

# **MOSIX Tutorial**

**[http:// www . MOSIX . org](http://www.MOSIX.org)**

# About

**This tutorial has 2 parts:**

- **Part I: for new users. It covers basic tools and operations such as monitors, how to run and control processes, view processes and handling unsupported features**
- **Part II: advanced topics such as freezing processes, checkpoint & recovery, running a large set of processes, I/O optimizations, configuration and management and the programming interface**

# Part I: Basics

- **Tools**

- **mosmon**
- **mosrun, mosnative**
- **mosps**
- **mosmigrate**

- **Operations**

- **Initial assignment**
- **Running Linux processes**

- **Encountering unsupported features**

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

# Monitors - seeing what is going on

*mosmon* 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
  - “d/D” - Dead nodes
  - “h” – help for complete list of options

# *mosrun* – running MOSIX processes

- To run a program under MOSIX 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 (started by *mosrun*) can use the *mosnative* utility to spawn child processes that run in native Linux mode

# Example: view the process migration

- Login to any node in a MOSIX cluster
- On one window run *mosmon*
- On another window start 2 CPU-intensive processes, e.g. the *mostestload* program (included in the MOSIX distribution):
  - >*mosrun testload &*
  - >*mosrun testload &*
- Observe in the *moson* window how the processes move across the 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
- **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* – memory specs

- ***-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
- The ***-m*** options also affects the initial assignment (***-b*** flag)
- **Example:**
  - > *mosrun -m1000 myprog* (allows *myprog* to run only on nodes with at least 1GB of free memory)



## *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
- *FRZ* – if frozen why: “A”- auto frozen (due to load); “M”- manually; “-” – not frozen; “N/A” – can’t be frozen
- *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**

# *mosps* – example

```
> mosps -AMn
```

PID	WHERE	FROM	CLASS	FRZ	NMI	GS	TTY	CMD
24078	cmos-18	here	local	-	1		pts/1	mosrun -b mostestload
24081	here	here	local	-	0		pts/1	mosrun -b mostestload
24089	cmos-16	here	local	-	1		pts/1	mosrun -b mostestload
30115	here	here	local	M	0		pts/1	mosrun mostestload
30253	here	mos3	local	N/A	N/A		?	/sbin/remote

# *mosmigrate* – control MOSIX processes

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

For more options, see the [mosmigrate](#) manual

# Encountering unsupported features

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

**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**
- **What is not supported**
- **Configuration and management**
- **The programming interface**

# Freezing processes

**MOSIX process can be frozen, usually to prevent memory thrashing**

- **While frozen, the process is still alive but is not running - its memory image is left on disk**
- **Frozen processes do not respond to non-fatal signals**
- **Processes can be frozen in 3 ways:**
  - **Manually – upon user's request**
  - **When being expelled from a remote node that was reclaimed**
  - **When the local load is higher than configured**

# Freezing - example

- **Running a process**
  - `mosrun mostestload -m2`
- **Finding the process-ID**
  - `mosps`

PID	WHERE	FROM	FRZ	TTY	CMD
1234	here	here	-	pts/0	mosrun mostestload -m2
- **Using the migrate command to freeze the process**
  - `mosmigrate 1234 freeze`
- **Using migrate to “continue” frozen processes**
  - `mosmigrate 1234 continue`



# 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 minutes
- 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 minutes to files: */tmp/myckpt.1, /tmp/myckpt.2, ...* )
  - > ***mosrun myprog*** (whenever the user requests a checkpoint manually, a checkpoint 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 were 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. To check a specific program, run a sample copy 3-4 times on identical nodes and measure the times, as follows:
  - 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)

# I/O considerations

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

# Improving the I/O performance

- **“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**

# 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 *mostestload* program writes 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**

# 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**

# Solving problems

- In case of a problem:
- Check with **mosmon** that the cluster is working
- Run “**mossetpe -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**

**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
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
  Remote Daemon: Up
  Postal Daemon: uo
Guest processes from grid: 0/10
```



## *mossetpe* – view the cluster/multi-cluster configuration

- *mossetpe -r* lists the nodes in the local cluster
- *mossetpe -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

# 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)`