NAME

mosix – LINUX enhancements for cluster computing

INTRODUCTION

Mosix is a set of enhancements of Linux for supporting cluster computing. The core of the MOSIX technology are adaptive resource sharing (load-balancing and memory ushering) algorithms and a preemptive process migration mechanism, that allow a cluster of workstations and servers (nodes) to work cooperatively as if part of a single system. Mosix for Linux was specifically designed for a cluster of computing nodes of the same family (at this stage, only the Intel-x86 family is supported, but the plans are to support other platforms). The cluster may include a large number of nodes (up to 65535), which can be any combination of workstations, servers, single-processor or SMPs of any speed. `MOSIX` is implemented within the Linux kernel and it maintains 100% compatiblity with standard Linux, hence there is no need to modify any user-level files, programs or binaries. When activated, normal Linux processes may be allocated and migrate automatically and transparently to other nodes within the cluster in order to achieve a better resource usage. As the demands for resources change across the cluster, processes may migrate again, as many times as necessary, to continue optimizing the overall resource usage. Sysadmin and user manual-control of process migration is also available. `MOSIX` is particularly suitable for the execution of CPU-bound and other computational tasks with unpredictable resource usage and execution times.

REQUIREMENTS

1. All the nodes in the cluster must be connected by LANs that support TCP/IP under Linux.
2. All nodes must have a mathematical (floating-point) unit (built in for Pentiums and up, 80387 or Waitec for earlier models).
3. In Multiprocessor nodes (SMP), all the processors must be of the same speed.

CONFIGURATION

The file `/etc/mosix.map` defines the computer-nodes in the MOSIX cluster by mapping MOSIX node-numbers to IP addresses. Each line (other than blank or comment-lines beginning with a #) describes one range of nodes with consecutive IPs. Each line contains 3 fields:

`MOSIX-node-number` of the first node in the range (could be any small integer up to 65535).

`IP-address` – of the first node in the range, or its corresponding host-name as listed in "/etc/hosts".

`range-size` – the number of nodes in the range.

The maximum number of lines, i.e. ranges, is `MAX_MOSNET_ENTS` as defined in `include/linux/mosctl.h` (currently 256).

EXAMPLE: If you have 6 nodes, with 3 nodes in class-C internet addresses 200.200.120.xxx, 1 node in addresses 200.200.130.xxx and 2 nodes in addresses 200.200.140.xxx, then your `/etc/mosix.map` could have the following lines:

```
1       200.200.120.1   3
4       200.200.130.1   1
5       200.200.140.1   2
```

In the most common case, where the network to be used by MOSIX is the same standard network that is used by the cluster for other purposes, the MOSIX node-number of the local node is determined by the IP address within the list that matches the local host's name. If this is not the case, then the MOSIX node-number must written to `/etc/mospe`.

MOSIX can be activated or deactivated in the different run-levels in the standard way, by using the Linux run-level editor. MOSIX must be activated after the network is started and decativated BEFORE shutting down the network. This is achieved by giving MOSIX a higher start-priority than the network's of 10 and a lower stop-priority than the network's of 97.

While manual process-migration is resident in the kernel, the automatic load-balancing (if present) module, is automatically loaded when MOSIX is configured. The automatic load-balancing module is never removed automatically (use `rmmod mosix` to remove).

ADMINISTRATION

The system-administrator can control the behaviour of MOSIX by writing to files in the `/proc/mosix/admin` directory.

The most important file is the configuration: `/proc/mosix/admin/config`, but being a binary file, the easy way to use it is via the `setpe(1)` utility. Similarly, the node's number, written to `/proc/mosix/admin/mospe`, but accepted only under particular circumstances, should be left for `setpe(1)`.

Writing a "1" to `/proc/mosix/admin/block`, blocks the arrival of foreign (guest) processes into the local node. Writing a "0" allows guest processes in. To also send back guest processes that are ALREADY running on the local node, write "1" to `/proc/mosix/admin/expel`.

To prevent automatic migration of processes from the local node, write "1" to `/proc/mosix/admin/stay`. Writing a "0" enables outbound automatic migration. To prevent automatic migration of local processes, but still allow automatic outbound migration of guest processes, write "1" to `/proc/mosix/admin/lstay`. To also bring back local processes that ALREADY run on other nodes, write "1" to `/proc/mosix/admin/bring`. Writing a "0" to `/proc/mosix/admin/lstay` allows outbound automatic migration of local processes (unless a "1" was written to `/proc/mosix/admin/stay`).

Writing a "1" to `/proc/mosix/admin/quite` is necessary to produce accurate results while running the `tune_kernel` utility (see `tune(1)`): there is no need for this mode at any other time. Writing a "0" cancels the quiet-tuning mode. It is recommended to run `tune_kernel` once after installing MOSIX and every time you change the type of processor or network used by MOSIX.

The 14 (or more) tuning-parameters produced by the `tune_kernel` utility are normally included in the kernel under the file: "include/mos/dscosts.h". If for any reason you do not wish to modify that file - for example, if you want the same binary copy of the kernel to be shared among different nodes, or if you temporarily use a different network for MOSIX (which you tuned earlier), you can copy the 14 (or more) constants found in the file `/tmp/overheads` by `tune_kernel` as a space-separated list into `/proc/mosix/admin/overheads`. Alternately, if a file names `/etc/overheads` is present, it is copied to `/proc/mosix/admin/overheads` by the MOSIX initialization script.

MOSIX collects information about resources used by processes which decays in time. Although the decay policy may be specifically modified by each process, the defaults can be modified by the system-administrator: the interval between decays (in seconds) can be modified from its default of 2 seconds by writing any other positive number into `/proc/mosix/admin/decayinterval`. The amount of statistical information to keep every such interval is expressed as an integer value out of 1000. To modify the default, or slow decay, defaulting to 975, write any other value into `/proc/mosix/admin/slowdecay`. To modify the default for processes who elect to use the fast-decay, defaulting to 926, write any other value into `/proc/mosix/admin/fastdecay`. MOSIX enforces that at any particular time, the slow-decay value must be greater or equal to the fast-decay value.

MOSIX measures the processor's speed at boot-time, relative to a Pentium-2 at 400 MHz representing 10000 units of speed. Processor speeds do, however, vary for different benchmarks. If you have a good reason to believe that the typical applications that run on the cluster run faster (slower) on this node, then your system-

administrator may increase (decrease) the value of `/proc/mosix/admin/speed`.

When obtaining load-information from the kernel, especially by the `mon(1)` program, 100 units of load roughly represents a single CPU-bound process running on a single-processor standard-processor (eg. Pentium-2 400 MHz). If most or all of your nodes are of a different speed, you may wish to modify `/proc/mosix/admin/sspeed` accordingly, so that 100 units of load represent a single CPU-bound process running on a single-processor of your most typical computer, of a given speed. For example, if running on a node of your typical computer, you may type:
`cp /proc/mosix/admin/speed /proc/mosix/admin/sspeed`.

Most files in `/proc/mosix/admin` can also be read, to indicate their current values.

Even the system administrator may not kill or otherwise interfere with guest processes, other than sending them away, by writing the desired node-number to which they should be sent into `/proc/mosix/remote/{pid}/goto`. Unless there is a good reason to do otherwise, one should write a "0" to that file, indicating that the guest should move back to its home-node. Writing a "-1" will cause the guest to consider migration based on optimal placement.

INFORMATION

`Mosix` provides the following extra information per process:

`/proc/{pid}/where` tells where the process currently runs, 0 being its home-node.

`/proc/{pid}/lock` tells whether the process has been locked (1) or not (0) against automatic migration from its current node.

`/proc/{pid}/nmigs` tells how many times the process ever migrated.

`/proc/{pid}/cantmove` lists reasons (if any) why the process may not migrate away from its home-node. Such reasons include the use of shared-memory, direct access to the computer's bus, direct mapping to device-memory, having locked-memory, cloning where more than one process uses the same virtual-memory, system (kernel-mode) daemons, emulating in the VM86 mode, or real-time scheduling.

The following (limited) information is available about guest-processes via the `/proc/mosix/remote/{pid}` directory. It is purposefully that no identifying information, such as user-ID or the original process-ID on the home-node is given, but only enough information to indicate the amount of local resources used:

`/proc/mosix/remote/{pid}/from` tells where the process came from - as a MOSIX node-number, followed by the IP address.

`/proc/mosix/remote/{pid}/statm` provides the following memory statistics, as per local processes: total size, resident size, shared-memory size (always 0, or the process could not migrate), text pages, library pages, data pages, dirty pages.

`/proc/mosix/remote/{pid}/stats` provides selected CPU information in a "variable=value" form:

1. utime=user-time since arrived at this node.
2. cutime=sons/descendents user-time, provided those descendents never left this node.
3. nice=nice value (scheduling priority)
4. state="R" for running, "S" for being in any other state.
5. vsize=process' virtual size.
6. rss=number of resident pages.

7. nswap=number of page-swaps.

8. cnswap=number of page-swaps by descendents, provided those descendents never left this node.

Information about all configured nodes is available via the `/proc/mosix/nodes/{node-number}` directories. This includes:

`/proc/mosix/nodes/{node-number}/cpus`: the number of CPUs (eg. on SMP nodes) in the node.

`/proc/mosix/nodes/{node-number}/load`: the current load of the node (100 representing roughly one CPU-bound process on a standard computer with one processor).

`/proc/mosix/nodes/{node-number}/mem`: available memory on the node as MOSIX sees it.

`/proc/mosix/nodes/{node-number}/rmem`: available memory on the node as Linux believes.

`/proc/mosix/nodes/{node-number}/tmem`: total memory on the node.

`speed`: node's processor speed, 10000 representing the standard processor of Pentium-2 400 MHz.

`util`: node's utilizability, normally 100 times the number of CPUs. If less, than the node is in serious trouble of memory-thrashing.

`status`: The following bit-values ORed in:

1 MOSIX is configured - the only instance when this may not be set, is if we managed to get into the directory before the node was removed from the configuration.

2 The node is up and responding to MOSIX communications.

4 The node does not allow outbound automatic migration.

8 The node does not allow outbount automatic migration of its own processes.

16 The node does not accept guest processes.

32 The node is in special "quiet" mode.

64 The node is in special "tune" mode.

The same information is also available in condensed binary form for the use of `mon(1)` or similar future utilities via the `/proc/mosix/info` directory.

MANUAL CONTROL

A process may disable automatic migration for itself and its future descendents, effectively locking itself on the current node where it runs by writing a "1" to `/proc/self/lock`. Writing a "0", cancels this lock (for the process, but not for already-spawned descendants). Please note that even with the lock set, nothing can prevent a process from being sent back to its home-node under extreme circumstances, such as computer shut-downs, explicit requests by the system-administrator on either the home-node or the node where the process runs, or if the processes requests a type of resource that is only available on its home-node.

Only a process may lock or unlock itself, except that the super-user has the special privillege of unlocking processes that run on their home-node.

A process may migrate to any desired node by writing that node-number to `/proc/self/migrate`. Writing a "0" is equivalent to writing the number of the home-node, while writing a "-1" requests MOSIX to consider migration based on optimal placement. Writing to `/proc/self/migrate` overwrites the above lock. The "write" succeeds if and only if the migration is successful (a "-1" request always succeeds).

Writing to any other process' `/proc/{pid}/migrate` will fail. To request another process to migrate (you must have permission to send signals to that process), write a similar value to `/proc/{pid}/goto`. This should only be seen as an assynchroneous recommendation and the "write" is successful if permission is

granted. Except for the "-1" value, this request overrides the other process being in a locked state.

A process may advise MOSIX about the nature of the resources it is about to use via the `/proc/mosix/decay` directory:

Writing a "1" to `/proc/mosix/decay/clear` clears all the process' statistics, requesting MOSIX to turn a new leaf.

Writing a "1" to `/proc/mosix/decay/cpujob` advises MOSIX that the process is totally CPU-bound.

Writing a "1" to `/proc/mosix/decay/iojob` advises MOSIX that the process is totally I/O bound.

Writing a "1" to `/proc/mosix/decay/slow` advises MOSIX that the process wishes to decay its statistics slowly, thus rely upon a long-term history.

Writing a "1" to `/proc/mosix/decay/fast` advises MOSIX that the process wishes to decay its statistics fast, thus rely only upon short-term history.

Writing two space-separated integer values to `/proc/mosix/decay/own`, indicates the accurate decay policy desired by the process, with the first number, in the range of 0-1000, telling how much out of a 1000 units of the previous history to maintain each decay-cycle, occuring every number of seconds as indicated by the second number.

Because the nature of a process is redefined by the `execve(2)` system call (and its variants), not only its statistics are cleared, but the decay policy itself is restored to the default of slow-decay. A process wishing to secure its decay-policy across the `execve(2)` system call, should write "1" to `/proc/mosix/decay/exec`. Alternatively, if it wishes to maintain its decay-policy over a forthcoming `execve`, but not on subsequent calls, it should write "1" to `/proc/mosix/decay/execonce`. A process wishing to inherit its decay policy to its descendants (across the `fork(2)` system call), should write "1" to `/proc/mosix/decay/inherit`.

Writing 0 to `/proc/mosix/decay/exec`, `/proc/mosix/decay/execonce` and `/proc/mosix/decay/inherit` cancels those options. The decay modes, however: `cpujob`, `iojob`, `slow`, `fast` and `own`, are mutually exclusive - selecting the one cancels the others.

To find out the current decay policy, read the files: `/proc/mosix/decay/cpujob`, `/proc/mosix/decay/iojob`, `/proc/mosix/decay/slow`, `/proc/mosix/decay/fast`, `/proc/mosix/decay/exec`, `/proc/mosix/decay/execonce` and `/proc/mosix/decay/inherit`: A value of 1 indicates that the mode is set and a 0 indicates that it is clear. Reading `/proc/mosix/decay/own` will produce the two abovementioned decay parameters if in that mode, or two zeroes otherwise.

SECURITY

All nodes within the MOSIX cluster must trust each other's super-user, especially since the Linux super-user is allowed to send and receive direct IP-level network packets. This usually means that all nodes are in the same department (or even the same locked room) and under control of the same system-administrator, and that no untrusted personnel have direct access to the MOSIX network (by plugging their private laptops into an ethernet or ISDN socket on the wall).

The network used for MOSIX carries sensitive kernel-level communications, which could crash nodes or create any damage if fallen in the wrong hands. Therefore it must be made sure that no outsiders can masquerade as having an IP address that is defined in the MOSIX configuration file, pretending to be one of the trusted MOSIX nodes in the cluster. This is normally achieved by either a router or a firewall.

BUGS

Please report any bugs in `http://www.mosix.cs.huji.ac.il/bugs`.

SEE ALSO

`setpe`(1), `mon`(1), `tune`(1). `mosctl`(1), `migrate`(1).

HISTORY

This is the 7th version of MOSIX. The 6th version was developed for BSD/OS. The MOSIX book, see a reference in our web page, describes the previous releases.