MOSIX2 Tutorial

L. Amar, A. Barak, T. Maoz, E. Meiri, A. Shiloh
Department of Computer Science
The Hebrew University

http:// www . MOSIX . org

January 2009

About

This tutorial has 2 parts:

- The first part is intended for new users. It covers basic tools and operations such as monitors, how to run and control processes, view processes, view and control the queue and how to handle unsupported features
- The second part includes advanced topics such as freezing processes, checkpoint & recovery, running a large set of processes, I/O optimizations, running Matlab, running parallel jobs, configuration and management, and the programming interface

Part I: Basics

- **Tools**
  - **mon, mmon**
  - **mosrun, native**
  - **mosps**
  - **mosq**
  - **migrate**
- **Operations**
  - **Initial assignment**
  - **Running Linux processes**
- **Encountering unsupported features**

Detailed information about each command is available in the corresponding manual pages: man mosix | mosrun | mosps…

Monitors - seeing what is going on

***mon* and *mmon* are 2 monitors for viewing the status of the resources in each cluster and all the clusters**

- ***mon* – displays basic information (tty format) about resources in the local cluster**
- **To display type:**
  - **"l" – CPU load (relative)**
  - **"f" – Number of frozen processes**
  - **"m" - Memory (used + free), swap-space (used + free) – type consecutively**
  - **"u" - Utilization**
  - **"d/D" - Dead nodes**
  - **"h" – help for complete list of options**

mmon

- ***mmon*** **– includes all the options of *mon* and many more**
- **To display type (consecutively):**
  - **"l" - Load, load + frozen processes, frozen processes**
  - **"m" - Memory (used + free), swap-space, disk-space**
  - **"g" – multi-cluster status (running local/guests, priority)**
  - **"i" – Read I/O rate, Write rate, total I/O**
  - **"o" – network activity (in, out, total), RPC rate**
  - **"s" - CPU speed, status, MOSIX version, uptime**
  - **"d/D" - dead nodes**
  - **"h" – help for complete list of options**

Other *mmon* features

- **Can run on non-MOSIX nodes, e.g. your workstation**
  - ***mmon –h bmos-01*** **(node #1 in the bmos cluster)**
- **Display the status of several clusters (consecutively)**
  - ***Example: mmon –c amos-01, bmos-01, cmos-01***
    - **Use the > ,< keys to skip from one cluster to another**
- **Private color scheme using the file *~/.mmon.cfg***

mosrun – running MOSIX processes

- To run a program under the MOSIX discipline start it with *mosrun, e.g., mosrun myprog*
  - Such programs can migrate to other nodes
  - Example:

    > mosrun myprog 1 2 3 (run myprog, possibly with arguments)
- Programs that are not started by *mosrun* run in native Linux mode and CANNOT migrate
- A program that is started by *mosrun* and all its children remain under the MOSIX discipline
- MOSIX processes (that were started by *mosrun)* can use the *native* utility to spawn children that run in native Linux mode

Example: view the process migration

- Login to any node in a MOSIX cluster
- On one window run *mon* or *mmon*
- On another window start 2 CPU-intensive processes, e.g. the testload program (included in the MOSIX distribution):

  >mosrun testload &

  >mosrun testload &
- Observe in the *mon/mmon* window how the processes move across the cluster's nodes
- Type *moskillall* to stop the running processes

mosrun – node assignment options

- -r{hostname} – run on this host (node)
- -{a.b.c.d} – run on node with this IP
- -{n} – run on node number n
- -h – run on the home node
- -b – attempt to select the best node
- -jID1-ID2[,ID3-ID4] – run on a random node in the ranges
  ID1-ID2, ID3-ID4, …
- Examples:
  > *mosrun –rmos1 myprog* (run on node mos1)
  > *mosrun –b myprog* (run on the best node)
  > *mosrun –3 myprog 1 2 3* (run on node #3, with arg 1 2 3)

mosrun – where can processes migrate to

- -G – allows processes to migrate to nodes in other clusters
  - Otherwise, processes are confined to the local cluster
  - *-G{class}* – if class > 0 than a process is allowed to migrate to nodes in other clusters. Note that –G is equivalent to –G1
- -m{megabytes} – specifies the maximal amount of memory needed by your program, to prevent migration of processes to nodes that do not have sufficient free memory
- Beside migration, the –G and -m options also affect the initial assignment (-b flag) and queuing (see below)
- Example:
  > *mosrun –G –m1000 myprog* (allows *myprog* to run on other clusters, but only on nodes with at least 1GB of free memory)

mosrun - marking your processes

- The -J{Job Id} option of *mosrun* allows bundling for easy identification of several instances of *mosrun*
  - The "Job Id" is an integer (default value is 0)
  - Each user can assign their own "Job Ids"
- "Job Id" is inherited by all child processes
- "Job Id" can be viewed by *mosq and mosps*
- All jobs of a user with the same "Job Id" can be collectively killed (signaled) by *moskillall* and migrated by *migrate*
- Examples:

  > *mosrun –J20 myprog* (run *myprog* with Job_ID = 20)

  > *mosps –J20* (list only my processes with Job_ID = 20)

  > *moskillall –J20* (kill all my processes with Job_ID = 20)

mosrun – run batch jobs

mosrun can send batch jobs to other nodes in the local cluster

- There are 2 types:
  - -E – runs native Linux processes
  - -M – runs MOSIX processes but their home node can be a different node in the local cluster
  - The combination of -E and –b attempts to assign a Linux job to the best available node
  - The combination of -M and –b attempts to assign both the home-node and where the process starts to run, to the best available nodes
  - The -E/*{dir}* and *–M/{dir}* specify the directory where the job should run
    - By default it will run in the directory named as the current directory in this node
- Examples:

  > mosrun –E –rmos4 mylinuxprog

  > mosrun –M/tmp –b mymosixprog

mosps – view MOSIX processes

mosps **(like *ps*) provides information about your MOSIX processes (and many standard *ps* fields, see the next slide), including:**

- ***WHERE*** **– where (or in what special state) is your process**
- ***FROM*** **– from where this process came**
- ***ORIGPID*** **– original pid at home node**
- ***CLASS*** **– class of the process**
  - **For MOSIX processes defined by *mosrun –G{class)***
  - **Other values are *batch* and *native***
- ***FRZ*** **– if frozen why: A- auto frozen (due to load); E- expelled; *M*- manually; - - NOTFROZEN; N/A – can't be frozen; *DUE* – when user expects process to complete**
- ***NMIGS*** **– number of migrations so far**

mosps - options

mosps **supports most standard *ps* flags**

- **Special *mosps* flags are:**
  - ***-I*** **– nodes displayed as IP addresses**
  - ***-h*** **– nodes displayed as host names**
  - ***-M*** **– display only last component of host name**
  - ***-L*** **– show only local processes**
  - ***-O*** **– Show only local processes that are away**
  - ***-n*** **– display NMIGS**
  - **-V – show only guest processes**
  - ***-P*** **– display ORIGPID**
  - ***-D*** **– display DUE**
  - ***-J{JobID}*** **– show only processes with this Job ID (see advanced features)**

mosps – example

```
> mosps -AMn

 PID   WHERE  FROM CLASS FRZ NMIGS TTY        CMD
24078 cmos-18 here local  -    1   pts/1 mosrun -b testload
24081 here    here local  -    0   pts/1 mosrun -b testload
24089 cmos-16 here local  -    1   pts/1 mosrun -b testload
30145 queue   here  N/A  N/A  N/A  pts/1 mosqueue -b testload
30115 here    here local  M    0   pts/1 mosrun testload
30253 here    mos3 local N/A  N/A  ?     /sbin/remote
```


mosrun - queuing

- With the -q flag, *mosrun* places the job in a queue
- Jobs from all the nodes in each cluster share one queue
  - The prevailing queue policy is First-come-first-serve, with several exceptions
  - Users can assign priorities to their jobs, using the *–q{pri}* option
    - The lower the value of *pre* the higher priority
    - The default priority is 50. It can be changed by the sysadmin
    - Running jobs with pri < 50 should be coordinated with the cluster's manager
- Examples:

  > mosrun –q –b –m1000 myprog (queue a MOSIX program to run in the cluster)

  > mosrun –q60 –G –b -J1 myprog (queue a low priority job to run in the multi-cluster)

  >mosrun –q30 –E –m500 myprog (queue a high priority batch job)

mosq – view and control the queue

- *mosq list* – **list the jobs waiting in the queue**
- *mosq listall* – **list jobs already running from the queue and jobs waiting in the queue**
- *Mosq delete {pid}* – **delete a waiting job from the queue**
- *Mosq run {pid}* – **run a waiting process now**
- *Mosq cngpri {newpri}{pid}* – ***change the priority of a waiting job***
- *Mosq advance {pid}* – ***move a waiting job to the head of its priority group within the queue***
- *Mosq retard {pid}* – ***move a waiting job to the end of its priority group within the queue***

For more options, see the mosq manual

mosq - example

```
> mosq listall
  PID  USER  MEM(MB) GRID  PRI FROM           COMMAND
21666  lior       -   NO   RUN here           mosrun -b testload
21667  lior       -   NO    50 here           mosrun -b testload
21155  lior       -   NO    50 cmos-17        mosrun testload
> mosq cngpri 20 21155
> mosq listall
  PID USER  MEM(MB) GRID  PRI FROM            COMMAND
21666 lior       -   NO   RUN here            mosrun -b testload
21155 lior       -   NO    20 cmos-17         mosrun testload
21667 lior       -   NO    50 here            mosrun -b testload
```

migrate – control MOSIX processes

- *migrate{pid} {node-number | ip-address | host-name}* – move your process to the given node
- *migrate{pid} home* - request your process to return home
- *migrate{pid} freeze* – request to freeze your process
- *migrate{pid} continue* – unfreeze your process
- *migrate{pid} checkpoint* – request your process to checkpoint
- *migrate{pid} checkstop* – request your process to checkpoint and stop
- *migrate{pid} exit* – checkpoint your process and exit

For more options, see the migrate manual

Encountering unsupported features

Some utilities and libraries use features that are not supported by MOSIX

Usually, such features are not critical

If you run a utility and encounter a message such as:

- MOSRUN: Shared memory (MAP_SHARED) not supported under MOSIX

or

- MOSRUN: system-call 'futex' not supported under MOSIX

Try to use the "*-e*" flag of *mosrun* to bypass the problem

- Example: instead of mosrun ls -la use mosrun –e ls –la

Part II: Advanced topics

- Freezing processes
- Checkpoint/Recovery
- running a large set of processes
- Before using the multi-cluster
- I/O optimizations
- Running Matlab
- Parallel jobs
- What is not supported
- Configuration and management
- The programming interface

Freezing processes

MOSIX process can be frozen, usually to prevent memory threshing

- **While frozen, the process is still alive but is not running - its memory image is left in the disk**
- **Frozen processes do not respond to caught signals**

- **Processes can be frozen in 3 ways:**
  - **Manually – upon user's request**
  - **When being evacuated from a remote node that was reclaimed**
  - **When the local load is too high for a certain class of processes**

Freezing - example

- **Running some processes**
  - **6 X "mosrun –J1 testload –m 2"**
- **Using the migrate command to freeze all the processes from job 1**
  - **migrate –J1 freeze**
- **Using migrate to "continue" frozen processes**
  - **migrate –J1 continue**
- **Note that it is possible to freeze/continue individual processes as well**

Checkpoint/recovery

Most CPU-intensive MOSIX processes can be checkpointed, then recovered from that point

- **In a checkpoint, the image of a process is saved to a file**
- **Processes with open pipes or sockets, or with setuid/setgid privileges cannot be checkpointed**
- **Processes that wait indefinitely, e.g. for terminal/pipe/socket I/O or another process, will only produce a checkpoint once the wait is over**

- **The following processes may not run correctly after a recovery**:
  - Processes that rely on process id or parent-child relations
  - Processes that communicate with other processes
  - Processes that rely on timers and alarms or cannot afford to lose signals

mosrun - how to checkpoint

- *-C{base-filename}* – **specifies file names (with extensions .1, .2, .3,…) where checkpoints are to be saved**
- *-N{max}* – **specifies the maximum number of checkpoints to produce before re-cycling extensions**
- *-A{min}* – **produces a checkpoint every given number of minuts**
  - **Checkpoint can also be triggered by the program itself (see the MOSIX manual) and by the user (see the migrate manual)**
- **Example:**

  > *mosrun –C/tmp/myckpt –A20 myprog* **(create a checkpoint every 20 minuts to files: /tmp/myckpt.1, /tmp/myckpt.2, …)**

  > *mosrun myprog* **(whenever the user requests a checkpoint manually, a checpoint will be written to files: ckpt.{pid}.1, ckpt.{pid}.2, …)**

mosrun – how to recover

- **Use *-I{file}* to view the list of files that where used by the process at the time of checkpoint**
- **Use *-R{file}* to recover and continue the program from a given checkpoint**
  - **With *-R{file}* you can also use *–O{fd1=filename1, fd2=filename2,…}* - to use the given file location(s) per file-descriptor instead of the original location(s)**
- **Examples:**

  > *mosrun –I/tmp/ckpt.1*

  Standard Input (0): special file, Read-Write

  Standard Output(1): special file, Read-Write

  Standard Error (2): special file, Read-Write

  File-Descriptor #3: /usr/home/me/tmpfile, offset=1234, Read-Write

  > *mosrun –R/tmp/ckpt.1 –O3=/user/home/me/oldtmpfile*

mosrun - running a large set of processes

- The -S{maxjobs} option runs under *mosrun* multiple command-lines from the file *commands-file*
  - Command-lines are started in the order they appear
  - Each line contains a program and its given *mosrun* arguments
    - Example of a commands-file:

```
my_program –a1 –if1 –of1
my_program –a2 –if2 –of2
my_program –a3 –if3 –of3
```

- While the number of command-lines is unlimited, *mosrun* will run up to *maxjobs* command-lines concurrently at any time
  - Whenever one process finish, a new line will start

Before running processes on the multi-cluster

- Some programs do not perform well on the cluster (multi-cluster) due to various overheads and improper tuning. To check a specific program, run a sample copy 3 (4) times on identical nodes and measure the times:
  - As a regular Linux process (without mosrun)
  - As a non-migrated MOSIX process in the local node
    - mosrun –h –L …
  - As a migrated MOSIX process to a remote node in the local cluster
    - mosrun –r<node-name> -L …
  - In case of multi-cluster, repeat the last test to a remote node in another cluster
- The running times of the program should increase gradually by few percents but not significantly
  - If this is not the case you should investigate the reasons
  - For example: use *strace* to see if the process is doing I/O in an efficient way (a reasonable system call rate)

Example: Matlab with an external library

Matlab needs to load a special external library

The measured run times are:

- **As a Linux process (no mosrun): 12:06 Min.**
- **With mosrun, on the home node: 12:27 Min.**
- **With mosrun in a remote node in the same cluster: 12:47 Min.**
- **With mosrun on a remote node in another cluster: 13:04 Min.**

The slowdown is caused by the time to load the external library, Matlab performs many system calls during this phase

- **In longer jobs, the slowdown is less significant**

Using *strace* to see what your process is doing

- **strace = trace system calls and signals**
  - **Allow you to see which system calls are used by your process**
- **Use the *–c* argument of *strace* to print a summary of all the system calls used**
- **A special version of *strace* can also show a histogram of the block sizes**
  - **/cs/mosix/sbin/strace**

Example: *blas* with too many *futex*

% time	seconds	usecs/call	calls	errors	syscall
99.78	16.847495	17	1019960	2712	futex
0.17	0.028892	95	303	275	open
0.01	0.002178	128	17		munmap
0.01	0.002096	40	53		read
0.01	0.001843	29	64	60	stat64
0.01	0.000993	7	143		brk
0.00	0.000474	43	11	11	access
0.00	0.000370	10	38		old_mmap
0.00	0.000300	12	25		mmap2
0.00	0.000236	9	27		close
0.00	0.000160	6	28		fstat64
0.00	0.000115	58	2		write
0.00	0.000087	12	7		mprotect
0.00	0.000073	6	13		times
0.00	0.000056	5	11		_llseek
0.00	0.000025	5	5		gettimeofday
0.00	0.000020	20	1		clone
0.00	0.000011	11	1		_sysctl
0.00	0.000010	5	2		rt_sigaction
100.00	16.885463		1020716	3058	total

blas using a different implementation

Allocation memory

% time	seconds	usecs/call	calls	errors	syscall
78.32	0.030613	99	309	281	open
5.54	0.002167	22	98	83	stat64
5.33	0.002083	130	16		munmap
4.78	0.001868	37	51		read
1.76	0.000689	5	138		brk
0.87	0.000339	42	8	8	access
0.84	0.000329	8	40		old_mmap
0.68	0.000267	9	29		fstat64
0.46	0.000178	7	27		close
0.37	0.000143	72	2		write
0.28	0.000110	11	10		mprotect
0.28	0.000108	5	21		mmap2
0.13	0.000049	49	1		uname
0.12	0.000046	4	13		times
0.09	0.000034	3	11		_llseek
0.05	0.000018	18	1		set_thread_area
0.04	0.000017	3	5		gettimeofday
0.02	0.000008	8	1		_sysctl
0.02	0.000007	4	2		rt_sigaction
0.01	0.000005	5	1		futex
100.00	0.039088		787	372	total

I/O considerations

- MOSIX programs that issue a large number of system-calls or perform intensive I/O relative to the amount of computation are expensive because those operations are emulated
- When a process is running in a remote node, there is also overhead of sending those operations to the home-node over the network
- Such processes will automatically be migrated back to the home-node, to eliminate the communication overhead

Using *strace* to see the I/O granularity

```
/cs/mosix/sbin/strace -c testload -f big --write 4 --iosize  10

Allocation memory
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 97.73    0.017740           7      2560           write
  1.55    0.000282          47         6         2 open
  0.01    0.000002           2         1           getpid
   …       …                 …         …           …
  0.01    0.000002           2         1           set_thread_area
------ ----------- ----------- --------- --------- ----------------
100.00    0.018152                  2598         5 total


IO Summary:
-----------------------------------------------
    sys-call            MB
-----------------------------------------------
        read         0.001      (64B :    1) (512B :    1)
       write        10.000      (4K : 2560)
```

Improving the I/O performance

- **Consider running such programs as native Linux, using the "-E" flag of *mosrun* for the main process, or "native" for child processes**
- **"gettimeofday()" can be a very frequent system-call: use the "-t" flag to get the time from the hosting node instead of the home-node**
- **Try to perform I/O in larger chunks (less system-calls)**
- **Avoid unnecessary system-calls (such as "lseek()" - use "pread/pwrite" instead)**
- **The "-c" flag prevents bringing home processes due to system-calls: it should be used when the I/O phase is expected to be short. For programs with more complex patterns of alternating CPU-intensive and I/O periods, learn about the "-d" option**

Running jobs with intensive I/O

- Use the **"network display"** and **"disk display"** screens of ***mmon*** to see what you application is doing
- Spread your input files across the cluster
  - In /tmp or /scratch
- Use the –M flag to spread the processes home nodes across different nodes in the cluster, to balance the I/O

Temporary private files

- Normally files are accessed via the home-node
- In certain cases it is possible to use Temporary Private Files
- Such private files can be accessed only by the processes
- When the process migrate, temporary private files are migrated with it
- Once the process exit, the files are automatically deleted

Example: temporary private files

- *mosrun –X/tmp –rmos2 –L testload –f/tmp/big –iosize 100 –write 4 –cpu 1*
- In this example the *testload* program writs a file named */tmp/big* in chunks of 4KB up to a size of 100MB
- Since the temporary private files feature is used, the file access will be performed locally

Running Matlab Version 7.4 (or older) jobs

The Matlab program in /usr/local/bin/ is usually a wrapper script like:

- **#! /bin/sh -**
- **LM_LICENSE_FILE=1700@my.university.edu**
- **export LM_LICENSE_FILE**
- **LD_ASSUME_KERNEL=2.4.1**
- **export LD_ASSUME_KERNEL**
- **exec /usr/local/matlab7/5/bin/matlab $***

To run Matlab in the MOSIX environment comment out the following lines:

- **#LD_ASSUME_KERNEL=2.4.1**
- **#export LD_ASSUME_KERNEL**

When running jobs, the following Matlab (and mosrun) flags should be used:

- **mosrun -e matlab -nojvm -nodesktop -nodisplay < matlab-script.m**

See the MOSIX FAQ for further details

Running Matlab Version 7.5 (or newer) jobs

- **Jobs running MATLAB Version 7.5 (or newer) should be started by**
  > mosrun -E -b -i matlab ....

  for assignment to the best node in the cluster.
- **Example: to run the MATLAB test.m program:**

  a=randn(3000);

  b=svd(a);

  use:

  mosrun -E -b -i matlab -nojvm -nodesktop -nodisplay < test.m

Running parallel jobs

- *mosrun -P{n} –q / -Q:* **to run {n} parallel processes by requesting n nodes simultaneously from the queue manager**
- **A patch for MPI uses the above option. It allows adding of an MPI job with n processes to the queue, so that the job is released from the queue only when n processors are available**
  - **Note: this patch uses the –b option of mosrun in order to place processes on the best available nodes**

What is not supported

- **Shared memory, including files mmap'ed as MAP_SHARED and SYSV-shm**
- **The "clone" system-call (causing parent and child processes to share memory and/or files and/or signals and/or current-directory, etc)**
- **Mapping special block/character files to memory**
- **Process tracing (ptrace)**
- **System calls that are esoteric; recently added; intended for system-administration; or to support cloning**
- **Locking memory in core (mlock – has no meaning when a process migrates)**
- **The "-e" flag fails MOSRUN unsupported system-calls (with errno ENOSYS) rather than abort the program when such calls are encountered. "-w" also produces a warning on stderr**

What is not supported - example

mos1:~> mosrun ls –la

MOSRUN: Shared memory (MAP_SHARED) not supported under MOSIX

- **Mmap with the flag "MAP_SHARED" is not supported under mosix**
- **The command "ls –la" will work when using the –e flag of mosrun (mosrun –e ls –la).**
- **This is since the –e tells mosrun to replace the MAP_SHARED with MAP_PRIVATE**
- **Same goes for the error**
  - **MOSRUN: system-call 'futex' not supported under MOSIX**
- **This will not work for all cases, see the MOSIX manual "*man mosix*" for further details**

Solving problems

- **In case of a problem:**
- **Check with mmon, mon that the cluster is working**
- **Run "setpe –r" to see the cluster/multi-cluster configuration**
- **Run "mosctl status" on the problematic node**
- **Try sending a process manually with**

  mosrun –rnode-name program

Configuration

- **All MOSIX configuration files are kept in the *`/etc/mosix`* directory**
- **These files can be modified manually**
- **Or by using the *mosconf* program which allows the sysadmin to configure a MOSIX cluster/multi-cluster by following few basic steps**

mosctl – reading the MOSIX state

- ***mosctl status {node-number | ip-address | host-name}*** **- provides useful information about MOSIX nodes**
- ***mosctl localstatus*** **– provides more information about the local node**
- ***mosctl whois {node-number}*** **– convert a logical MOSIX node number to a host name (or IP address if host name can't be located)**
- ***mosctl whois {IP-address | host-name}*** **- convert an IP address or a host name to a logical MOSIX node number**
- **For explanations of output and more options see the mosctl manual**

mosctl - example

> mosctl status
Status: Running Normally
Load: 0.29 (equivalent to about 0.29 CPU processes)
Speed: 10012 units
CPUS: 1
Frozen: 0
Util: 100%
Avail: YES
Procs: Running 1 MOSIX processes
Accept: Yes, will welcome processes from here
Memory: Available 903MB/1010MB

mosctl localstatus - example

root@mos1:~# mosctl localstatus
Status: Running Normally
Load: 0
Speed: 3333 units
CPUS: 1
Frozen: 0
Util: 100%
Avail: YES
Procs: Running 0 MOSIX processes
Accept: Yes, will welcome processes from here
Memory: Available 93MB/249MB
Swap: Available 0.8GB/0.9GB
Daemons:
Master Daemon: Up
MOSIX Daemon : Up
Queue Manager: Up
Remote Daemon: Up
Guest processes from grid: 0/10

setpe – view the cluster/multi-cluster configuration

- **setpe -r lists the nodes in the local cluster**
- **setpe -R lists the nodes in the local cluster and the multi-cluster**
- **Important information that may be listed:**
  - **pri={pri} - priority we give to that cluster: the lower the better**
  - **proximate - there is a very fast network connection to those nodes**
  - **outsider - processes of class 0 cannot migrate there**
  - **dontgo - local processes cannot migrate there**
  - **dont_take - not accepting guests from there**

setpe - example

```
mos1:~> setpe -R
This MOSIX node is: 132.65.174.1 (mos1.cs.huji.ac.il) (no features)

Nodes in this cluster:
======================
mos1.cs.huji.ac.il - mos38.cs.huji.ac.il

Nodes from the rest of the multi-cluster:
=================================
mos51.cs.huji.ac.il - mos66.cs.huji.ac.il: pri=45,proximate,outsider
cmos-16.cs.huji.ac.il - cmos-20.cs.huji.ac.il: pri=45,proximate,outsider
vmos-02.cs.huji.ac.il - vmos-03.cs.huji.ac.il: pri=45,proximate,outsider
```

The MOSIX programming interface

- **The MOSIX interface allows the users to control some MOSIX options from within their programs**
- **This can be done by accessing special files on the /proc filesystem.**
- **The files are private to the process**
- **The files include:**
  - /proc/self/migrate
  - /proc/self/lock
  - /proc/self/whereami
  - /proc/self/nmigs
  - /proc/self/needmem
  - /proc/self/clear
  - ….

Programming interface - examples

- **To modify the maximal amount of memory that a process may require:**
  - **open("/proc/self/needmem", 1|O_CREAT, 1200)**
- **To lock the program in the current node:**
  - **open("/proc/self/lock", 1|O_CREAT, 1)**
- **To clear statistics after some phase of the program:**
  - **open("/proc/self/clear", 1|O_CREAT, 1)**
- **To find where the current process is running now:**
  - **open("/proc/self/whereami", 0)**