster's partitioning at any time without needing to coordinate your internal division of your cluster with system-administrators of other clusters.

For each range of nodes (consecutive IP addresses) you can define to which partition it belongs (use any positive integer to number your partitions).

Aliases

Some nodes may have more than one IP address so that network packets that are sent from them to different nodes can be seen as arriving from different IP addresses. For example, a junction node can have a dual function of both being part of a logical MOSIX cluster as well as serve as a router to a physical cluster: nodes inside the physical cluster and outside it may see different IP addresses coming from the junction node. In MOSIX, each node must be identified by a unique IP address, so one of the junction-node's IP addresses is used as its main address, while the others can be configured as aliases: when MOSIX receives TCP/UDP connections from an alias IP address, it recognizes them as actually coming from the main address.

Unusual circumstances with IP addresses

There are rare cases when the IP address of a node does not appear in the output of "ifconfig" and even more rare cases when more than one IP address that belongs to a node is configured as part of the MOSIX cluster AND appears in the output of "ifconfig" (for example, a node with two Network-Interface-Cards sometimes boots with one, sometimes with the other and sometimes with both, so MOSIX has both addresses configured "just in case"). When this happens, you need to manually configure the main MOSIX address (using "Miscellaneous policies" of "mosconf").

4.3 Configuring the multi-cluster

4.3.1 Partner-clusters

Now is the time to inform MOSIX which other clusters (if any) are part of your MOSIX multi-cluster.

In a MOSIX multi-cluster, there is no need for each cluster to be aware of all the other clusters, but only of those partner-clusters that we want to send processes to or are willing to accept processes from.

You should identify each partner-cluster with a name: usually just one word (if you need to use more, do not use spaces, but '-' or '_' to separate the words). Please note that this name is for your own use and does not need to be identical across the multi-cluster. Next you can add a longer description (in a few words), for better identification.

4.3.2 Which nodes are in a partner-cluster

In most cases, you do not want to know exactly which nodes are in a partner-cluster - otherwise you would need to update your configuration whenever system-administrators make changes to partner-clusters: instead you only need to know about a few nodes (usually one or two are sufficient) that belong to each partner-cluster - these are called "core-nodes". If possible, choose the core-nodes so that at any given time at least one of them would up and running.

There are three methods of determining which nodes are in a partner-cluster:

1. The default and easiest method of operation is to trust the core-nodes to correctly inform your cluster which nodes are in their cluster.

2. MOSIX obtains the list of nodes from the core-nodes, but you also configure a list of allowed nodes. If a core-node informs us that its cluster includes node(s) that are not on our list - ignore them. The result is the intersection of "our list" and "their list".

3. Configure the list of nodes of the partner-cluster locally, without consulting any core-nodes.

Even when trusting the core-nodes, you can still specify particular nodes that you want to exclude.

Nodes of partner-clusters are defined by ranges of IP addresses, just like in the local cluster - see above. As above, a few "holes" are acceptable.

For each range of nodes that you define, you will be asked (the questions are in the singular case if the range is of only one node):

1. Are these core-nodes [Y/n]?

2A. Should these nodes be excluded [y/N]?

or for core-nodes:

2B. The following option is extremely rare, but is permitted: are these nodes ONLY used as core nodes, but not as part of 'cluster-name' [y/N]?
Note: it is permitted to define nodes that are both core-nodes AND excluded: they tell which nodes are in their cluster, but are not in it themselves.

3. Are these nodes distant [Y/n]?
"nearby" and "distant" are defined in Section 4.2.2 above. Unlike the single cluster, the default here is "distant".
Note: all core-nodes must be either "nearby" or "distant", you cannot have both for the same partner.

4.3.3 Partner-cluster relationship

By default, migration can occur in both directions: local processes are allowed to migrate to partner-clusters and processes from partners-clusters are allowed to migrate to the local cluster (subject to priorities, see below). As an option, you can allow migration only in one direction (or even disallow migration altogether if all you want is to be able to view the load and status of the other cluster).

4.3.4 Priorities

Each cluster is given a priority: this is a number between 0 and 65535 (0 is not recommended as it is the local cluster's own priority) - the lower it is, the higher the priority. When one or more processes originating from the local cluster, or from partner-clusters of higher priority (lower number), wish to run on a node from our cluster, all processes originating from clusters of a lower priority (higher number) are immediately moved out (evacuated) from this node (often, but not always, back to their home cluster). When you define a new partner-cluster, the default priority is 50.

4.3.5 Priority stabilization

The following option is suitable for situations where the local node is normally occupied with privileged processes (either local processes, processes from your own cluster or processes from more privileged clusters), but repeatedly becomes idle for short periods.

If you know that this is the pattern, you may want to prevent processes from other clusters from arriving during these short gaps when the local node is idle, only to be sent away shortly after. You can define a minimal gap-period (in seconds) once all higher-privileged processes terminated (or left). During that period processes of less-privileged clusters cannot arrive: use "Miscellaneous policies" of "mosconf" to define the length of this period.

4.3.6 Maximum number of guests

The maximal number of simultaneous guest-processes from partner-clusters is limited: the default limit is 8 times the number of local processors, but you can change it using "Miscellaneous policies" of "mosconf" (note that the number of processes from your own cluster is not limited).

4.4 Configuring the queuing system

MOSIX processes can be queued, so as more processors and memory become available, more new jobs are started.

4.4.1 Queuing is an option

Queuing is an option - if it is not needed, there is no need to configure it.

As the system-administrator, it is up to you to set (and enforce) a policy whether or not your users should use queuing, because if some users do not use it, they gain an advantage over the users that do use it. Similarly, you should also set a policy of whether and when users can use priorities other than the default.

4.4.2 Selecting queue-managers

Your first (and often the only) task, is to select queue-manager node(s). Queues are managed on a per cluster-partition basis (or for the whole cluster when it is not partitioned), so you should select one node from your cluster (or from each partition) as the "queue-manager". Any node can be a queue manager (it requires very little resources), but it is best to select node(s) that are most stable and unlikely to come down. When configuring the nodes

in your cluster, "mosconf" will suggest making the first node in your cluster (or in each cluster-partition) the queue-manager: it is up to you to either accept this suggestion, or select another node.

Exception: if you have both 32-bit and 64-bit nodes in the same cluster-partition (or the whole cluster if not partitioned), then you should not accept the suggestion of "mosconf", but assign separate queue-managers for the 32-bit nodes and for the 64-bit nodes. Important: if you choose to mix 32-bit and 64-nodes in the same cluster-partition (the same cluster if not partitioned), then you must have separate configuration files for the 32-bit nodes and the 64-nodes, and run "mosconf" separately for each.

Queue-manager nodes are not supposed to be turned off, but if you do need to take down a queue-manager for more than a few minutes (while the rest of your cluster remains operational), you should first assign another node to take its place as queue-manager. You should be aware that, although no jobs will be lost, rebooting or changing the queue-manager can distort the order of the queue (between jobs that originated from different nodes - the order of jobs that originated from the same node is always preserved).

4.4.3 Advanced

Now for the advanced options:

Default queuing priority per node

The default priority of queued jobs is normally 50 (the lower the better), no matter from which node they originated. If you want jobs that originate from specific nodes to receive a different default priority, you can configure that on a per-node basis (but this requires those nodes to have separate MOSIX configuration files).

User-ID equivalence

It is assumed that the user-ID's are identical in all the nodes of each partition: this allows the user to cancel or modify the priority of their jobs from any node (of the same partition) - not just the one from which they started their job. Otherwise (if user-ID's are not identical), you must configure that fact. Note that in such a case, users will only be able to control their jobs from the node where they started them.

Limiting the number of running jobs

You can fix a maximal number of queued jobs that are allowed to run at any given time - even when there are sufficient resources to run more processes.

Target processes per processor

You can request MOSIX to attempt to run X queued jobs per processor at any given time, instead of the default of 1. The range is 1 to 8.

Provision for urgent jobs

You can assign an additional number of "urgent" jobs (priority-0, the highest possible priority) to run regardless of the available resources and other limitations. If you want to use this option, you first need to discuss with your users which jobs should be considered as

"urgent". It is then your responsibility to ensure that at any given time, running those additional "urgent" jobs will in fact have sufficient memory/swap-space to proceed reasonably. The default is 0 additional jobs and it is highly recommended to keep this number small. Note that if there are more "urgent" jobs in the queue, those above this configured number will still need to wait in the queue for resources, as usual.

Guarantee a number of jobs per-user

You can guarantee a small, minimum number of jobs per user to run, if necessary even out of order and when resources are insufficient. This, for example, allows users to run and get results from short jobs while very long jobs of other users are running.

Along with this option, you usually want to set a memory limit, so jobs that require much memory are not started out of order. Jobs (per user) above this number and jobs that require more memory, will be queued as usual.

Note that when users do not specify the memory requirements of their jobs, (using "mosrun -mmb"), their jobs are considered to require no significant memory, so when using this option you should request your users to always specify their maximum memory-requirement for their queued jobs.

Fair-share policy

The default queue policy is "first-come-first-serve", regardless of which users sent the jobs. If you prefer, you may configure a "fair-share" policy, where jobs (of the same priority) from different users are interleaved, with each user receiving an equal share. If you want to grant different share to certain users, read the section about "Fair-share policy" in the MOSIX manual ("man mosix").

4.5 Configuring the freezing policies

4.5.1 Overview

When too many processes are running on their home node, the risk is that memory will be exhausted, the processes will be swapped out and performance will decline drastically. In the worst case, swap-space may also exhausted and then the Linux kernel will start killing processes. This scenario can happen for many reasons, but the most common one is when another cluster shuts down, forcing a large number of processes to return home simultaneously. The MOSIX solution is to freeze such returning processes (and others), so they do not consume precious memory, then restart them again later when more resources become available.

Please note that this section only deals with local processes: guest processes are not subject to freezing because at any time when the load rises, they can instead simply migrate back to their home-nodes (or elsewhere).

Every process can be frozen, but not every process can be frozen and restarted safely without ill side effects. For example, if even one among communicating parallel processes are frozen, all the others also become blocked. Other examples of processes that should not be frozen, are processes that can time-out or provide external services (such as over the web).

While both the user and the system-administrator can freeze any MOSIX process manually at any time (using "migrate pid freeze"), below we shall discuss how to set up a policy for automatic freezing to handle different scenarios of process-flooding.

4.5.2 Process classes

The freezing policies are based on process-classes: Each MOSIX process can be assigned to a class, using the "mosrun -Gclass" option. Processes that do not use this option are of class 0 and cannot migrate outside their cluster, hence the main cause for flooding is eliminated. Common MOSIX processes are run with "mosrun -G", which brings them into the default, class 1.

When you install MOSIX for the first time, you get a default freezing policy for class 1 and for no other class. This is often sufficient, but the discussion below describes how to optimize and tune it for your specific needs.

As the need arises, you should identify with your users different classes of applications that require different automatic-freezing policies. Example 1: if some of your users run parallel jobs that should not be frozen, you can assign for them a specific class-number (for example 20), and tell them: "in this case, use 'mosrun -G20'", then as the system-administrator make sure that no freezing-policy is defined for class #20. Example 2: if a certain user has long batch jobs with large memory demands, you can assign a different class number (for example 8), and tell them: "for those batch jobs, use 'mosrun -G8'", then as the system-administrator create a freezing policy for class #8 that will start freezing processes of this class earlier (when the load is still relatively lower) than processes of other classes.

4.5.3 Freezing-policy details

In this section, the term "load" refers to the local node.

The policy for each class that you want to auto-freeze consists of:

- The "Red-Mark": when the load reaches above this level, processes (of the given class) will start to be frozen until the load drops below this mark.
- The "Blue-Mark": when the load drops below this level, processes start to un-freeze. Obviously the "Blue-Mark" must be significantly less than the "Red-Mark".
- "Home-Mark": when the load is at this level or above and processes are evacuated from other clusters back to their home-node, they are frozen on arrival (without consuming a significant amount of memory while migrating).
- "Cluster-Mark": when the load is at this level or above and processes from this home-node are evacuated from other clusters back to this cluster, they are instead brought frozen to their home-node.
- Whether the load for the above 4 load marks ("Red", "Blue", "Home", "Cluster") is expressed in units of processes or in standardized MOSIX load: The number of processes is more natural and easier to understand, but the MOSIX load is more accurate and takes into account the number and speed of the processors: roughly, a MOSIX load unit is the number of processes divided by the number of processors (CPUs) and by their speed relative to a "standard" processor (currently Intel Core at

3GHz). Using the MOSIX standardized load is recommended in clusters with nodes of different types.

- Whether to keep a given, small number of processes from this class running (not frozen) at any time despite the load.
- Whether to allow only a maximum number of processes from this class to run (that run on their home-node - not counting migrated processes), freezing any excess processes even when the load is low.
- Time-slice for switching between frozen processes: whenever some processes of a given class are frozen and others are not, MOSIX rotates the processes by allowing running processes a given number of minutes to run, then freezing them to allow another process to run instead.
- Policy for killing processes that failed to freeze, expressed as memory-size in MegaBytes: in the event that freezing fails (due to insufficient disk-space), processes that require less memory are kept alive (and in memory) while process requiring the given amount of memory or more, are killed. Setting this value to 0, causes all processes of this class to be killed when freezing fails. Setting it to a very high value (like 1000000 MegaBytes) keeps all processes alive.

The default freezing policy for class #1, that you get when installing MOSIX for the first time, is:

RED-MARK	= 6.0 MOSIX standardized load units
BLUE-MARK	= 4.0 MOSIX standardized load units
HOME-MARK	= 0.0 (eg. always freeze evacuated processes)
CLUSTER-MARK	= 0.0 (eg. always freeze evacuated processes)
MINIMUM-UNFROZEN	= 1 (process)
MAXIMUM-RUNNING	= unlimited
TIME-SLICE	= 20 minutes
KILLING-POLICY	= always

4.5.4 Disk-space for freezing

Next, you need inform MOSIX where to store the memory-image of frozen processes, which is configured as directory-name(s): the exact directory name is not so important (because the memory-image files are unlinked as soon as they are created), except that it specifies particular disk partition(s).

The default is that all freeze-image files are created in the directory (or symbolic-link) "freeze" (please make sure that it exists, or freezing will always fail). Instead, you can select a different directory(/disk-partition) or up to 10 different directories.

If you have more than one physical disk, specifying directories on different disks can help speeding up freezing by writing the memory-image of different processes in parallel to different disks. This can be important when many large processes arrive simultaneously (such as from other clusters that are being shut-down).

You can also specify a "probability" per directory (eg. per disk): This defines the relative chance that a freezing process will use that directory for freezing. The default

probability is 1 (unlike in statistics, probabilities do not need to add up to 1.0 or to any particular value).

When freezing to a particular directory (eg. disk-partition) fails (due to insufficient space), MOSIX will try to use the other freezing directories instead, thus freezing fails only when all directories are full. You can specify a directory with probability 0, which means that it will be used only as a last resort (it is useful when you have faster and slower disks).

4.5.5 Ownership of freezing-files

Freezing memory-image files are usually created with Super-User ("root") privileges. If you do your freezing via NFS (it is slow, but sometimes you simply do not have a local disk), some NFS servers do not allow access to "root": if so, you can select a different user-name, so that memory-image files will be created under its privileges.

4.6 Configuring the processor-speed

4.6.1 Standard processor

MOSIX defines that the speed of a "standard processor" is 10000 units. In the latest MOSIX releases, the "standard processor" is an Intel Core (Duo). The speed of other processors is measured relative to the "standard processor", based on their model and clock-frequency: the faster the processor, the higher the speed.

4.6.2 No one benchmark for all

Extensive tests have revealed that there is no linear way to rate the speed of the current processors in the market. For example, even with the same CPU clock-frequency, some CPUs perform floating-point operations faster than the rest, other CPUs perform integer arithmetic faster than the rest, while again other CPUs perform single-word memory access faster than the rest.

No single benchmark is therefore possible that can determine a good-for-all processor-speed.

The processor-speed is an important factor in deciding whether and where to migrate processes, but due to the absence of a linear benchmark, MOSIX needs to use a simplistic criteria, that is essentially based on the processor's clock-frequency, with very few adjustments.

4.6.3 Setting the CPU speed manually

If you, as the system-administrator, find that most or all of the work of your users involves certain applications that work better than average on some of your nodes, you can quantify it by overriding the MOSIX estimates and forcing a more accurate speed (using "Miscellaneous policies" of "mosconf").

Also, some installations prefer to keep some of their computers mostly idle (for example, in order to give users a better interactive response). If you have nodes in this category, you can create this effect by forcing their speed down to a very low value (such as 1000, compared with 10000 of the standard processor).

Chapter 5

Storage allocation

5.1 Swap space

As on a single computer, you are responsible to make sure that there is sufficient swap-space to accommodate the memory demands of all the processes of your users: the fact that processes can migrate does not preclude the possibility of them arriving at times back to their home-node for a variety of reasons: please consider the worst-case and have sufficient swap-space for all of them.

You do not need to take into account batch jobs that are sent to other nodes in your cluster.

5.2 MOSIX files

During the course of its operation, MOSIX creates and maintains a number of small files in the directory "etc/mosix/var". When there is no disk-space to create those files, MOSIX operation (especially load-balancing and queuing) will be disrupted.

When MOSIX is installed for the first time (or when upgrading from an older MOSIX version that had no "etc/mosix/var"), you are asked whether you prefer "etc/mosix/var" to be a regular directory or a symbolic link to "var/mosix". However, you can change it later.

Normally the disk-space in the root partition is never exhausted, so it is best to let "etc/mosix/var" be a regular directory, but some diskless cluster installations do not allow modifications within "etc": if this is the case, then "etc/mosix/var" should be a symbolic link to a directory on another partition which is writeable and have the least chance of becoming full. This directory should be owned by "root", with "chmod 755" permissions and contain a sub-directory "grid/".

5.3 Freezing space

MOSIX processes can be temporarily frozen for a variety of reasons: it could be manually using the command: "migrate pid freeze" (which as the Super-User you can also use to freeze any user's processes), or automatically as the load increases, or when evacuated from another cluster. In particular, when another cluster(s) shuts down, many processes can be evacuated back home and frozen simultaneously.

Frozen processes keep their memory-contents on disk, so they can release their main-memory image. By default, if a process fails to write its memory- contents to disk because there is insufficient space, that process is killed: this is done in order to save the system from filling up the memory and swap-space, which causes Linux to either be deadlocked or start killing processes at random.

As the system-administrator, you want to keep the killing of frozen processes only as the last resort: use either or both of the following two methods to achieve that:

1. Allocate freezing directory(s) on disk partitions with sufficient free disk-space: freezing is by default to the "freeze" directory (or symbolic-link), but you can re-configure it to any number of freezing directories.

2. Configure each class of processes that are automatically frozen so processes of that class are not killed when freeze-space is unavailable unless their memory-size is extremely big (specify that threshold in MegaBytes - a value such as 1000000MB would prevent killing altogether).

5.4 Private-file space

MOSIX users have the option of creating private files that migrate with their processes. If the files are small (up to 10MB per process) they are kept in memory - otherwise they require backing storage on disk and as the system-administrator it is your responsibility to allocate sufficient disk-space for that.

You can set up to 3 different directories (therefore up to 3 disk partitions) for the private files of local processes ; guest processes from the same cluster ; and guest processes from other clusters. For each of those you can also define a per-process quota.

When a guest process fails to find disk-space for its private files, it will transparently migrate back to its home-node, where it is more likely to find the needed space; but when a local process fails to find disk-space, it has nowhere else to go, so its "write()" system-call will fail, which is likely to disrupt the program.

Efforts should therefore be made to protect local processes from the risk of finding that all the disk-space for their private files was already taken by others: the best way to do it is to allocate a separate partition at least for local processes (by default, space for private files is allocated in "private" for both local and guest processes).

For the same reason, local processes should usually be given higher quotas than guest processes (the default quotas are 5GB for local processes, 2GB for guests from the cluster and 1GB for guests from other clusters).

Chapter 6

Managing jobs

As the system administrator you can make use of the following tools:

6.1 Monitoring (mon)

mon ("man mon"): monitor the load, memory-use and other parameters of your MOSIX cluster or even the whole multi-cluster.

6.2 Listing MOSIX processes (mosps)

mosps ("man mosps"): view information about current MOSIX processes. In particular, "mosps a" shows all users, and "mosps -V" shows guest processes. Please avoid using "ps" because each MOSIX process has a shadow son process that "ps" will show, but you should only access the parent, as shown by "mosps".

6.3 Controlling running processes (migrate)

migrate ("man migrate"): you can manually migrate the processes of all users - send them away; bring them back home; move them to other nodes; freeze; or unfreeze (continue) them, overriding the MOSIX system decisions as well as the placement preferences of users. Even though as the Super-User you can technically do so, you should never kill (signal) guest processes. Instead, if you find guest processes that you don't want running on one of your nodes, you can use "migrate" to send them away (to their home-node or to any other node).

6.4 Viewing and controlling queued processes (mosq)

mosq ("man mosq"): list the jobs waiting on the MOSIX queue and possibly modify their priority or even start them running out of the queue.

6.5 Controlling the MOSIX node (mosctl)

mosctl ("man mosctl"): This utility provides a variety of functions. The most important are:

- "mosctl stay" - prevent automatic migration away from this node.
- ("mosctl nostay" to undo).
- "mosctl lstay" - prevent automatic migration of local processes away from this node.
- ("mosctl nolstay" to undoa.)
- "mosctl block" - do not allow further migrations into this node.
- ("mosctl noblock" to undo).
- "mosctl bring" - bring back all processes whose home-node is on this node. You would usually combine it with using "mosctl lstay" first.
- "mosctl expel" - send away all guest processes. You would usually combine it with using "mosctl block" first.
- "mosctl shutdown" - disconnect this node from the cluster. All processes are brought back home, guest processes expelled and the node is isolated from its cluster (and the multi-cluster).
- "mosctl isolate" - isolate the node from the multi-cluster (but not from its cluster)
- ("mosctl rejoin" to undo)
- "mosctl cngpri partner newpri" - modify the guest-priority of another cluster in the multi-cluster (the lower the better).
- "mosctl localstatus" - check the health of MOSIX on this node.

Chapter 7

Security

7.1 Abuse by gaining control of a node

A hacker that gains Super-User access on any node of any cluster could intentionally use MOSIX to gain control of the rest of the cluster and the multi-cluster. Therefore, before joining into a MOSIX multi-cluster, trust needs to be established among the owners (Super-Users) of all clusters involved (but not necessarily among ordinary users). In particular, system-administrators within a MOSIX multi-cluster need to trust that all the other system-administrators have their computers well protected against theft of Super-User rights.

7.2 Abuse by connecting hostile computers

Another risk is of hostile computers gaining physical access to the internal cluster's network and masquerading the IP address of a friendly computer, thus pretending to be part of the MOSIX cluster/multi-cluster. Normally within a hardware cluster, as well as within a well-secured organization, the networking hardware (switches and routers) prevents this, but you should especially watch out for exposed Ethernet sockets (or wireless connections) where unauthorized users can plug their laptop computers into the internal network. Obviously, you must trust that the other system-administrators in your multi-cluster maintain a similar level of protection from such attacks.

7.3 Multi-cluster password

Part of configuring MOSIX ("Authentication" of "mosconf") is selecting a multi-cluster-protection key (password), which is shared by the entire multi-cluster. Please make this key highly-secure - a competent hacker that obtains it can gain control over your computers and thereby the entire multi-cluster.

7.4 Organizational Multi-cluster Grid

This level of security is usually only achievable within the same organization, hence we use the term "organizational Multi-cluster Grid", but if it can exist between different organizations, nothing else prevents them from sharing a MOSIX multi-cluster.

7.5 Batch password

If you intend to run MOSIX batch-jobs, you also need to select batch keys: a "client-key" and a "server-key". These keys should be different in each cluster-partition. A node will only provide batch-service to nodes whose client-key is identical to its server-key (and are both present). In the usual case, when you want to allow all your nodes to be both batch-clients and batch-servers, set the same key as both the client-key and the server-key. If, however, you want some nodes to only be clients and others to only be servers, set the client-key on the clients identical to the server-key on the servers, and use no server-key on the clients and no client-key on the servers. Again, please make this key highly-secure.

Part II
Manuals

Chapter 8

Manuals

The manuals in this chapter are provided for general information. Users of MOSIX are advised to use the on-line manuals that are provided with their specific distribution.

The manuals are arranged in 3 sets:

For Users

mosrun - Running MOSIX programs
mosps - List information about MOSIX processes
mosq - MOSIX queue control
mospipe - Run pipelined jobs efficiently using Direct Communication
moskillall - Kill or signal all your MOSIX processes
bestnode - Select the best node to run on
migrate - Manual control of running MOSIX processes
mon - MOSIX monitor
testload - MOSIX test program

For Programmers

mosix - sharing the power of clusters and multi-clusters
direct communication - migratable sockets between MOSIX processes

For Administrators

mosctl - Miscellaneous MOSIX functions
setpe - Configure the local cluster
topology - incorporating networking costs in MOSIX

NAME

MOSRUN - Running MOSIX programs

SYNOPSIS

```
mosrun [location_options] [program_options] program [args] ...
mosrun -S{maxjobs} [location_options] [program_options] {commands-file}
       [,{failed-file}]
mosrun -R{filename} [-O{fd=filename}[,{fd2=fn2}]]... [location_options]
mosrun -I{filename}
mosenv { same-arguments-as-mosrun }
native program [args]...

       Location options:

            [-r{hostname} |-{a.b.c.d} |-{n} |-h |-b |
            -jID1-ID2[,ID3-ID4]... }][-G[{class}]][-F][-L][-l]
            [-D{DD:HH:MM}][-A{minutes}][-N{max}][-{q|Q}[{pri}]]
            [-P{parallel_processes}][-J{JobID}]

       Program Options:

            [-m{mb}][-d {0-10000}][-c][-n][-z][-e][-u][-w][-t][-T]
            [-E[/{cwd}]][-M[/{cwd}]][-i][-C{filename}][-X{/directory}]...
```

DESCRIPTION

Mosrun runs a program under the MOSIX discipline: this means that programs activated by mosrun can potentially migrate to other nodes within the cluster or multi-cluster grid (see mosix(7)): programs that are not started by mosrun, run in "native" Linux mode and cannot migrate.

Once running under MOSIX, the program and all its child-processes remain under the MOSIX discipline, with the exception of the native utility, that allows programs (mainly shells) that already run under mosrun to spawn children that run in native Linux mode.

The following arguments may be used to specify the program's initial assignment:

-r{hostname}	on the given host
-{a.b.c.d}	on the given IP address
-{n}	on the given node-number
-h	on the home-node
-b	the program attempts to select the best node
-jID1-ID2[,ID3-ID4]...	select at random from the given list of hosts, IP's and/or node numbers.

When none of the above arguments is used, the program will start wherever its parent process is running.

The -F flag states that mosrun should start the program somewhere else, even if the requested node (above) is not available.

The `-L` flag states that the program should not be allowed to migrate automatically. It may still be migrated manually or when situations arise that do not allow it to continue running where it is.

The `-l` flag negates the `-L` flag and allows the program to migrate automatically: this is useful when `-L` was already applied to the program (usually a shell) that calls `mosrun`.

The `-G` argument states that the program should be allowed to migrate to nodes in other partitions and clusters within the grid, rather than only within the local partition. This argument may be followed by a positive integer, `-G[{class}]` that specify the program's class: when that number is omitted, the class of the program is assumed to be 1. It is also possible to specify `-G0`, meaning that the program may not migrate outside the local partition (this is useful when `-G` was already applied the calling program).

The `-D{timespec}` allows the user to provide an estimate on how long their job should run. MOSIX does not use this information - it is provided in order to help `mosps(1)` keep track of processes. `timespec` can be specified in any of the following formats (DD/HH/MM are numeric for days, hours and minutes respectively): `DD:HH:MM`; `HH:MM`; `DDd`; `HHh`; `MMm`; `DDdHHhMMm`; `DDdHHh`; `DDdMMm`; `HHhMMm`. Periods when the process is frozen are automatically added to that estimate.

The `-m{mb}` argument states that the program requires a certain amount of memory (in Megabytes) and should not run with less. This has the effect of:

1. Combined with the `-b` flag, the program will only consider to start running on nodes with available memory of at least `{mb}` Megabytes: the program will not even start until at least one such node is found.
2. The program will not automatically migrate to nodes with less than `{mb}` Megabytes free memory (with the exception of the home node, when the program must move back home).
3. The queuing system (see below) will take the program's memory requirements into account when deciding which and how many jobs to allow to run at any point in time.

Most system-calls are supported by MOSIX, but a few are not (such as mapping shared memory or cloning - see the "LIMITATIONS" section below). By default, when a program under `mosrun` encounters an unsupported system-call, it is killed. The `-e` flag, however, allows the program to continue and behave as follows:

1. `mmap(2)` with (flags & MAP_SHARED) - but !(prot & PROT_WRITE), replaces the MAP_SHARED with MAP_PRIVATE (this combination seems unusual or even faulty, but is unnecessarily used within some Linux libraries).

2. all other unsupported system-call return -1 and "errno" is set to ENOSYS.

The `-w` flag is the same as `-e`, but it also causes `mosrun` to print an error message to the standard-error when an unsupported system-call is encountered. The `-u` flag returns to the default of killing the process.

System calls and I/O operations are monitored and taken into account in automatic migration considerations, tending to pull processes towards their home-nodes. The `-c` flag tells `mosrun` not to take system calls and I/O operations in the migration considerations. The `-n` flag reverts to taking them into account.

Even when running elsewhere, programs running under MOSIX obtain the results of the `gettimeofday(2)` system-call from their home-nodes. The `-t` flag tells mosrun to take the time from the local node (where the process currently runs), thus reducing the communication overhead with the home-node. Note that this can be a problem when the clocks are not synchronized. The `-T` flag reverses the effect of `-t`.

The `-d{decay}` argument, where decay is an integer between 0 and 10000, sets the rate of decay of process-statistics as a fraction of 10000 per second (see `mosix(7)`).

The `-z` flag states that the program's arguments begin at argument #0 - otherwise, the arguments (if any) are assumed to begin at argument #1 and argument #0 is assumed to be identical to the program-name.

`mosrun` can send batch jobs to other nodes of the local cluster-partition. There are two types of batch jobs: those produced by the `-E` argument are native Linux jobs, while those produced by the `-M` argument are MOSIX jobs - but possibly with a different home-node.

Batch jobs are executed 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 after forking, signals are delivered to the whole process-group rather than just the parent; they cannot communicate with other processes on the local node using pipes and sockets (other than standard input/output/error), semaphores, messages, etc. and can only receive signals, but not send them. The main advantage of batch jobs is that they save time by not needing to refer to the home-node to perform system-calls, so temporary files for example, can be created on the node where they start, preventing the calling node from becoming a bottleneck. This approach is recommended for programs that perform a significant amount of I/O.

Batch jobs use the path of the current directory as their current-directory on the other node. It is possible to override that path by specifying a different directory in the `-E{/cwd}` or `-M{/cwd}` arguments.

The `-i` flag states that all the standard-input of a batch job is for its exclusive use: it is especially recommended when the input of a batch job is redirected from a file. Programs that use `poll(2)` or `select(2)` to check for input before reading from their standard-input can only work in batch mode with the `-i` flag. This flag can also improve the performance. An example when the `-i` flag cannot be used, is when an interactive shell places a batch job in the background (because typed input that is intended for the shell may go to the batch job instead).

MOSIX-specific arguments (`-G`, `-F`, `-L`, `-l`, `-m`, `-d`, `-c`, `-n`, `-e`, `-u`, `-t`, `-T`, `-A`, `-N`, `-C`), do not apply to native Linux batch jobs that are started with the `-E` argument, but they do apply to jobs started with the `-M` argument.

Permission is required from the other node to send batch jobs there (see `mosix(7)` for more information).

The following arguments: `-G`, `-L`, `-l`, `-m`, `-d`, `-c`, `-n`, `-e`, `-u`, `-t`, `-T` are inherited by child processes: see however in `mosix(7)` how those can be changed at run time from within the program.

The variant `mosenv` is used to circumvent the loss of certain environment variables by the GLIBC library due to the fact that `mosrun` is a "setuid" program: if your program relies on the settings of dynamic-linking environment variables (such as LD_LIBRARY_PATH) or `malloc(3)` debugging (MALLOC_CHECK_), use `mosenv` instead of `mosrun`.

CHECKPOINTS

Most CPU-intensive processes running under `mosrun` can be checkpointed: this means that an image of those processes is saved to a file, and when necessary, the process can later recover itself from that file and continue to run from that point.

For successful checkpoint and recovery, the process must not depend heavily on its Linux environment. Specifically, the following processes cannot be checkpointed at all:

1. Processes with setuid/setgid privileges (for security reasons).
2. Processes with open pipes or sockets.

The following processes can be checkpointed, but may not run correctly after being recovered:

1. Processes that rely on process-ID's of themselves or other processes (parent, sons, etc.).
2. Processes that rely on parent-child relations (e.g. use `wait(2)`, Terminal job-control, etc.).
3. Processes that coordinate their input/output with other running processes.
4. Processes that rely on timers and alarms.
5. Processes that cannot afford to lose signals.
6. Processes that use system-V IPC (semaphores and messages).

The `-C{filename}` argument specifies where to save checkpoints: when a new checkpoint is saved, that file-name is given a consecutive numeric extension (unless it already has one). For example, if the argument `-Cmysave` is given, then the first checkpoint will be saved to `mysave.1`, the second to `mysave.2`, etc., and if the argument `-Csave.4` is given, then the first checkpoint will be saved to `save.4`, the second to `save.5`, etc. If the `-C` argument is not provided, then the checkpoints will be saved to the default: `ckpt.{pid}.1`, `ckpt.{pid}.2` ... The `-C` argument is NOT inherited by child processes.

The `-N{max}` argument specifies the maximum number of checkpoints to produce before recycling the checkpoint versions. This is mainly needed in order to save disk space. For example, when running with the arguments: `-Csave.4 -N3`, checkpoints will be saved in `save.4`, `save.5`, `save.6`, `save.4`, `save.5`, `save.6`, `save.4 ...`
The `-N0` argument returns to the default of unlimited checkpoints; an argument of `-N1` is risky, because if there is a crash just at the time when a backup is taken, there could be no remaining valid checkpoint file. Similarly, if the process can possibly have open pipe(s) or socket(s) at the time a checkpoint is taken, a checkpoint file will be created and counted - but containing just an error message, hence this argument should have a large-enough value to accommodate this possibility. The `-N` argument is NOT inherited by child processes.

Checkpoints can be triggered by the program itself, by a manual request (see `migrate(1)`) and/or at regular time intervals. The `-A{minutes}` argument requests that checkpoints be automatically taken every given number of minutes. Note that if the process is within a blocking system-call (such as reading from a terminal) when the time for a checkpoint comes, the checkpoint will be delayed until after the completion of that system call. Also, when the process is frozen, it will not produce a checkpoint until unfrozen. The `-A` argument is NOT inherited by child processes.

With the `-R{filename}` argument, `mosrun` recovers and continue to run the process from its saved checkpoint file. Program options are not permitted with `-R`, since their values are recovered from the checkpoint file.

It is not always possible (or desirable) for a recovered program to continue to use the same files that were open at the time of checkpoint: `mosrun -I{filename}` inspects a checkpoint file and lists the open files, along with their modes, flags and offsets, then the `-O` argument allows the recovered program to continue using different files. Files specified using this option, will be opened (or created) with the previous modes, flags and offsets. The format of this argument is usually a comma-separated list of file-descriptor integers, followed by a '=' sign and a file-name. For example:

`-O1=oldstdout,2=oldstderr,5=tmpfile`, but in case one or more file-names contain a comma, it is optional to begin the argument with a different separator, for example: `-O@1=file,with,commas@2=oldstderr@5=tmpfile`.

In the absence of the `-O` argument, regular files and directories are re-opened with the previous modes, flags and offsets.

Files that were already unlinked at the time of checkpoint, are assumed to be temporary files belonging to the process, and are also saved and recovered along with the process (an exception is if an unlinked file was opened for write-only). Unpredictable results may occur if such files are used to communicate with other processes.

As for special files (most commonly the user's terminal, used as standard input, output or error) that were open at the time of checkpoint - if `mosrun` is called with their file-descriptors open, then the existing open files are used (and their modes, flags and offsets are not modified). Special files that are neither specified in the `-O` argument, nor open when calling `mosrun`, are replaced with `/dev/null`.

While a checkpoint is being taken, the partially-written checkpoint file has no permissions (chmod 0). When the checkpoint is complete, its mode is changed to 0400 (read-only).

QUEUING

MOSIX incorporates a queuing system that allow users to submit a number of jobs that will be scheduled to run when resources are available. Although the number of queued jobs can be large, it is limited by the number of Linux processes (about 30000 for all users): to queue more jobs, see the "RUNNING MULTIPLE JOBS" section below.

The queuing system is common to each cluster-partition and using it is optional. It is recommended that a policy is decided where either all the users of a cluster use it, or all do not. Queued jobs can also be controlled using `mosq(1)`.

The `-q` argument causes the whole `mosrun` command to be queued and postponed until the queuing system launch it.

The letter `q` may optionally be followed by a non-negative integer, specifying the job's priority - the lower the number, the higher the priority (in the absence of this number, a pre-configured, per-node default of 50 is used, unless configured otherwise by the system-administrator).

Queued programs are shown `mosps(1)` and `ps(1)` as `"mosqueue"`.

The `-Q` argument is similar to `-q`, except that if MOSIX is stopped (or restarted) while the program is queued, or if the queuing system attempts to abort the job (see `mosq(1)`), with `-q` the program will be killed, while with `-Q` it will bypass the queuing system and begin running.

The `-P{parallel_processes}` argument informs the queuing system that the job may split into a given number of parallel processes (hence more resources must be reserved for it).

The `-J{JobID}` argument allows bundling of several instances of `mosrun` with a single "job" ID for easy identification and manipulation (the concept of what a "job" means is left for each user to define). "Jobs" can then be handled collectively by `mosq(1)`, `migrate(1)`, `mosps(1)` and `moskillall(1)`.

Job-ID's can be either a non-negative integer or a token from the file `$HOME/.jobids`: if this file exists, each line in it contains a number (JobID) followed by a token that can be used as a synonym to that JobID. The default JobID is 0.

Job ID's are inherited by child processes.

This argument is ignored for batch jobs originating from other nodes.

RUNNING MULTIPLE JOBS

The `-S{maxjobs}` option runs under `mosrun` multiple command-lines from the file specified by `commands-file`, each with the given `mosrun` arguments.

This option is commonly used to run the same program with many different sets of arguments. For example, the contents of `commands-file` could be:

```
my_program -a1 < ifile1 > output1
my_program -a2 < ifile2 > output2
my_program -a3 < ifile3 > output3
```

Command-lines are started in the order they appear in `commands-file`. While the number of command-lines is unlimited, `mosrun` will run concurrently up to `maxjobs` (1-30000) command-lines at any given time: when any command-line terminates, a new command-line is started.

Command lines are interpreted by the standard shell (`bash(1)`). Please note that `bash` has the property that when redirection is used, it spawns a son-process to run the command: if the number of processes is an issue, it is recommended to prepend the keyword `exec` before each command line that uses redirection. For example:

```
exec my_program -a1 < ifile1 > output1
exec my_program -a2 < ifile2 > output2
exec my_program -a3 < ifile3 > output3
```

The exit status of `mosrun -S{maxjobs}` is the number of command-lines that failed (255 if more than 255 command-lines failed).

As a further option, the `commands-file` argument can be followed by a comma and another filename: `commands-file,failed-commands`. `Mosrun` will create the second file and write to it the list of all the commands (if any) that failed (this provides an easy way to re-run only those commands that failed).

The `-S{maxjobs}` option combines well with the queuing system (the `-q` argument), setting an absolute upper limit on the number of simultaneous jobs whereas the number of jobs allowed to run by the queuing system depends on the available multi-cluster resources. With this combination, to prevent an unnecessary and excessive number of waiting processes, no more than 10 jobs will be queued at any given moment.

PRIVATE TEMPORARY FILES

Normally, all files are created on the home-node and all file-operations are performed there. This is important because programs often share files, but can be costly: many programs use temporary files which they never share - they create those files as secondary-memory and discard them when they terminate. It is best to migrate such files with the process rather than keep them in the home-node.

The `-X {/directory}` argument tells `Mosrun` that a given directory is only used for private temporary files: all files that the program creates in this directory are kept with the process that created them and migrate with it.

The `-X` argument may be repeated, specifying up to 10 private temporary directories. The directories must start with '/'; can be up to 256 characters long; cannot include ".."; and for security reasons cannot be within "/etc", "/proc", "/sys" or "/dev".

Only regular files are permitted within private temporary directories: no sub-directories, links, symbolic-links or special files are allowed (except that sub-directories can be specified by an extra `-X` argument).

Private temporary file names must begin with '/' (no relative pathnames) and contain no ".." components. The only file operations currently supported for private temporary files are: open, creat, lseek, read, write, close, chmod, fchmod, unlink, truncate, ftruncate, access, stat.

File-access permissions on private temporary files are provided for compatibility, but are not enforced: the `stat(2)` system-call returns 0 in `st_uid` and `st_gid`. `stat(2)` also returns the file-modification times according to the node where the process was running when making the last change to the file.

The per-process maximum total size of all private temporary files is set by the system-administrator. Different maximum values can be imposed when running on the home-node, in the local cluster and on other clusters in the grid - exceeding this maximum will cause a process to migrate back to its home-node.

ALTERNATIVE FREEZING SPACE

MOSIX processes can sometimes be frozen (you can freeze your processes manually and the system-administrator usually sets an automatic-freezing policy - See `mosix(7)`).

The memory-image of frozen processes is saved to disk. Normally the system-administrator determines where on disk to store your frozen processes, but you can override this default and set your own freezing-space. One possible reason to do so is to ensure that your processes (or some of them) have sufficient freezing space regardless of what other users do. Another possible reason is to protect other users if you believe that your processes (or some of them) may require so much memory that they could disturb other users.

Setting your own freezing space can be done either by setting the environment-variable `FREEZE_DIR` to an alternative directory (starting with '/'); or if you wish to specify more than one freeze-directory, by creating a file: `$HOME/.freeze_dirs` where each line contains a directory-name starting with '/'. For more details, read about "lines starting with '/'" within the section about configuring `/etc/mosix/freeze.conf` in the `mosix(7)` manual.

You must have write-access to the your alterantive freeze-directory(s). The space available in alternative freeze-directories is subject to possible disk quotas.

RECURSIVE MOSRUN

It is possible to run `mosrun` within an already-running `mosrun`: this can happen, for example, when a shell-script that contains calls to `mosrun` is itself run by `mosrun`, or when running `mosrun make` with a `Makefile` that contains calls to `mosrun`.

The following arguments (and only those) of the outer `mosrun` will be preserved by the inner `mosrun` (unless the inner `mosrun` explicitly requests otherwise): `-c`, `-d`, `-e`, `-J`, `-G`, `-L`, `-l`, `-m`, `-n`, `-T`, `-t`, `-u`, `-w`.

LIMITATIONS

32-bit processes must have a 32-bit home-node (but they can be assigned or migrated to 64-bit nodes). Attempts to execute a 32-bit binary under a 64-bit home-node will turn the process into a native Linux process (and if that process has open private-temporary-files or uses direct communication, it will be killed). Obviously, 64-bit processes cannot run on 32-bit nodes.

Batch jobs from 64-bit nodes are currently not permitted to run on 32-bit nodes.

Some system-calls are not supported by `mosrun`, including system-calls that are tightly connected to resources of the local node or intended for system-administration. These are:

acct, add_key, adjtimex, afs_syscall(x86_64), alloc_hugepages(i386), bdflush, capget, capset, chroot, clock_getres, clock_nanosleep, clock_settime, create_module(x86_64), delete_module, epoll_create, epoll_ctl, epoll_pwait, epoll_wait, eventfd, free_hugepages(i386), futex, get_kernel_syms(x86_64), get_mempolicy, get_robust_list, getcpu, getpmsg(x86_64), init_module, inotify_add_watch, inotify_init, inotify_rm_watch, io_cancel, io_destroy, io_getevents, io_setup, io_submit, ioperm, iopl, ioprio_get, ioprio_set, kexec_load(x86_64), keyctl, lookup_dcookie, madvise, mbind, migrate_pages, mlock, mlockall, move_pages, mq_getsetattr, mq_notify, mq_open, mq_timedreceive, mq_timedsend, mq_unlink, munlock, munlockall, nfsservctl, personality, pivot_root, ptrace, quotactl, reboot, remap_file_pages, request_key, rt_sigqueueinfo, rt_sigtimedwait, sched_get_priority_max, sched_get_priority_min, sched_getaffinity, sched_getparam, sched_getscheduler, sched_rr_get_interval, sched_setaffinity, sched_setparam, sched_setscheduler, security(x86_64), set_mempolicy, setdomainname, sethostname, set_robust_list, settimeofday, shmat, signalfd, swapoff, swapon, syslog, timer_create, timer_delete, timer_getoverrun, timer_gettime, timer_settime, timerfd, timerfd_gettime, timerfd_settime, tuxcall(x86_64), unshare, uselib, vm86(i386), vmsplice, waitid.

In addition, `mosrun` supports only limited options for the following system-calls:

`clone` The only permitted flags are CLONE_CHILD_SETTID, CLONE_PARENT_SETTID, CLONE_CHILD_CLEARTID, and the combination CLONE_VFORK|CLONE_VM; the child-termination signal must be SIGCLD and the stack-pointer (child_stack) must be NULL.

`getpriority`
may refer only to the calling process.

`ioctl` The following requests are not supported: TIOCSERGSTRUCT, TIOCSERGETMULTI, TIOCSERSETMULTI, SIOCSIFFLAGS, SIOCSIFMETRIC, SIOCSIFMTU, SIOCSIFMAP, SIOCSIFHWADDR, SIOCSIFSLAVE, SIOCADDMULTI, SIOCDELMULTI, SIOCSIFHWBROADCAST, SIOCSIFTXQLEN, SIOCSMIIREG, SIOCBONDENSLAVE, SIOCBONDRELEASE, SIOCBONDSETHWADDR, SIOCBONDSLAVEINFOQUERY, SIOCBONDINFOQUERY, SIOCBONDCHANGEACTIVE, SIOCBRADDIF, SIOCBRDELIF. Non-standard requests that are defined in drivers that are not part of the standard Linux kernel are also likely to not be supported.

`ipc` the following SYSV-IPC calls are not supported: `shmat`, `semtimedop`, new-version calls (bit 16 set in call-number).

`mmap` MAP_SHARED and mapping of special-character devices are not permitted.

`prctl` only the PR_SET_DEATHSIG and PR_GET_DEATHSIG options are supported.

`setpriority`
may refer only to the calling process.

`setrlimit`
it is not permitted to modify the maximum number of open files (RLIMIT_NOFILES): `mosrun` fixes this limit at 1024.

Programs that fail to run because they call an unsupported system-call can still run in batch mode (`'mosrun -E'`).

Users are not permitted to send the SIGSTOP signal to MOSIX processes: SIGTSTP should be used instead (and `moskillall(1)` changes SIGSTOP to SIGTSTP).

SEE ALSO

`migrate`(1), `mosq`(1), `moskillall`(1), `mosps`(1), `direct_communication`(7), `mosix`(7).

NAME

MOSPS - List information about MOSIX processes

SYNOPSIS

mosps [subset of ps(1) options][-I] [-h] [-M] [-L] [-O] [-n] [-P] [-V] [-D] [-S] [-J{JobID}]

Supported ps(1) options:

1. single-letter options:	TUacefgjlmnptuwx
2. single-letters preceded by '-':	-AGHNTUadefgjlmptuwx

DESCRIPTION

Mosps lists MOSIX processes in "ps" style, emphasizing MOSIX-related information, see the ps(1) manual about the standard options of ps. Since some of the information in the ps(1) manual is irrelevant, mosps does not display the following fields: %CPU, %MEM, ADDR, C, F, PRI, RSS, S, STAT, STIME, SZ, TIME, VSZ, WCHAN.

Instead, mosps can display the following:

WHERE where is the process running.
Special values: "here" - this node; "queue" - not yet started; "Mwait" - the process was started with mosrun -b and is waiting for a suitable node to start on; "Bwait" - the batch job was started with mosrun -b and is waiting for a suitable node to start on.

FROM where is the process' home-node.
Special value: "here" - this node;

CLASS the class of the process (see mosrun(1)).
Special values: "native" - exited MOSIX using the native utility; "batch" - a batch job (started with mosrun -E or mosrun -M).

ORIGPID the original process ID in the process' home-node (in the case of guest batch jobs, several processes from the same guest batch job may share the same PID). "N/A" when the home/sending-node is here.

FRZ Freezing status:
- not frozen
A automatically frozen
E frozen due to being expelled back to the home/cluster
M manually frozen
N/A cannot be frozen (batch, native or guest)

NMIGS the number of times the process (or its MOSIX ancesstors before it was forked) had migrated so far ("N/A" for guest, batch and native processes).

Normally, if the nodes in the WHERE and FROM fields are listed in /etc/mosix/userview.map, then they are displayed as node-numbers: otherwise as IP addresses. The -I argument forces all nodes to be displayed as IP addresses and the -h argument forces all nodes to be displayed as host-names (when the host-name can be found, otherwise as an IP address). Similarly, the -M argument displays just the first component of the host-names. Regardless of those arguments, the local node is always displayed as "here".

When the -L argument is specified, only local processes (those whose home-node is here) are listed.

When the `-O` argument is specified, only local processes that are currently running away from home are listed.

The `-n` argument displays the number of migrations (`NMIGS`).

The `ORIGPID` field is displayed only when the `-P` and/or `-V` arguments are specified.

When the `-V` argument is specified, only guest processes (those whose home-node is not here) are listed: the listing includes `ORIGPID`, but not `WHERE` and `FRZ` (as those only apply to local processes).

The `-D` argument displays the user's estimate of the remaining duration of their process.

The `-S` argument displays the progress of multiple-commands (`mosrun -S`). Instead of ordinary MOSIX processes. Only the main processes (that read commands-files) are displayed. The information provided is:

`TOTAL` total number of command-lines given.
`DONE` number of completed command-lines (including failed commands).
`FAIL` number of command-lines that failed.

The `-J{JobID}` argument limits the output to processes of the given JobID (see `mosrun(1)`).

IMPORTANT NOTES

1. In conformance to the `ps` standards, since guest processes do not have a controlling terminal on this node, in order to list such processes either use the `-V` option, or include a suitable `ps` argument such as `-A`, `ax` or `-ax` (it may depend on the version of `ps` installed on your computer).
2. the `c` option of `ps(1)` is useful to view the first 15 characters of the command being run under `mosrun` instead of seeing only `"mosrun"` in the command field.

SEE ALSO

`ps`(1), `mosrun`(1), `moskillall`(1), `mosix`(7).

NAME

MOSQ - MOSIX queue control

SYNOPSIS

```
mosq [-j] [-p] list
mosq [-j] [-p] listall
mosq [-j] [-p] locallist
mosq [-j] [-p] locallistall
mosq [-j] run {pid|jobID} [{hostname}|{IP}|{node-number}]
mosq [-j] abort {pid|jobID} [{hostname}|{IP}|{node-number}]
mosq [-j] cngpri {newpri} {pid|jobID} [{hostname}|{IP}|{node-number}]
mosq [-j] advance {pid|jobID} [{hostname}|{IP}|{node-number}]
mosq [-j] retard {pid} [{hostname}|{IP}|{node-number}]
```

DESCRIPTION

Mosq displays and controls the content of the job queue - e.g., jobs that were submitted using `mosrun -q`.

`mosq list` displays an ordered table of all queued jobs: their process-ID; user-name; memory requirement (if any); whether confined to the local partition or allowed to use other partitions and clusters in the grid; their priority (the lower the better); the node where they were initiated; and the command line (when available).

`mosq listall` is similar to `list`, except that it also shows jobs that were once queued and are now running. For these jobs, the `PRI` field shows "RUN" instead of a priority.

`mosq locallist` is similar to `list`, but displays only jobs that were initiated on the local node. The `FROM` field is not shown; and unlike `list`, the order of jobs in `locallist` (within each priority and unless affected by the actions below), is according to the submission time of the jobs and not their actual place in the queue.

`mosq locallistall` is similar to `locallist`, except that it also shows jobs that were once queued and are now running.

While `list` and `listall` may be blocked when the per-partition node that is responsible for queuing is inaccessible, `locallist` and `locallistall` can not be blocked because they depend only on the local node.

The `-p` argument adds the number of parallel processes ("NPROC") to the listing.

When the `-j` argument is used in conjunction with `list`, `listall`, `locallist` or `locallistall`, the Job-ID field is included in the listing (it is assigned by `mosrun(1)` using the `"mosrun -J{jobID}"` parameter).

The following commands operate on selected jobs from the queue: when the `-j` argument is not specified, a single job is selected by its process-ID and initiating node, but when the `-j` argument is specified, all jobs with the same User-ID as the caller, and the given Job-ID and initiating node, are selected. The initiating node can be specified as either an IP address, a host-name, a MOSIX logical node-number, or omitted if the job(s) were initiated from the current node.

`mosq cngpri` modifies the priority of the selected job(s): the lower the [non-negative] number - the higher the priority.

`mosq run` force the release of the selected job(s) from the queue and cause them to start running (regardless of the available cluster/multi-cluster resources).

`mosq abort` removes the selected job(s) from the queue, normally killing them (but job(s) that were started by `"mosrun -Q"`, will start running instead).

`mosq advance` move the selected job(s) forward in the queue, making them the first among the queued jobs with the same priority.

`mosq retard` move the selected job(s) backward in the queue, making them the last among the queued jobs with the same priority.

SYNONYMS

The following synonyms are provided for convenience and may be used interchangeably:

`locallist` - listlocal
`locallistall` - listalllocal; listlocalall
`cngpri` - changepri; newpri
`run` - launch; release; activate
`abort` - cancel; kill; delete

SEE ALSO

`mosrun`(1), `mosix`(7).

NAME

MOSPIPE - Run pipelined jobs efficiently using Direct Communication

SYNOPSIS

mospipe [mosrun-options1] {program1+args1} [-e] [mosrun-options2]
{program2+args2} [[-e] [mosrun-options3] {program3+args3}]...

Mosrun options:
[-h|-a.b.c.d|-r{hostname}|-{nodenumber}] [-L|-l]
[-F] [-G[{n}]] [-m{mb}]

DESCRIPTION

Mospipe runs two or more pipelined programs just as a shell would run:

program1 [args1] | program2 [args2] [| program3 [args3]] ...

except that instead of pipes, the connection between the programs uses the MOSIX feature of direct_communication(7) that makes the transfer of data between migrated programs much more efficient.

Each program argument includes the program's space-separated arguments: if you want to have a space (or tab, carriage-return or end-of-line) as part of a program-name or argument, use the backslash(\) character to quote it ('\' can quote any character, including itself).

Each program may optionally be preceded by certain mosrun(1) arguments that control (see mosrun(1) for further details):

-h|-a.b.c.d|-r{hostname}|-{nodenumber}
On which node should the program start.
-F Whether to run even when the designated node is down.
-L|-l Whether the program should be allowed to automatically migrate or not.
-G{n} The class of the program.
-m{mb} The amount of memory that the program requires.

The -e (or -E) argument between programs indicates that as well as the standard-output, the standard-error of the preceding program should also be sent to the standard-input of the following program.

If mospipe is not already running under mosrun(1), then in order to enable direct communication it places itself under mosrun(1). In that case it also turns on the -e flag of mosrun(1) for the programs it runs.

APPLICABILITY

Mospipe is intended to connect simple utilities and applications that read from their standard-input and write to their standard-output (and standard-error).

mospipe sets the MOSIX direct_communication(7) to resemble pipes, so applications that expect to have pipes or sockets as their standard-input/output/error and specifically applications that only use the stdio(3) library to access those, should rarely have a problem running under mospipe.

However, direct_communication(7) and pipes are not exactly the same, so sophisticated applications that attempt to perform complex operations on file-descriptors 0, 1 and 2 (such as lseek(2), readv(2), writev(2), fcntl(2), ioctl(2), select(2), poll(2), dup(2), dup2(2), fstat(2), etc.) are likely to fail. This regrettably includes the tcsh(1) shell.

The following anomalies should also be noted:

* mosrun(1) and native (See mosrun(1)) cannot run under mospipe: attempts to run them will produce a "Too many open files" error.
* An attempt to write 0 bytes to the standard-output/error will create an End-Of-File condition for the next program.
* Input cannot be read by child-processes of the receiver (open direct-communication connections are not inheritable).
* Direct_Communication(7) should not be used by the applications (or at least extreme caution must be used) since direct communication is already being used by mospipe to emulate the pipe(s).

EXAMPLES

```
mospipe "echo hello world" wc
```
is like the shell-command:
```
echo hello world | wc
```
and will produce:
```
   1    2    12
```

```
mospipe "ls /no-such-file" -e "tr [a-z\ ] [A-Z+]"
```
is like the shell-command:
```
ls /no-such-file |& tr '[a-z ]' '[A-Z+]'
```
and will produce:
LS:+/NO-SUCH-FILE:+NO+SUCH+FILE+OR+DIRECTORY

```
b='bestnode'
mospipe "echo hello world" -$b -L bzip2 -$b -L bzcat "tr [a-z] [A-Z]"
```
is like the shell-command:
```
echo hello world | bzip | bzcat | tr '[a-z]' '[A-Z]'
```

It will cause both compression (bzip) and decompression (lzop -d) to run and stay on the same and best node for maximum efficiency, and will produce:

HELLO WORLD

SEE ALSO

direct_communication(7), mosrun(1), mosix(7).

NAME

MOSKILLALL - Kill or signal all your MOSIX processes

SYNOPSIS

moskillall [-{signum} | -{symbolic_signal}] -G[{class}] [-J{jobid}]

Symbolic signals:

HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE
ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU
XFSZ VTALRM PROF WINCH POLL PWR SYS

DESCRIPTION

Moskillall kills or sends a signal to a group of processes: the default signal is SIGTERM, unless the numeric -{signum} or symbolic -{symbolic_signal} argument specify a different signal.

If no arguments are specified, the signal is sent to all of the user's MOSIX processes. When invoked by the Super-User, the signal is sent to all the MOSIX processes of all the users.

The -G[{class}] argument causes the signal to be sent only to MOSIX processes of the given class (See mosrun(1)). When class is omitted, its value is assumed to be 1.

The -J{jobid} argument causes the signal to be sent only to the user's (including the Super-User) processes that were started with 'mosrun -J{jobid}' (but when jobid is 0, the signal is also sent to all the user's MOSIX processes that were started without the -J argument: See mosrun(1)).

Note that moskillall cannot provide an absolute guarantee that all processes of the requested group will receive the signal when there is a race condition in which one or more processes are forking at the exact time of running it.

DEFINITION OF MOSIX PROCESSES

MOSIX processes are processes that were started by the mosrun(1) utility, including batch and native processes that do not run under MOSIX and processes that are queued, but excluding processes that were originated from other nodes (even if a guest batch job invoked mosrun explicitly).

SEE ALSO

mosps(1), mosrun(1), mosix(7).

NAME

BESTNODE - Select best node to run on

SYNOPSIS

bestnode [-u] [-n] [-G] [-w] [-m{mb}]

DESCRIPTION

Bestnode selects the best node to run a new job.

Bestnode normally prints the selected node's IP address. If the -u argument is used and the node has an associated MOSIX node number, bestnode prints its MOSIX node number instead.

The selection is normally for the immediate sending of a job to the selected node by means other than mosrun(1) (such as "rsh", "ssh", or MPI). MOSIX is updated to assume that a new process will soon start on the selected node: when calling bestnode for any other purpose (such as information gathering), use the -n argument, to prevent misleading MOSIX that a new process is about to be started.

The -G argument widens the node selection to the whole multi-cluster grid - otherwise, only nodes within the local cluster are chosen (the local partition, if the cluster is partitioned).

The -m{mb} argument narrows the selection to nodes that have at least mb Megabytes of free memory.

When the -w argument is used, bestnode waits until an appropriate node is found: otherwise, if no appropriate node is found, bestnode prints "0" and exits.

SEE ALSO

mosrun(1), mosix(7).

NAME

MIGRATE - Manual control of running MOSIX processes

SYNOPSIS

```
migrate {{pid}|-j{jobID}} {node-number|IP-address|host}
migrate {{pid}|-j{jobID}} home
migrate {{pid}|-j{jobID}} out
migrate {{pid}|-j{jobID}} freeze
migrate {{pid}|-j{jobID}} continue
migrate {{pid}|-j{jobID}} checkpoint
migrate {{pid}|-j{jobID}} chkstop
migrate {{pid}|-j{jobID}} chkexit
```

DESCRIPTION

`Migrate {pid}` manually migrates or otherwise affects a given MOSIX process (`pid`).
`Migrate -j{jobID}` does the same to all of the user's processes with the given `jobID` (see `mosrun(1)`).

The first option (`{node-number|IP-address|host}`) specifies a recommended target node to which to migrate the process(es). Note that no error is returned if MOSIX ignores this recommendation.

The `home` option forces the process(es) to migrate back to its home-node.

The `out` option forces a guest process(es) to move out of this node (this option is available only to the Super-User,

The `freeze` option freezes the process(es) (guest processes may not be frozen).

The `continue` option unfreeze the process(es).

The `checkpoint` option requests the process(es) to take a checkpoint.

The `chkstop` option requests the process(es) to take a checkpoint and stop: the process(es) may then be resumed with SIGCONT.

The `chkexit` option requests the process(es) to take a checkpoint and exit.

`Migrate` sends the instruction, but does not wait until the process(es) respond to it (or reject it).

Except for the Super-User, one can normally migrate or affect only their own processes. The rule is: if you can kill it, you are also allowed to migrate/affect it.

The best way to locate and find the `PID` of MOSIX processes is by using `mosrun(1)`, `mosps(1)`.

SEE ALSO

`mosps`(1), `mosix`(7).

NAME

mon − MOSIX monitor

SYNOPSIS

mon [**-v** | **-V** | **-w**] [**-t**] [**-d**] [**-s** | **-m** | **-m -m** | **-u** | **-f** | **-l**]

DESCRIPTION

Mon displays useful current information about MOSIX nodes. The information is represented as a bar-chart, which allows a comparison between different nodes.

The display shows nodes that are assigned node numbers in `/etc/mosix/userview.map` (see `mosix(7)`).

The following options are available:

-w Select horizontal numbering: better display - but less nodes are visible.

-v Select vertical numbering: more nodes are visible - but denser display.

-V Select tight vertical numbering: maximal number of nodes are visible - but even denser display.

-t Display the number of operational nodes and number of CPUs.

-d Display also dead (not responding) nodes.

-s Display the CPU-speeds.

-m Display the used vs. total memory.

-m -m
Display the used vs. total swap space.

-u Display the "utilizability" (see below).

-f Display the number of frozen processes.

-l Display the load (default).

While in `mon`, the following keys may be used:

`v` Select vertical numbering.

`V` Select tight vertical numbering.

`w` Select horizontal numbering.

`a` Select automatic numbering (vertical numbering will be selected if it would make the difference between viewing all nodes or not - otherwise, horizontal numbering is selected).

`s` Display CPU-speeds and number of nodes (if more than 1).
10000 units represent a standard processor (currently Pentium-IV CPU at 3GHz).

`m` Display used memory vs. total memory: the used memory is displayed as a solid bar while the total memory is extended with '+' signs. The memory is shown in Megabytes.

`m m` (pressing `m` twice) Display used swap-space vs. total swap-space: the used swap-space is displayed as a solid bar while the total swap-space is extended with '+' signs. Swap-space is shown in Gigabytes and is accurate to 0.1GB.

`u` Display the percentage of the CPU "utilizability". A node is considered utilizable when all its CPUs are either running a process, or when no processes wish to run: this excludes the situation when there are processes that wish to run but cannot because they wait for a swap-page. Utilizability should normally be 100% - otherwise, it means that memory shortage causes the node to slow down.

`f` Display the number of frozen processes.

`l` Display loads. The load represents one CPU-bound process that runs on a node with a single CPU under normal conditions. The load increases proportionally on slower CPUs, and decreases proportionally on faster CPUs and on nodes with more than one CPU. The load also increases when the utilizability is below 100%.

`d` Display also dead (not-responding) nodes.

`D` Stop displaying dead nodes.

`t` Toggle displaying the count of operational nodes.

`Right-Arrow`
Move one node to the right (when not all nodes fit on the screen).

`Left-Arrow`
Move one node to the left (when not all nodes fit on the screen).

`n` Move one screen to the right (when not all nodes fit on one screen).

`p` Move one screen to the left (when not all nodes fit on one screen).

`h` Bring up a help screen.

`Enter` Redraw the screen.

`q` Quit **`Mon.`**

SEE ALSO

`mosix`(7).

NAME

`testload` − V1.1, MOSIX test program

SYNOPSIS

```
testload [OPTIONS]
```

DESCRIPTION

A test program that generates artificial load and consumes memory for testing the operation of MOSIX.

OPTIONS

`-t{seconds}` | `--time={seconds}`
: Run for a given number of CPU `seconds`: the default is 1800 seconds (30 minutes). A value of 0 causes `testload` to run indefinitely. OR:

`-t{min},{max}` | `--time={min},{max}`
: Run for a random number of seconds between `min` and `max`.

`-m{mb}, --mem={mb}`
: amount of memory to consume in Megabytes (by default, `testload` consumes no significant amount of memory).

`--random-mem`
: Fill memory with a random pattern (otherwise, memory is filled with the same byte-value).

`--cpu={N}`
: When testing pure CPU jobs - perform `N` units of CPU work, then exit. When also doing system-calls (`--read`, `--write`, `--noiosyscall`) - perform `N` units of CPU work between chunks of system-calls.

`--read[={size}[,{ncalls}[,{repeats}]]]`
`--write[={size}[,{ncalls}[,{repeats}]]]`
: perform `read` OR `write` system calls of `size` KiloBytes (default=1KB). These calls are repeated in a chunk of `ncalls` times (default=1024), then those chunks are repeated `repeats` times (default=indefinitely), with optional CPU work between chunks if the `--cpu` option is also set.

`--noiosyscall={ncalls}[,{repeats}]`
: perform some other system call that does not involve I/O `ncalls` times (default=1024), repeat this `{repeats}` times (default=indefinitely), with optional CPU work in between if the `--cpu` option is also set.

`-d, --dir={directory}`
`-f, --file={filename}`
: select a directory OR a file on which to perform reading or writing (the default is to create a file in the /tmp directory).

`--maxiosize={SIZE}`
: Once the file size reaches `SIZE` megabytes, further I/O will resume at the beginning of the file.

`-v, --verbose`
: produce debug-output.

`--report-migrations`
: Report when `testload` migrates.

`-r, --report`
: Produce summary at end of run.

`--sleep SEC`
Sleep for `SEC` seconds before starting

`-h, --help`
Display a short help screen.

EXAMPLES

`testload -t 20`
run CPU for 20 seconds

`testload -l 10 -h 20`
runs CPU for a random period of time between 10 and 20 seconds.

`testload -f /tmp/20MB --write 32,640,1`
writes 32 KiloBytes of data 640 times (total 20 megabytes) to the file /tmp/20MB.

`testload -f /tmp/10MB --write 32,640 --maxiosize 10 --cpu=20`
writes 32 KiloBytes of data 640 times (total 20 megabytes) to the file /tmp/10MB, alternating this indefinitely with running 20 units of CPU. The file "/tmp/10MB" is not allowed to grow beyond 10 MegaBytes: once reaching that limit, writing resumes at the beginning of the file.

AUTHOR

Adapted from code by Lior Amar

NAME

MOSIX − sharing the power of clusters and multi-cluster grids

INTRODUCTION

MOSIX is a generic solution for dynamic management of resources in a cluster or in a multi-cluster organizational grid. MOSIX allows users to draw the most out of all the connected computers, including utilization of idle computers.

At the core of MOSIX are adaptive resource sharing algorithms, applying preemptive process migration based on processor loads, memory and I/O demands of the processes, thus causing the cluster or the multi-cluster grid to work cooperatively similar to a single computer with many processors.

Unlike earlier versions of MOSIX, only programs that are started by the `mosrun(1)` utility are affected and can be considered "migratable" - other programs are considered as "standard Linux programs" and are not affected by MOSIX.

MOSIX maintains a high level of compatiblity with standard Linux, so that binaries of almost every application that runs under Linux can run completely unmodified under the MOSIX "migratable" category. The exceptions are usually system-administration or graphic utilities that would not benefit from process-migration anyway. If a "migratable" program that was started by `mosrun(1)` attempts to use unsupported features, it will either be killed with an appropriate error message, or if a ‘‘do not kill’’ option is selected, an error is returned to the program: such programs should probably run as standard Linux programs.

In order to improve the overall resource usage, processes of "migratable" programs may be moved automatically and transparently to other nodes within the cluster or even the multi-cluster grid. As the demands for resources change, processes may move again, as many times as necessary, to continue optimizing the overall resource utilization, subject to the inter-grid priorities and policies. Manual-control over process migration is also supported.

`MOSIX` is particularly suitable for running CPU-intensive computational programs with unpredictable resource usage and run times, and programs with moderate amounts of I/O. Programs that perform large amounts of I/O should better be run as standard Linux programs.

Apart from process-migration, MOSIX can provide both "migratable" and "standard Linux" programs with the benefits of optimal initial assignment and live-queuing. The unique feature of `live-queuing` means that although a job is queued to run later, when resources are available, once it starts, it remains attached to its original Unix/Linux environment (standard-input/output/error, signals, etc.).

REQUIREMENTS

1. All nodes must run Linux (any distribution - mixing allowed).
2. All participating nodes must be connected to a network that supports TCP/IP and UDP/IP, where each node has a unique IP address in the range 0.1.0.0 to 255.255.254.255 that is accessible to all the other nodes.
3. TCP/IP ports 249-253 and UDP/IP ports 249-250 must be available for MOSIX (not used by other applications or blocked by a firewall).

4. The architecture of all nodes can be either i386 (32-bit) or x86_64 (64-bit). Processes that are started on a 32-bit node can migrate to a 64-bit node, but not the opposite.
5. In multiprocessor nodes (SMP), all the processors must be of the same speed.
6. The system-administrators of all the connected nodes must be able to trust each other (see more on SECURITY below).

CLUSTER, GRID, PARTITION

The MOSIX concept of a "cluster" is a collection of computers that are owned and managed by the same entity (a person, a group of people or a project) - this can at times be quite different than a hardware cluster, as each MOSIX cluster may range from a single workstation to a large combination of computers - workstations, servers, blades, multi-core computers, etc. possibly of different speeds and number of processors and possibly in different locations.

A MOSIX multi-cluster "grid" is a collection of clusters that belong to different entities (owners) who wish to share their resources subject to certain administrative conditions. In particular, when an owner needs its computers - these computers must be returned immediately to the exclusive use of their owner. An owner can also assign priorities to guest processes of other owners, defining who can use their computers and when. Typically, an owner is an individual user, a group of users or a department that own the computers. The grid is usually restricted, due to trust and security reasons, to a single organization, possibly in various sites/branches, even across the world.

MOSIX supports dynamic grid configurations, where clusters can join and leave the grid at any time. When there are plenty of resources in the grid, the MOSIX queuing system allows more processes to start. When resources become scarce (because other clusters leave or claim their resources and processes must migrate back to their home-clusters), MOSIX has a freezing feature that can automatically freeze excess processes to prevent memory-overload on the home-nodes.

Clusters may also be sub-divided into "partitions". Nodes that are assigned to different cluster-partitions are halfway between being part of the cluster and belonging to a different cluster in the grid.

Just as within the cluster:
1. All cluster-partitions seem to other clusters in the grid as one cluster (eliminating the need to inform and update system-administrators of other clusters about internal changes to one's cluster).
2. Processes that migrate to another partition share the same top-priority over processes from other clusters.
3. Processes that migrate to another partition share the "Cluster" category disk-space allocation rather than the "Grid" category for Private Temporary Files (see below).

However, just as other clusters:
1. Only processes that were allowed to migrate to other clusters are allowed to migrate to other partitions.
2. Batch jobs cannot be assigned to nodes in other partitions.
3. Each partition maintains its own job-queue.

When you have both 32-bit and 64-bit computers in the same cluster, it is highly recommended (but not mandatory) to set them up as different cluster partitions.

CONFIGURATION

To configure MOSIX interactively, simply run `mosconf`: it will lead you step-by-step through the various configuration items.

`Mosconf` can be used in two ways:

1. To configure the local node (press <Enter> at the first question).
2. To configure MOSIX for other nodes: this is typically done on a server that stores an image of the root-partition for some or all of the cluster-nodes. This image can, for example, be NFS-mounted by the cluster-nodes, or otherwise copied or reflected to them by any other method: at the first question, enter the path to the stored root-image.

There is no need to stop MOSIX in order to modify the configuration - most changes will take effect within a minute. However, after modifying the list of nodes in the cluster (`/etc/mosix/mosix.map`) or `/etc/mosix/mosip` or `/etc/mosix/myfeatures`, you should run the command `"setpe"` (but when you are using `mosconf` to configure your local node, this is not necessary).

Below is a detailed description of the MOSIX configuration files (if you prefer to edit them manually).

The directory `/etc/mosix` should include at least the subdirectories `/etc/mosix/partners`, `/etc/mosix/var`, `/etc/mosix/var/grid` and the following files:

`/etc/mosix/mosix.map`

This file defines which computers participate in your MOSIX cluster. The file contains up to 256 data lines and/or alias lines that can be in any order. It may also include any number of comment lines beginning with a '#', as well as empty lines.

Data lines have 2 or 3 fields:

1. The IP ("a.b.c.d" or host-name) of the first node in a range of nodes with consecutive IPs.
2. The number of nodes in that range.
3. Optional combination of letter-flags and/or an integer:
   `p[roximate]` do not use compression on migration, e.g., over fast networks or slow CPUs.
   `o[utsider]` inaccessible to local-class processes.
   `{partition}` a positive integer indicating the partition number for that range.

Alias lines are of the form:

```
a.b.c.d=e.f.g.h
```

or

```
a.b.c.d=host-name
```

They indicate that the IP address on the left-hand-side refers to the same node as the right-hand-side.

`NOTES:`

1. It is an error to attempt to declare the local node an "outsider".
2. When using host names, the first result of `gethostbyname(3)` must return their IP address that is to be used by MOSIX: if in doubt - specify the IP address.

3. The right-hand-side in alias lines must appear within the data lines.
4. IP addresses 0.0.x.x and 255.255.255.x are not allowed in MOSIX.
5. If you change `/etc/mosix/mosix.map` while MOSIX is running, you need to run `setpe` to notify MOSIX of the changes.

`/etc/mosix/secret`
: This is a security file that is used to prevent ordinary users from interfering and/or compromizing security by connecting to the internal MOSIX TCP ports. The file should contain just a single line with a password that must be identical on all the nodes of the cluster/grid. This file must be accessible to ROOT only (chmod 600!)

`/etc/mosix/ecsecret`
: Like `/etc/mosix/secret`, but used for running batch jobs as a client (see `mosrun(1)`). If you do not wish to allow this node to send batch-jobs, do not create this file.

`/etc/mosix/essecret`
: Like `/etc/mosix/secret`, but used for running batch jobs as a server (see `mosrun(1)`). The password must match the client's `/etc/mosix/ecsecret`. If you do not wish to allow this node to be a batch-server, do not create this file.

The following files are optional:

`/etc/mosix/mosip`
: This file contains our IP address, to be used for MOSIX purposes, in the regular format - `a.b.c.d`. This file is only necessary when the node's IP address is ambiguous: it can be safely omitted if the output of `ifconfig(8)` ("inet addr:") matches exactly one of the IP addresses listed in the data lines of `/etc/mosix/mosix.map`.

`/etc/mosix/myfeatures`
: This file contains one line of comma-separated topological features for this node (if any). For example: `yellow,wood,chicken`.

  The list of all 32 features (one line per feature) can be found in `/etc/mosix/features`.

  If this file is missing, this node is assumed to have no topological features. (see `topology(7)`)

`/etc/mosix/freeze.conf`
: This file sets the automatic freezing policies on a per-class basis for MOSIX processes originating in this node. Each line describes the policy for one class of processes. The lines can be in any order and classes that are not mentioned are not touched by the automatic freezing mechanisms.

  The space-separated constants in each line are as follows:
  1. class-number
     A positive integer identifying a class of processes.
  2. load-units:
     Used in fields #3-#6 below: 0=processes; 1=standard-load
  3. RED-MARK (floating point)
     Freeze when load is higher.

4. BLUE-MARK (floating point)
   Unfreeze when load is lower.
5. minautofreeze (floating point)
   Freeze processes that are evacuated back home on arrival if load gets equal or above this.
6. minclustfreeze (floating point)
   Freeze processes that are evacuated back to this cluster on arrival if load gets equal or above this.
7. min-keep
   Keep running at least this number of processes - even if load is above RED-MARK.
8. max-procs
   Freeze excess processes above this number - even if load is below BLUE-MARK.
9. slice
   Time (in minutes) that a process of this class is allowed to run while there are automatically-frozen process(es) of this class. After this period, the running process will be frozen and a frozen process will start to run.
10. killing-memory
    Freezing fails when there is insufficient disk-space to save the memory-image of the frozen process - kill processes that failed to freeze and have above this number of MegaBytes of memory. Processes with less memory are kept alive (and in memory). Setting this value to 0, causes all processes of this class to be killed when freezing fails. Setting it to a very high value (like 1000000 MegaBytes) keeps all processes alive.

NOTES:

1. The load-units in fields #3-#6 depend on field #2. If 0, each unit represents the load created by a CPU-bound process on this computer. If 1, each unit represents the load created by a CPU-bound process on a "standard" MOSIX computer (e.g. a 3GHz Intel Core). The difference is that the faster the computer and the more processors it has, the load created by each CPU process decreases proportionally.
2. Fields #3,#4,#5,#6 are floating-point, the rest are integers.
3. A value of "-1" in fields #3,#5,#6,#8 means ignoring that feature.
4. The first 4 fields are mandatory: omitted fields beyond them have the following values: minautofreeze=-1,minclusterfreeze=-1,min-keep=0, max-procs=-1,slice=20.
5. The RED-MARK must be significantly higher than BLUE-MARK: otherwise a perpetual cycle of freezing and unfreezing could occur. You should allow at least 1.1 processes difference between them.
6. Frozen processes do not respond to anything, except an unfreeze request or a signal that kills them.
7. Processes that were frozen manually are not unfrozen automatically.

This file may also contain lines starting with '/' to indicate freezing-directory names. A "Freezing directory" is an existing directory (often a mount-point) where the memory contents of frozen process is saved. For successful freezing, the disk-partition of freezing-directories should have sufficient free disk-space to contain the memory image of all the frozen processes.

If more than one freezing directory is listed, the freezing directory is chosen at random by each freezing process. It is also possible to assign selection probabilities by adding a numeric weight after the directory-name, for example:

```
/tmp      2
/var/tmp  0.5
/mnt/tmp  2.5
```

In this example, the total weight is 2+0.5+2.5=5, so out of every 10 frozen processes, an average of 4 (10∗2/5) will be frozen to `/tmp`, an average of 1 (10∗0.5/5) to `/var/tmp` and an average of 5 (10∗2.5/5) to `/mnt/tmp`.

When the weight is missing, it defaults to 1. A weight of 0 means that this directory should be used only if all others cannot be accessed.

If no freezing directories are specified, all freezing will be to the `/freeze` directory (or symbolic-link).

Freezing files are usually created with "root" (Super-User) permissions, but if `/etc/mosix/freeze.conf` contains a line of the form:

```
U {UID}
```

then they are created with permissions of the given numeric `UID` (this is sometimes needed when freezing to NFS directories that do not allow "root" access).

`/etc/mosix/partners/∗`

If your cluster is part of a multi-cluster grid, then each file in `/etc/mosix/partners` describes another cluster that you want this cluster to cooperate with.

The file-names should indicate the corresponding cluster-names (maximum 128 characters), for example: "geography", "chemistry", "management", "development", "sales", "students-lab-A", etc. The format of each file is a follows:

`Line #1:`

A verbal human-readable description of the cluster.

`Line #2:`

Four space-separated integers as follows:

1. Priority:
   0-65535, the lower the better.
   The priority of the local cluster is always 0. MOSIX gives precedence to processes with higher priority - if they arrive, guests with lower priority will be expelled.
2. Cango:
   0=never send local processes to that cluster.
   1=local processes may go to that cluster.
3. Cantake:
   0=do not accept guest-processes from that cluster.
   1=accept guest-processes from that cluster.

4. Canexpand:
   - 0=no: Only nodes listed in the lines below may be recognized as part of that cluster: if a core node from that cluster tells us about other nodes in their cluster - ignore those unlisted nodes.
   - 1=yes: Core-nodes of that cluster may specify other nodes that are in that cluster, and this node should believe them even if they are not listed in the lines below.
   - -1=do not ask the other cluster: do not consult the other cluster to find out which nodes are in that cluster: instead just rely on and use the lines below.

`Following lines:`

Each line describes a range of consecutive IP addresses that are believed to be part of the other cluster, containing 5 space-separated items as follows:

1. IP1 (or host-name): First node in range.
2. n: Number of nodes in this range.
3. Core:
   - 0=no: This range of nodes may not inform us about who else is in that cluster.
   - 1=yes: This range of nodes could inform us of who else is in that cluster.
4. Participate:
   - 0=no This range is (as far as this node is concerned) not part of that cluster.
   - 1=yes This range is probably a part of that cluster.
5. Proximate:
   - 0=no Use compression on migration to/from that cluster.
   - 1=yes Do not use compression when migrating to/from that cluster (network is very fast and CPU is slow).

`NOTES:`

1. From time-to-time, MOSIX will consult one or more of the "core" nodes to find the actual map of their cluster. It is recommended to list such core nodes. The alternative is to set `canexpand` to -1, causing the map of that cluster to be determined solely by this file.
2. Nodes that do not "participate" are excluded even if listed as part of their cluster by the core-nodes (but they could possibly still be used as "core-nodes" to list other nodes)
3. All core-nodes must have the same value for "proximate", because the "proximate" field of unlisted nodes is copied from that of the core-node from which we happened to find about them and this cannot be ambiguous.
4. When using host names rather than IP addresses, the first result of `gethostbyname(3)` must return their IP address that is used by MOSIX: if in doubt - specify the IP address instead.
5. IP addresses 0.0.x.x and 255.255.255.x cannot be used in MOSIX.

`/etc/mosix/userview.map`

Although it is possible to use only IP numbers and/or host-names to specify nodes in your cluster (and multi-cluster grid), it is more convenient to use small integers as node numbers: this file allows you to map integers to IP addresses. Each line in this file contains 3 elements:

1. A node number (1-65535)
2. IP1 (or host-name, clearly identifiable by `gethostbyname(3)`)
3. Number of nodes in range (the number of the last one must not exceed 65535)

It is up to the cluster administrator to map as few or as many nodes as they wish out of their cluster and multi-cluster grid - the most common practice is to map all the nodes in one's cluster, but not in other clusters.

`/etc/mosix/queue.conf`

This file configures the queueing system (see `mosrun(1)`, `mosq(1)`). All lines in this file are optional and may appear in any order.

Usually, one node in each cluster is elected by the system-administrator to manage the queue, while the remaining nodes point to that manager. As an exception, in a mixed cluster that has both 32-bit and 64-bit computers, a separate 32-bit node should be elected to exclusively manage the queue for all 32-bit nodes and a 64-bit node elected to exclusively manage the queue for all 64-bit nodes.

`Defining the queue manager:`

The line:

```
C {hostname}
```

assigns a specific node from the cluster (`hostname`) to manage the job queue. In the absence of this line, each node manages its own queue (which is usually undesirable). It is possible to have several 'C' lines - one for each cluster-partition.

`Defining the default priority:`

The line:

```
P {priority}
```

assigns a default job-priority to all the jobs from this node. The lower this value - the higher the priority. In the absence of this line, the default priority is 50.

Commonly, user-ID's are identical on all the nodes in the cluster. The line (with a single letter):

```
S
```

indicates that this is not the case, so users on other nodes (except the Super-User) will be prevented from sending requests to modify the status of queued jobs from this node.

`Configuring the queue manager:`

The following lines are relevant only in the queue manager node and are ignored on all other nodes:

The MOSIX queueing system determines dynamically how many processes to run. The line:

```
M {maxproc}
```

if present, imposes a maximal number of processes that are allowed to run from the queue simultaneously on top of the regular queueing policy. For example,

```
M 20
```

sets the upper limit to 20 processes, even when more resources are available.

The line:

```
X {1 <= x <= 8}
```

defines the maximal number of queued processes that may run simultaneously per CPU. This option applies only to processors within the cluster and is not available for other clusters in the

grid (where the queueing system assigns at most one process per CPU). In the absence of this line the default is

```
X 1
```

The line:

```
Z {n}
```

causes the first `n` jobs of priority `0` to start immediately (out of order), without checking whether resources are available, leaving that responsibility to the system administrator.

Example: the cluster has 10 dual-CPU nodes, so the queueing system normally allows 20 jobs to run. In order to allow urgent jobs to run immediately (without waiting for regular jobs to complete), the system administrator configures a line: `Z 10`, thus allowing each node to run a maximum of 3 jobs.

The line:

```
N {n} [{mb}]
```

causes the first `n` jobs of each user to start immediately (out of order), without checking whether resources are available. Only jobs above that number, per user, will be queued and whenever the number of a user's running jobs drops below this number, a new job of that user (if there is any waiting) will start to run.

When the `mb` parameter is given, only jobs that do not exceed this amount of memory in MegaBytes will be started this way.

The system-administrator should weigh carefully, based on knowledge about the patterns of jobs that users typically run, the benefits of this option against its risks, such as having at times more jobs in their cluster(s) than available memory to run them efficiently. If this option is selected with a memory-limitation (`mb`), then the system-administrator should request that users always specify the maximum memory-requiremnts for all their queued jobs (using `mosrun -m`).

`Fair-share policy:`

The fairness policy determine the order in which jobs are initially placed in the queue. Note that fairness should not be confused with priority (as defined by the `P {priority}` line or by `mosrun -q{pri}` and possibly modified by `mosq(1)`): priorities always take precedence, so here we only discuss the initial placement in the queue of jobs with the same priority.

The default queueing policy is "first-come-first-served". Alternatively, jobs of different users can be placed in the queue in an interleaved manner.

The line (with a single letter):

```
F
```

switches the queueing policy to the interleaved policy.

The advantage of the interleaved approach is that a user wishing to run a relatively small number of processes, does not need to wait for all the jobs that were already placed in the queue. The disadvantage is that older jobs need to wait longer.

Normally, the interleaving ratio is equal among all users. For example, with two users (A and B) the queue may look like A-B-A-B-A-B-A-B.

Each user is assigned an `interleave ratio` which determines (proportionally) how well their jobs will be placed in the queue relative to other users: the

smaller that ratio - the better placement they will get (and vice versa). Normally all users receive the same default interleave-ratio of `10` per process. However, lines of the form:

```
U {UID} {1 <= interleave <= 100}
```

can set a different interleave ratio for different users. `UID` can be either numeric or symbolic and there is no limit on the number of these `'U'` lines. Examples:

1. Two users (A & B):
   `U userA 5`
   (userB is not listed, hence it gets the default of 10)
   The queue looks like: A-A-B-A-A-B-A-A-B...
2. Two users (A & B):
   `U userA 20`
   `U userB 15`
   The queue looks like: B-A-B-A-B-A-B-B-A-B-A-B-A-B-B-A...
3. Three users (A, B & C):
   `U userA 25`
   `U userB 7`
   (userC is not listed, hence it gets the default of 10) The queue looks like: B-C-B-C-B-A-B-C-B-C-B-A-B-C-B-C...

Note that since the interleave ratio is determined per process (and not per job), different (more complex) results will occur when multi-process jobs are submitted to the queue.

`/etc/mosix/private.conf`

This file specifies where Private Temporary Files (PTFs) are stored: PTFs are an important feature of `mosrun(1)` and may consume a significant amount of disk-space. It is important to ensure that sufficient disk-space is reserved for PTFs, but without allowing them to disturb other jobs by filling up disk-partitions. Guest processes can also demand unpredictable amounts of disk-space for their PTFs, so we must make sure that they do not disturb local operations.

Up to 3 different directories can be specified: for local processes; guest-processes from the local cluster (including other partitions); and guest-processes from other clusters in the grid. Accordingly, each line in this file has 3 fields:

1. A combination of the letters: 'O' (own node), 'C' (own cluster) and 'G' (other clusters in the grid). For example, `OC`, `C`, `CG` or `OCG.`
2. A directory name (usually a mount-point) starting with '/', where PTFs for the above processes are to be stored.
3. An optional numeric limit, in Megabytes, of the total size of PTFs per-process.

If `/etc/mosix/private.conf` does not exist, then all PTFs will be stored in "/private". If the directory "/private" also does not exist, or if `/etc/mosix/private.conf` exists but does not contain a line with an appropriate letter in the first field ('O', 'C' or 'G'), then no disk-space is allocated for PTFs of the affected processes, which usually means that processes requiring PTFs will not be able to run on this node. Such guest processes that start using PTFs will migrate back to their home-nodes.

When the third field is missing, it defaults to:

5 Gigabytes for local processes.
2 Gigabytes for processes from the same cluster.
1 Gigabyte for processes from other clusters in the grid.

In any case, guest processes cannot exceed the size limit of their home-node even on nodes that allow them more space.

`/etc/mosix/retainpri`

This file contains an integer, specifying a delay in seconds: how long after all MOSIX processes of a certain priority (see above, `/etc/mosix/priority`) finish (or leave) to allow processes of lower priority (higher numbers) to start. When this file is absent, there is no delay and processes with lower priority may arrive as soon as there are no processes with a higher priority.

`/etc/mosix/speed`

If this file exists, it should contain a positive integer (1-10,000,000), providing the relative speed of the processor: the bigger the faster, where 10,000 units of speed are equivalent to a 3GHz Intel Core, and AMD (Athlon or Opteron) processors are, as a rule of thumb, 1.5 times faster than Intel processors of the same frequency.

Normally this file is not necessary because the speed of the processor is automatically detected by the kernel when it boots. There are however two cases when you should consider using this option:

1. When you have a heterogeneous cluster and always use MOSIX to run a specific program (or programs) that perform better on certain processor-types than on others.
2. On Virtual-Machines that run over a hosting operating-system: in this case, the speed that the kernel detects is unreliable and can vary significantly depending on the load of the underlying operating-systems when it boots.

`/etc/mosix/maxguests`

If this file exists, it should contain an integer limit on the number of simultaneous guest-processes from other clusters in the grid. Otherwise, the maximum number of guest-processes from other clusters is set to the default of 8 times the number of processors.

`/etc/mosix/.log_mosrun`

When this file is present, information about invocations of `mosrun(1)` and process migrations will be recorded in the system-log (by default "/var/log/messages" on most Linux distributions).

`/etc/mosix/newtune`

Tuning constants optimizes the MOSIX performance by telling it about the costs of networked operations. MOSIX has built-in tuning default constants. This file is used to override them to suit your particular hardware and networks.

For most users, This file is difficult to set up manually. Thus, MOSIX comes with a program to assemble it. For more information, see `topology(7)`.

INTERFACE FOR PROGRAMS

The following interface is provided for programs running under `mosrun(1)` that wish to interface with their MOSIX run-time environment:

All access to MOSIX is performed via the "open" system call, but the use of "open" is incidental and does not involve actual opening of files. If the program were to run as a regular Linux program, those "open" calls would fail, returning -1, since the quoted files never exist, and `errno(3)` would be set to ENOENT.

`open("/proc/self/{special}", 0)`
reads a value from the MOSIX run-time environment.

`open("/proc/self/{special}", 1|O_CREAT, newval)`
writes a value to the MOSIX run-time environment.

`open("/proc/self/{special}", 2|O_CREAT, newval)`
both writes a new value and return the previous value.

(the `O_CREAT` flag is only required when your program is compiled with the 64-bit file-size option, but is harmless otherwise).

Some "files" are read-only, some are write-only and some can do both (rw). The "files" are as follows:

`/proc/self/migrate`
: writing a 0 migrates back home; writing -1 causes a migration consideration; writing the unsigned value of an IP address or a logical node number, attempts to migrate there. Successful migration returns 0, failure returns -1 (write only)

`/proc/self/lock`
: When locked (1), no automatic migration may occur (except when running on the current node is no longer allowed); when unlocked (0), automatic migration can occur. (rw)

`/proc/self/whereami`
: reads where the program is running: 0 if at home, otherwise usually an unsigned IP address, but if possible, its corresponding logical node number. (read only)

`/proc/self/nmigs`
: reads the total number of migrations performed by this process and its MOSRUN ancesstors before it was born. (read only)

`/proc/self/sigmig`
: Reads/sets a signal number (1-64 or 0 to cancel) to be received after each migration. (rw)

`/proc/self/glob`
: Reads/modifies the process class. Processes of class 0 are not allowed to migrate outside the local cluster or even outside the local partition. Classes can also affect the automatic-freezing policy. (rw)

`/proc/self/needmem`
: Reads/modifies the process's memory requirement in Megabytes, so it does not automatically migrate to nodes with less free memory. Acceptable values are 0-262143. (rw)

`/proc/self/unsupportok`
: when 0, unsupported system-calls cause the process to be killed; when 1 or 2, unsupported system-calls return -1 with errno set to ENOSYS; when 2, an appropriate error-message will also be written to stderr. (rw)

`/proc/self/clear`
: clears process statistics. (write only)

`/proc/self/cpujob`
: Normally when 0, system-calls and I/O are taken into account for migration considerations. When set to 1, they are ignored. (rw)

`/proc/self/localtime`
: When 0, `gettimeofday(2)` is always performed on the home node. When 1, the date/time is taken from where the process is running. (rw)

`/proc/self/decayrate`
: Reads/modifies the decay-rate per second (0-10000): programs can alternate between periods of intensive CPU and periods of demanding I/O. Decisions to migrate should be based neither on momentary program behaviour nor on extremely long term behaviour, so a balance must be struck, where old process statistics gradually decay in favour of newer statistics. The lesser the decay rate, the more weight is given to new information. The higher the decay rate, the more weight is given to older information. This option is provided for users who know well the cyclic behavior of their program. (rw)

`/proc/self/checkpoint`
: When writing (any value) - perform a checkpoint. When only reading - return the version number of the next checkpoint to be made. When reading and writing - perform a checkpoint and return its version. Returns -1 if the checkpoint fails, 0 if writing only and checkpoint is successful. (rw)

`/proc/self/checkpointfile`
: The third argument (`newval`) is a pointer to a file-name to be used as the basis for future checkpoints (see `mosrun(1)`). (write only)

`/proc/self/checkpointlimit`
: Reads/modifies the maximal number of checkpoint files to create before recycling the checkpoint version number. A value of 0 unlimits the number of checkpoints files. The maximal value allowed is 10000000.

`/proc/self/checkpointinterval`
: When writing, sets the interval in minutes for automatic checkpoints (see `mosrun(1)`). A value of 0 cancels automatic checkpoints. The maximal value allowed is 10000000. Note that writing has a side effect of reseting the time left to the next checkpoint. Thus, writing too frequently is not recommended. (rw)

`open("/proc/self/in_cluster", O_CREAT, node);` and
`open("/proc/self/in_partition", O_CREAT, node);`
: return 1 if the given `node` is in the same cluster/partition, 0 otherwise. The `node` can be either an unsigned, host-order IP address, or a node-number (listed in `/etc/mosix/userview.map`).

More functions are available through the `direct_communication(7)` feature.

The following information is available via the /proc file system for everyone to read (not just within the MOSIX run-time environment):

`/proc/{pid}/from`
: The IP address (a.b.c.d) of the process' home-node ("0" if a local process).

`/proc/{pid}/where`
: The IP address (a.b.c.d) where the process is runing ("0" if running here).

`/proc/{pid}/class`
: The class of the process.

`/proc/{pid}/origipid`
: The original PID of the process on its home-node ("0" if a local process).

`/proc/{pid}/freezer`
: Whether and why the process was frozen:

  0 Not frozen

  1 Frozen automatically due to high load.

  2 Frozen by the evacuation policy, to prevent flooding by arriving processes when clusters are disconnected.

  3 Frozen due to manual request.

  -66 This is a guest process from another home-mode (freezing is always on the home-node, hence not applicable here).

Attempting to read the above for non-MOSIX processes returns the string "-3".

STARTING MOSIX

To start MOSIX, run `/etc/init.d/mosix start`. Alternately, run `mosd`.

SECURITY

All nodes within a MOSIX cluster and multi-cluster grid must trust each other's super-user(s) - otherwise the security of the whole cluster or grid is compromized.

Hostile computers must not be allowed physical access to the internal MOSIX network where they could masquerade as having IP addresses of trusted nodes.

SEE ALSO

`mosrun`(1), `mosctl`(1), `migrate`(1), `setpe`(1), `mon`(1), `mosps`(1), `moskillall`(1), `mosq`(1), `bestnode`(1), `mospipe`(1), `direct_communication`(7), `topology`(7).

HISTORY

This is the 10-th version of MOSIX. More information is available at http://www.mosix.org/wiki/index.php/History_of_MOSIX

NAME

DIRECT COMMUNICATION − migratable sockets between MOSIX processes

PURPOSE

Normally, MOSIX processes do all their I/O (and most system-calls) via their home-node: this can be slow because operations are limited by the network speed and latency. `Direct communication` allows processes to pass messages directly between them, bypassing their 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 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, the network is not used at all.

To facilitate `direct communication`, each MOSIX process (running under `mosrun(1)`) can own a "mailbox". This mailbox can contain at any time up to 10000 unread messages of up to a total of 32MB. MOSIX Processes can send messages to mailboxes of other processes anywhere within the multi-cluster Grid (that are willing to accept them).

`Direct communication` 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.

`Direct communication` guarantees that the order of messages per receiver is preserved, even when the sender(s) and receiver migrate - no matter where to and how many times they migrate.

SENDING MESSAGES

To start sending messages to another process, use:

```
them = open("/proc/mosix/mbox/{a.b.c.d}/{pid}", 1);
```

where `{a.b.c.d}` is the IP address of the receiver's home-node and `{pid}` is the process-ID of the receiver. To send messages to a process with the same home-node, you can use `0.0.0.0` instead of the local IP address (this is even preferable, because it allows the communication to proceed in the rare event when the home-node is shut-down from its cluster).

The returned value (`them`) is not a standard (POSIX) file-descriptor: it can only be used within the following system calls:

```
w = write(them, message, length);
fcntl(them, F_SETFL, O_NONBLOCK);
fcntl(them, F_SETFL, 0);
dup2(them, 1);
dup2(them, 2);
close(them);
```

Zero-length messages are allowed.

Each process may at any time have up to 128 open `direct communication` file-descriptors for sending messages to other processes. These file-descriptors are inherited by child processes (after `fork(2)`).

When `dup2` is used as above, the corresponding file-descriptor (1 for standard-output; 2 for standard-error) is associated with sending messages to the same process as `them`. In that case, only the above calls (`write`, `fcntl`, `close`, but not `dup2`) can then be used with that descriptor.

RECEIVING MESSAGES

To start receiving messages, create a mailbox:

```
my_mbox = open("/proc/mosix/mybox", O_CREAT, flags);
```

where `flags` is any combination (bitwise OR) of the following:

1 Allow receiving messages from other users of the same group (GID).
2 Allow receiving messages from all other users.
4 Allow receiving messages from processes with other home-nodes.
8 Do not delay: normally when attempting to receive a message and no fitting message was received, the call blocks until either a message or a signal arrives, but with this flag, the call returns immediately a value of -1 (with `errno` set to EAGAIN).
16 Receive a `SIGIO` signal (See `signal(7)`) when a message is ready to be read (for assynchroneous operation).
32 Normally, when attempting to read and the next message does not fit in the read buffer (the message length is bigger than the `count` parameter of the `read(2)` system-call), the next message is truncated. When this bit is set, the first message that fits the read-buffer will be read (even if out of order): if none of the pending messages fits the buffer, the receiving process either waits for a new message that fits the buffer to arrive, or if bit 8 ("do not delay") is also set, returns -1 with `errno` set to EAGAIN.
64 Treat zero-length messages as an end-of-file condition: once a zero-length message is read, all further reads will return 0 (pending and future messages are not deleted, so they can still be read once this flag is cleared).

The returned value (`my_mbox`) is not a standard (POSIX) file-descriptor: it can only be used within the following system calls:

```
r = read(my_mbox, buf, count);
r = readv(my_mbox, iov, niov);
dup2(my_mbox, 0);
close(my_mbox);
ioctl(my_mbox, SIOCINTERESTED, addr);
```
(see FILTERING below)

Reading `my_mbox` always reads a single message at a time, even when `count` allows reading more messages. A message can have zero-length, but `count` cannot be zero.

A `count` of -1 is a special request to test for a message without actually reading it. If a message is present for reading, `read(my_mbox, buf, -1)` returns its length - otherwise it returns -1 with `errno` set to `EAGAIN`.

unlike in "SENDING MESSAGES" above, `my_mbox` is NOT inherited by child processes.

When `dup2` is used as above, file-descriptor 0 (standard-input) is associated with receiving messages from other processes, but only the `read`, `readv` and `close` system-calls can then be used with file-descriptor 0.

Closing `my_mbox` (or `close(0)` if `dup2(my_mbox, 0)` was used - whichever is closed last) discards all pending messages.

To change the `flags` of the mailbox without losing any pending messages, open it again (without using `close`):

```
my_mbox = open("/proc/mosix/mybox", O_CREAT, new_flags);
```

Note that when removing permission-flags (1, 2 and 4) from `new_flags`, messages that were already sent earlier will still arrive, even from senders that are no longer allowed to send messages to the current process. Re-opening always returns the same value (`my_mbox`) as the initial `open` (unless an error occurs and -1 is returned). Also note that if `dup2(my_mbox, 0)` was used, `new_flags` will immediately apply to file-descriptor 0 as well.

Extra information is available about the latest message that was read (including when the `count` parameter of the last `read()` was -1 and no reading actually took place). To get this information, you should first define the following macro:

```
static inline unsigned int GET_IP(char *file_name)
{
        int ip = open(file_name, 0);
        return((unsigned int)((ip==-1 && errno>255) ?
        -errno: ip));
}
```

To find the IP address of the sender's home, use:

```
sender_home = GET_IP("/proc/self/sender_home");
```

To find the process-ID (PID) of the sender, use:

```
sender_pid = open("/proc/self/sender_pid", 0);
```

To find the IP address of the node where the sender was running when the message was sent, use:

```
sender_location = GET_IP("/proc/self/sender_location");
```

(this can be used, for example, to request a manual migration to bring together communicating processes to the same node)

To find the length of the last message, use:

```
bytes = open("/proc/self/message_length", 0);
```

(this makes it possible to detect truncated messages: if the last message was truncated, `bytes` will contain the original length)

FILTERING

The following facility allows the receiver to select which types of messages it is interested to receive:

```
struct interested
{
        unsigned char conditions; /* bitmap of conditions */
        unsigned char testlen;    /* length of test-pattern (1-8 bytes) */
        short pid;                /* Process-ID of sender */
        unsigned int home;        /* home-node of sender (0 = same home) */
        int minlen;               /* minimum message length */
        int maxlen;               /* maximum message length */
        int testoffset;           /* offset of test-pattern within message */
        unsigned char testdata[8]; /* expected test-pattern */
};
```

```
/* conditions: */
#define INTERESTED_IN_PID     1
#define INTERESTED_IN_HOME    2
#define INTERESTED_IN_MINLEN      4
#define INTERESTED_IN_MAXLEN      8
#define INTERESTED_IN_PATTERN     16

struct interested filter;

#define SIOCINTERESTED        0x8985
```

A call to:

```
ioctl(my_mbox, SIOCINTERESTED, &filter);
```

starts applying the given `filter`, while a call to:

```
ioctl(my_mbox, SIOCINTERESTED, NULL);
```

cancels the filtering. Closing `my_mbox` also cancels the filtering (but re-opening with different flags does not cancel the filtering).

When filtering is applied, only messages that comply with the filter are received: if there are no complying messages, the receiving process either waits for a complying message to arrive, or if bit 8 ("do not delay") of the `flags` from `open("/proc/self/mybox", O_CREAT, flags)` is set, `read(my_mbox,...)` and `readv(my_mbox,...)` return -1 with `errno` set to EAGAIN. Filtering can also be used to test for particular messages using `read(my_mbox, buf, -1)`.

Different types of messages can be received simply by modifying the contents of the `filter` between calls to `read(my_mbox,...)` (or `readv(my_mbox,...)`).

`filter.conditions` is a bit-map indicating which condition(s) to consider:

When `INTERESTED_IN_PID` is set, the process-ID of the sender must match `filter.pid`.

When `INTERESTED_IN_HOME` is set, the home-node of the sender must match `filter.home` (a value of 0 can be used to match senders from the same home-node).

When `INTERESTED_IN_MINLEN` is set, the message length must be at least `filter.minlen` bytes long.

When `INTERESTED_IN_MAXLEN` is set, the message length must be no longer than `filter.maxlen` bytes.

When `INTERESTED_IN_PATTERN` is set, the message must contain a given pattern of data at a given offset. The offset within the message is given by `filter.testoffset`, the pattern's length (1 to 8 bytes) in `filter.testlen` and its expected contents in `filter.testdata`.

ERRORS

`Sender errors:`

ENOENT
: Invalid pathname in `open`: the specified IP address is not part of this cluster/Grid, or the process-ID is out of range (must be 2-32767).

ESRCH
: No such process (this error is detected only when attempting to send - not when opening the connection).

EACCES
: No permission to send to that process.

ENOSPC
: Non-blocking (O_NONBLOCK) was requested and the receiver has no more space to accept this message - perhaps try again later.

ECONNABORTED
: The home-node of the receiver is no longer in our multi-cluster Grid.

EMFILE
: The maximum of 128 `direct communicaiton` file-descriptors is already in use.

EINVAL When opening, the second parameter does not contain the bit "1"; When writing, the length is negative or more than 32MB.

ETIMEDOUT
: Failed to establish connection with the mail-box managing daemon (`postald`).

ECONNREFUSED
: The mail-box managing (`postald`) refused to serve the call (probably a MOSIX installation error).

EIO Communication breakdown with the mail-box managing daemon (`postald`).

`Receiver errors:`

EAGAIN
: No message is currently available for reading and the "Do not delay" flag is set (or `count` is -1).

EXFULL
: Messages were possibly lost (usually due to insufficient memory): the receiver may still be able to receive new messages.

ENOMSG
: The receiver had insufficient memory to store the last message. Despite this error, it is still possible to find out who sent the last message and its original length.

EINVAL One or more values in the filtering structure are illegal or their combination makes it impossible to receive any message (for example, the offset of the data-pattern is beyond the maximum message length).

`Errors that are common to both sender and receiver:`

EINTR Read/write interrupted by a signal.

ENOMEM
: Insufficient memory to complete the operation.

EFAULT
: Bad read/write buffer address.

ENETUNREACH
: Could not establish a connection with the mail-box managing deamon (`postald`).

ECONNRESET
Connection lost with the mail-box managing daemon (`postald`).

POSSIBLE APPLICATIONS

The scope of `direct communication` is very wide: almost any program that requires communication between related processes can benefit. Following are a few examples:

1. Use `direct communication` within standard communication packages and libraries, such as MPI.
2. Pipe-like applications where one process' output is the other's input: write your own code or use the existing `mospipe(1)` MOSIX utility.
3. `Direct communiction` can be used to implement fast I/O for migrated processes (with the cooperation of a local process on the node where the migrated process is running). In particular, it can be used to give migrated processes access to data from a common NFS server without causing their home-node to become a bottleneck.

LIMITATIONS

Processes that are involved in direct communication (having open file-descriptors for either sending or receiving messages) cannot be checkpointed and cannot execute `mosrun` recursively or `native` (see `mosrun(1)`).

SEE ALSO

`mosrun`(1), `mospipe`(1), `mosix`(7).

NAME

MOSCTL - Miscellaneous MOSIX functions

SYNOPSIS

mosctl stay
mosctl nostay
mosctl lstay
mosctl nolstay
mosctl block
mosctl noblock
mosctl logmap
mosctl nologmap
mosctl expel
mosctl bring
mosctl shutdown
mosctl isolate
mosctl rejoin [**{maxguests}**]
mosctl gridguests [**{maxguests}**]
mosctl opengrid [**{maxguests}**]
mosctl closegrid
mosctl cngpri {partner} {newpri} [**{partner2} {newpri2}**]...,
mosctl whois [**{node_number}|IP-address|hostname**]
mosctl status [**{node_number}|IP-address|hostname**]
mosctl localstatus
mosctl rstatus [**{node_number}|IP-address|hostname**]

DESCRIPTION

Most Mosctl functions are for MOSIX administration, available only to the Super-User, except the whois, status and rstatus functions which provide information to all users.

mosctl stay prevents processes from migrating away automatically: mosctl nostay cancels this state.

mosctl lstay prevents local processes from migrating away automatically, but still allows guest processes to leave: mosctl nolstay cancels this state.

mosctl block prevents guest processes from moving in: mosctl noblock cancels this state.

mosctl logmap tells the kernel to log the MOSIX map of nodes to the console (and/or the Linux kernel-logging facility) whenever it changes (this is the default). mosctl nologmap stops logging such changes.

mosctl expel expels all guest processes. It does not return until all guest processes are moved away (it can be interrupted, in which case there is no guarantee that all guest processes were expelled).

mosctl bring brings back all processes whose home-node is here. It does not return until all these processes arrive back (it can be interrupted, in which case there is no guarantee that all the processes arrived back).

mosctl shutdown shuts down MOSIX. All guest processes are expelled and all processes whose home-node is here are brought back, then the MOSIX configuration is turned off.

`mosctl isolate` disconnects the cluster from the grid, bringing back all migrated processes whose home-node is in the disconnecting cluster and sending away all guest processes from other clusters. To actually disconnect a cluster, this command must be issued on all the nodes of that cluster.

`mosctl rejoin` cancels the effect of `mosctl isolate`: an optional argument sets the number of guest processes that are allowed to move to this node or run here from outside the local cluster. When this argument is missing, no guest processes from outside the cluster will be accepted.

`mosctl gridguests` prints the maximum number of guests that are allowed to migrate to this node from other clusters. `mosctl gridguests arg,` with a numeric argument `arg,` sets that maximum.

`mosctl opengrid` sets the maximum number of guest processes from outside the local cluster to its argument. If no further argument is provided, that value is taken from `/etc/mosix/maxguests` and in the absence of that file, it is set to 8 times the number of processors. `mosctl closegrid` sets that maximum to 0 - preventing processes from other clusters to run on this node.

`mosctl cngpri` modifies the priority of one or more grid-partners in `/etc/mosix/partners` (See `mosix(7)`). While it is also possible to simply edit the files in `/etc/mosix/partners`, using `mosctl cngpri` is easier and the changes take effect immediately, whereas when editing those files manually, the changes may take up to 20 seconds.

`mosctl whois`, depending on its argument, converts host-names and IP addresses to node numbers or vice-versa.

`mosctl status` outputs useful and user-friendly information about a given node. When the last argument is omitted, the information is about the local node.

`mosctl localstatus` is like `status`, but adds more information that is only available locally.

`mosctl rstatus` output raw information about a given node. When the last argument is omitted, the information is about the local node. This information consists of 11 integers:

1. `status:` a bit-map, where bits have the following meaning:

   1 The node is currently part of our MOSIX configuration.
   2 Information is available about the node.
   4 The node is in "stay" mode (see above).
   8 The node is in "lstay" mode (see above).
   16 The node is in "block" mode (see above).
   64 The node may accept processes from here.
   Reasons for this bit to NOT be set include:
   - We do not appear in that node's map.
   - That node is configured to block migration of processes from us.
   - Our configuration does not allow sending processes to that node.
   - That node is currently running higher-priority MOSIX processes.
   - That node is currently running MOSIX processes with the same priority as our processes, but is not in our cluster and already reached its maximum number of allowed guest-processes.
   - That node is blocked.

512 The information is not too old.
1024 The node prefers processes from here over its current guests.
2048 The node is a 64-bit computer.

2. `load:` a value of 100 represents a standard load unit.
3. `availability:` The lower the value the more available that node is: in the extremes, 65535 means that the node is available to all while 0 means that generally it is only available for processes from its own cluster.
4. `speed:` a value of 10000 represents a standard processor (Pentium-IV at 3GHz).
5. `ncpus:` number of processors.
6. `frozen:` number of frozen processes.
7. `utilizability:` a percentage - less than 100% means that the node is under-utilized due to swapping activity.
8. `available memory:` in pages.
9. `total memory:` in pages.
10. `free swap-space:` in 0.1GB units.
11. `total seap-space` in 0.1GB units.
12. `privileged memory:` in pages - pages that are currently taken by less privileged guests, but could be used by clusters of higher privilege (including this node when "1024" is included in the `status` above).
13. `number of processes:` only MOSIX processes are counted and this count could differ from the load because it includes inactive processes.

SEE ALSO

`mosix`(7).

NAME

SETPE - Configure the local cluster

SYNOPSIS

setpe [-m mapfile] [-p our.ip.x.y] [-f [{feature1} [,{feature2} . . .]]]
setpe -[r|R]
setpe -off

DESCRIPTION

Setpe defines the configuration of the local MOSIX cluster.

The cluster map (see mosix(7)) is obtained from /etc/mosix/mosix.map - unless a different file is specified by the -m argument.

The local IP address to be used by MOSIX is either taken from the ifconfig utility; provided by /etc/mosix/mosip; or specified by the -p argument.

The node features (see topology(7)) are either absent; provided by /etc/mosix/myfeatures; or supplied as a (comma-separated) list by the -f argument.

setpe -r reads the current cluster map.
setpe -R reads the current map of the whole multi-cluster grid.
setpe -off disables MOSIX.

Anyone can read the MOSIX configuration, but only the Super-User can modify it.

SEE ALSO

topology(7), mosix(7).

NAME

TOPOLOGY − incorporating networking overheads in MOSIX

TUNING

MOSIX can make better migration decisions when it has a good estimate of the overheads involved in running processes away from home: the program `tune`, with its front-end `tune_mosix`, can be used to measure 26 different constants that reflect those overheads between any given two nodes.

Those contants depend on various factors such as network speed and latency, processor type, memory type, network card, whether a VPN layer is used on top of the IP protocol, etc.

INTRODUCING TOPOLOGY

The overheads of running a process away from its home may not be uniform across the cluster or multi-cluster grid: the `topology` is therefore defined as a set of overhead constants measured between the local node and a subset of other nodes. MOSIX supports up to 10 topologies, allowing each node to define up to 10 sets of overhead constants, measured between itself and different sets of nodes in the cluster and/or multi-cluster grid.

MOSIX comes with a built-in single default topology - a set of pre-measured constants that applies uniformly to all nodes. To override this default, the system-administrator needs to create the file: `/etc/mosix/mostune` (once that file is created or modified, MOSIX will automatically update its topology within one minute).

Each line in `/etc/mosix/mostune` should contain 29 space-separated integers: 26 are the overhead constants (generated by `tune_mosix`) and 3 are topological conditions (see below) that describe to which node(s) those overhead constants apply (the last line can have only 26 constants, making it unconditional).

To decide which overhead constants apply for a given node, MOSIX scans the above conditions, starting with the first line and proceeding down the list until a condition is found that is satisfied by the given node (if no condition is satisfied, the first line is selected).

TOPOLOGICAL CONDITIONS

Topological conditions consist of three numbers (unsigned 32-bit integers), named: `FIRST`, `LAST`, and `FEATURES`. FIRST and LAST are IP addresses, while FEATURES is a bitmap (1st symbol in `/etc/mosix/features` is 1, 2nd symbol is 2, 3rd symbol is 4, etc.). The IP addresses are represented as an unsigned integer in host order, so for example, IP address 123.45.67.89 is represented as:
((123∗256+45)∗256+67)∗256+89 = 2066563929

To test whether a given node satisfies a condition, we consider both its IP address ("IP") and its features ("F"): the features are configured in `/etc/mosix/myfeatures` (see below) and are 0 if that file does not exist.

The table below covers all the 5 possible combinations of FIRST and LAST, describing when a condition is satisfied by a given node:

`FIRST == LAST == 0`	Always: unconditional
`FIRST == 0; LAST != 0`	(F & FEATURES) != 0

`LAST == 0; FIRST != 0`	IP != FIRST && ~(F & FEATURES)
`0 < FIRST <= LAST`	FIRST <= IP <= LAST || (F & FEATURES)
`FIRST > LAST > 0`	(IP < FIRST || IP > LAST) && ~(F & FEATURES)

CONFIGURING FEATURES

The features of a node are listed in the file `/etc/mosix/myfeatures` by a comma-separated list of symbols, selected out of the 32 symbols in the file `/etc/mosix/features`.

These symbols have no particular meaning other than to aid in constructing useful combinations of topological conditions. It is up to the multi-cluster system-administrators to agree between them on conventional meanings to those symbols.

System-administrators are also allowed to modify those symbols if they wish, provided that they keep `/etc/mosix/features` the same throughout the multi-cluster Grid (if they do so, they must remember to restore that file after upgrading to a new version of MOSIX).

SEE ALSO

`mosix`(7).