Clustering with openMosix

Maurizio Davini (Department of Physics and INFN Pisa)

Presented by

Enrico Mazzoni (INFN Pisa)

Introduction

- What is openMosix?
  - Single-System Image
  - Preemptive Process Migration
  - The openMosix File System (MFS)
- Application Fields
- openMosix vs Beowulf
- The people behind openMosix
- The openMosix GNU project
- Fork of openMosix code

The openMosix Project MileStones

- Born early 80s on PDP-11/70. One full PDP and disk-less PDP, therefore process migration idea.
- First implementation on BSD/pdp as MS.c thesis.
- VAX 11/780 implementation (different word size, different memory architecture)
- Motorola / VME bus implementation as Ph.D. thesis in 1993 for under contract from IDF (Israeli Defence Forces)
- 1994 BSDi version
- GNU and Linux since 1997
- Contributed dozens of patches to the standard Linux kernel
- Split Mosix / openMosix November 2001
- Mosix standard in Linux 2.5?

What is openMOSIX

- Linux kernel extension (2.4.20) for clustering
- Single System Image - like an SMP, for:
  - No need to modify applications
  - Adaptive resource management to dynamic load characteristics (CPU intensive, RAM intensive, I/O etc.)
  - Linear scalability (unlike SMP)

A two tier technology

1. Information gathering and dissemination

- **Support scalable configurations by probabilistic dissemination algorithms**
- **Same overhead for 16 nodes or 2056 nodes**

2. Pre-emptive process migration that can migrate any process, anywhere, anytime - transparently

- **Supervised by adaptive algorithms that respond to global resource availability**
- **Transparent to applications, no change to user interface**

Tier 1: Information gathering and dissemination

- **Each unit of time (1 second) each node gathers and disseminates information about:**
  - **CPU(s) speed, load and utilization**
  - **Free memory**
  - **Free proc-table/file-table slots**
- **Info sent to a randomly selected node**
  - **Scalable - more nodes better scattering**

Tier 2: Process migration by adaptive resource management algorithms

- **Load balancing: reduce variance between pairs of nodes to improve the overall performance**
- **Memory ushering: migrate processes from a node that nearly exhausted its free memory, to prevent paging**
- **Parallel File I/O: bring the process to the file-server, direct file I/O from migrated processes**

Performance of process migration

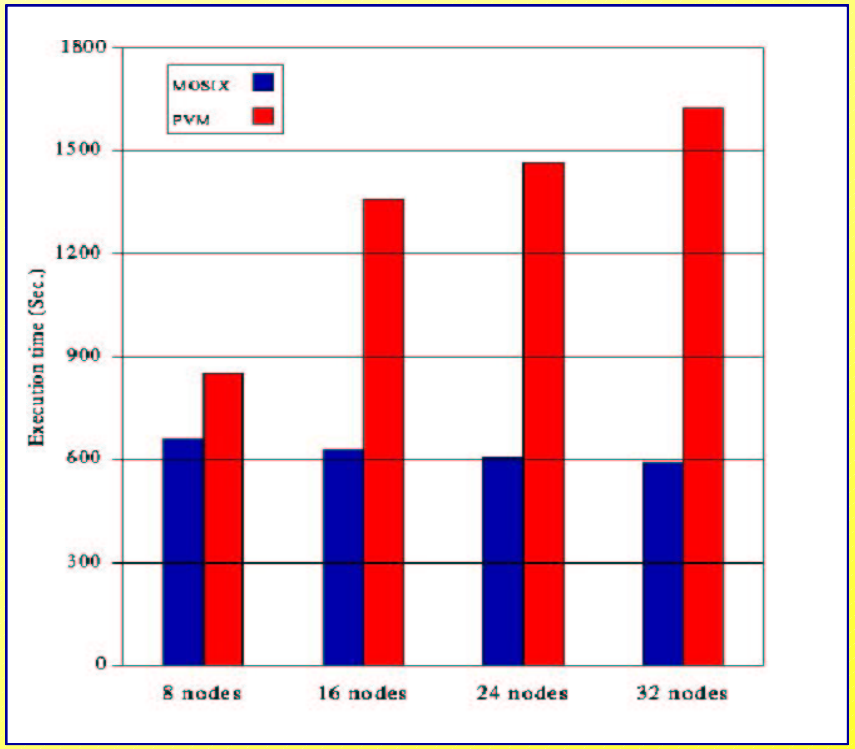
- **CPU: Pentium III 400 MHz**
- **LAN: Fast-Ethernet**
- **For reference: remote system call = 300microsec**
- **Times:**
  - **Initiation time = 1740microsec (less than 6 system calls)**
  - **Migration time = 351microsec per 4KB page**
- **Migration speed: 10.1 MB/Sec = 88.8 Mb/Sec**

Process migration (MOSIX) **vs.** static allocation (PVM/MPI)

Fixed number of processes per node

Random process size with average 8MB

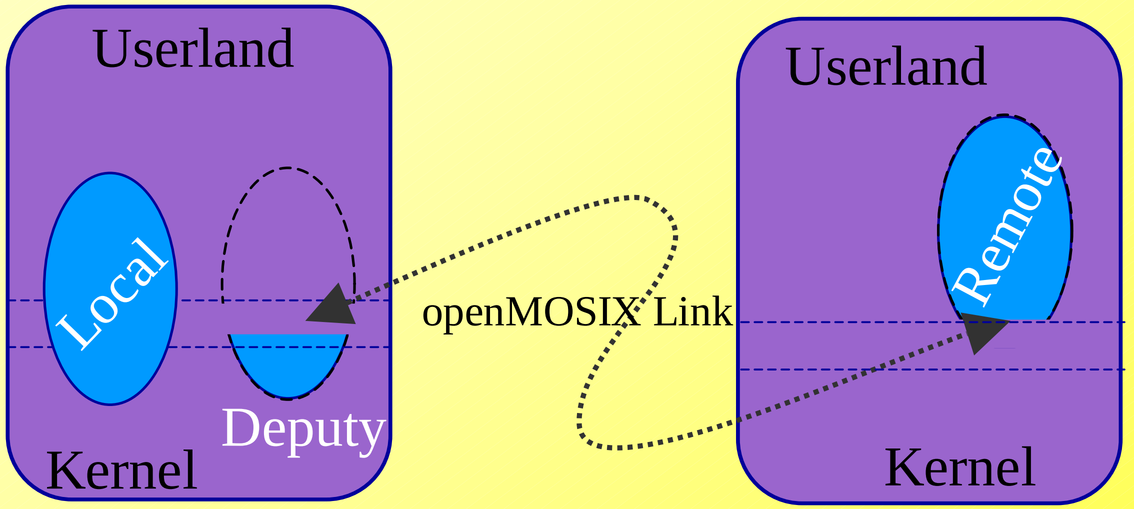
Note the performance **(un)**scalability !



System Image Cluster

- Users can start from any node in the cluster, or sysadmin setups a few nodes as "login" nodes
- Use round-robin DNS: "hpc.qlusters" with many IPs assigned to same name
- Each process has a Home-Node
  - Migrated processes always seem to run at the home node,
    e.g., "ps" show all your processes, even if they run elsewhere

Migration - Splitting the Linux process



- **System context (environment) - site dependent- "home" confined**
- **Connected by an exclusive link for both synchronous (system calls) and asynchronous (signals, MOSIX events)**
- **Process context (code, stack, data) - site independent - may migrate**

Direct File System Access (DFSA)

- **I/O access through the home node incurs high overhead**
- **Direct File System Access (DFSA) compliant file systems allow processes to perform file operations (directly) in the current node - not via the home node**
- **Available operations: all common file-system and I/O system-calls on conforming file systems**
- **Conforming FS: GFS, openMOSIX File System (MFS), Lustre, GPFS and pvfs in the future**

DFSA Requirements

- **The FS (and symbolic-links) are identically mounted on the same-named mount-points**
- **File consistency: when an operation is completed in one node, any subsequent operation on any other node see the results of that operation**
  - **Required because an openMOSIX process may perform consecutive syscalls from different nodes**
  - **Time-stamp consistency: if file A is modified after B, A must have a timestamp S B's timestamp**

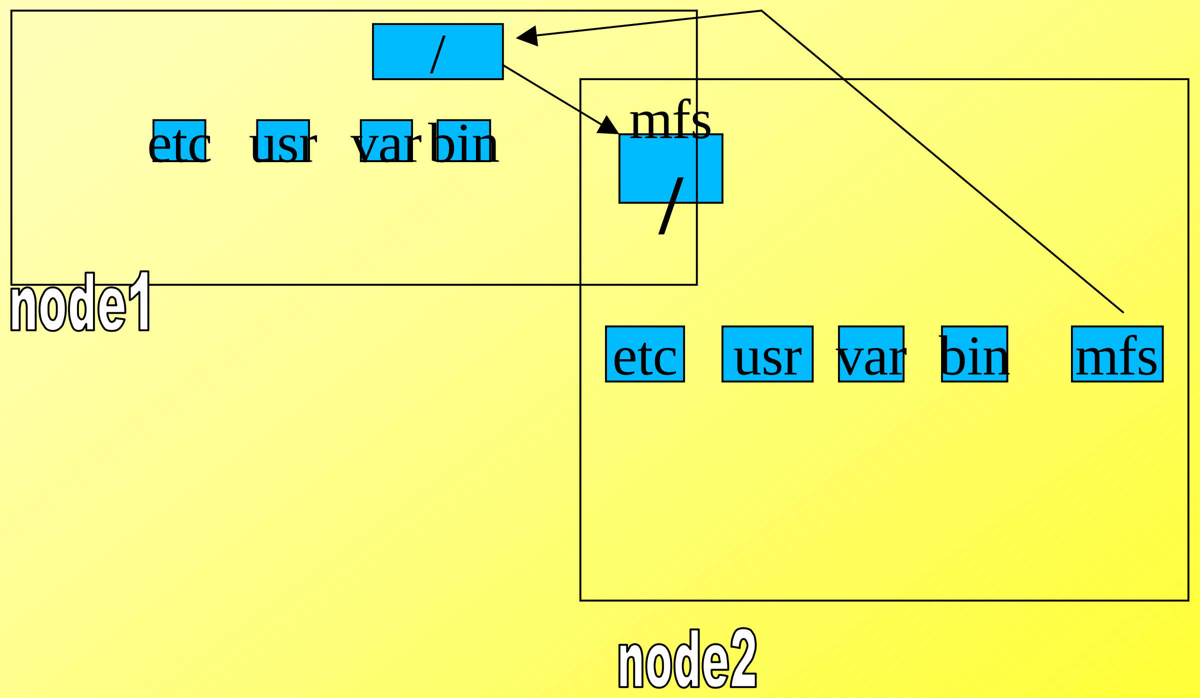
Global File System (GFS) with DFSA

- **Provides local caching and cache consistency over the cluster using a unique locking mechanism**
- **Provides direct access from any node to any storage entity (via Fiber-channel)**
- **Latest: GFS now includes support for DFSA**
- **GFS + process migration combine the advantages of load-balancing with direct disk access from any node - for parallel file operations**
- **Problem with License (SPL)**

The MOSIX File System (MFS)

- Provides a unified view of all files and all mounted FSs on all the nodes of a MOSIX cluster as if they were within a single file system
- Makes all directories and regular files throughout an openMOSIX cluster available from all the nodes
- Provides cache consistency as files viewed from different nodes by maintaining one cache at the server node
- Allows parallel file access by proper distribution of files (each process migrate to the node which has its files)

The MFS File System Namespace



Postmark (heavy FS load) client-server performance

Access Method	***Data Transfer Block Size***						
	64B	**512B**	**1KB**	**2KB**	**4KB**	**8KB**	**16KB**
Local (in the server)	102.6	102.1	100.0	102.2	100.2	100.2	101.0
MFS with DFSA	104.8	104.0	103.9	104.1	104.9	105.5	104.4
NFSv3	184.3	169.1	158.0	161.3	156.0	159.5	157.5
MFS without DFSA	**1711.0**	382.1	277.2	202.9	153.3	136.1	124.5

Kernel 2.4. API and Implementation

- **No new system-calls**
- **Everything done through /proc**

/proc/hpc	
/proc/hpc/admin	Administration
/proc/hpc/info	Cluster-wide information
/proc/hpc/nodes/nnnn/	Per-node information
/proc/hpc/remote/pppp/	Remote proc. information

Performance test (1): PVM on MOSIX

Introduction to PVM

Description

- PVM (Parallel Virtual Machine) is an integral framework that enables a collection of heterogeneous computers to be used in coherent and flexible concurrent computational resource that appear as one single "virtual machine"
- using dedicated library one can automatically start up tasks on the virtual machine. PVM allows the tasks to communicate and synchronize with each other
- by sending and receiving messages, multiple tasks of an application can cooperate to solve a problem in parallel

URL

http://www.epm.ornl.gov/pvm

CPU-bound test description

- this test compares the performance of the execution of sets of identical CPU-bound processes under PVM, with and without MOSIX process migration, in order to highlight the advantages of MOSIX preemptive process migration mechanism and its load balancing scheme
- hardware platform

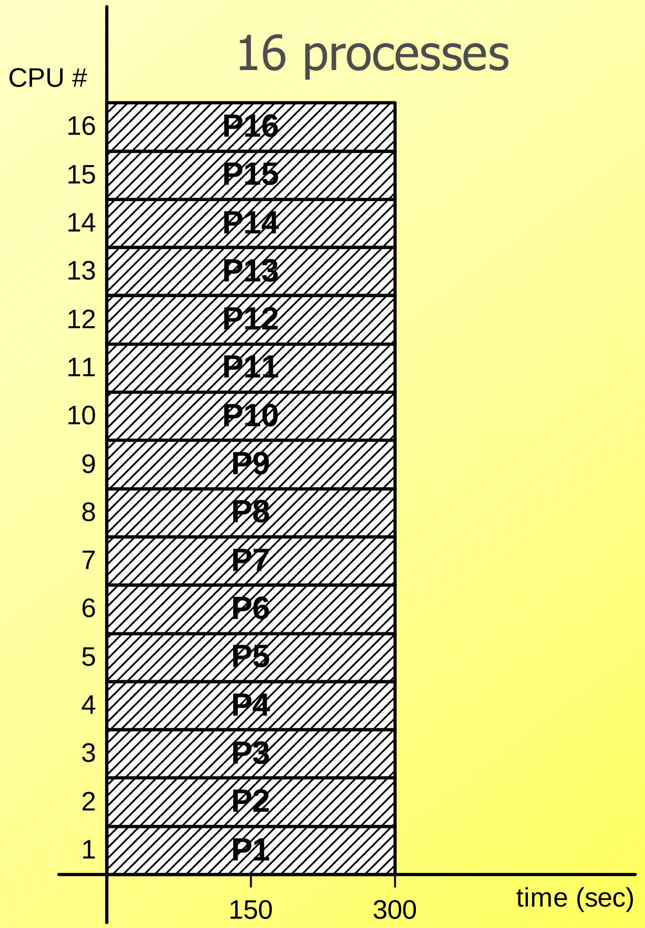
  16 Pentium 90 Mhz that were connected by an Ethernet LAN
- benchmark description

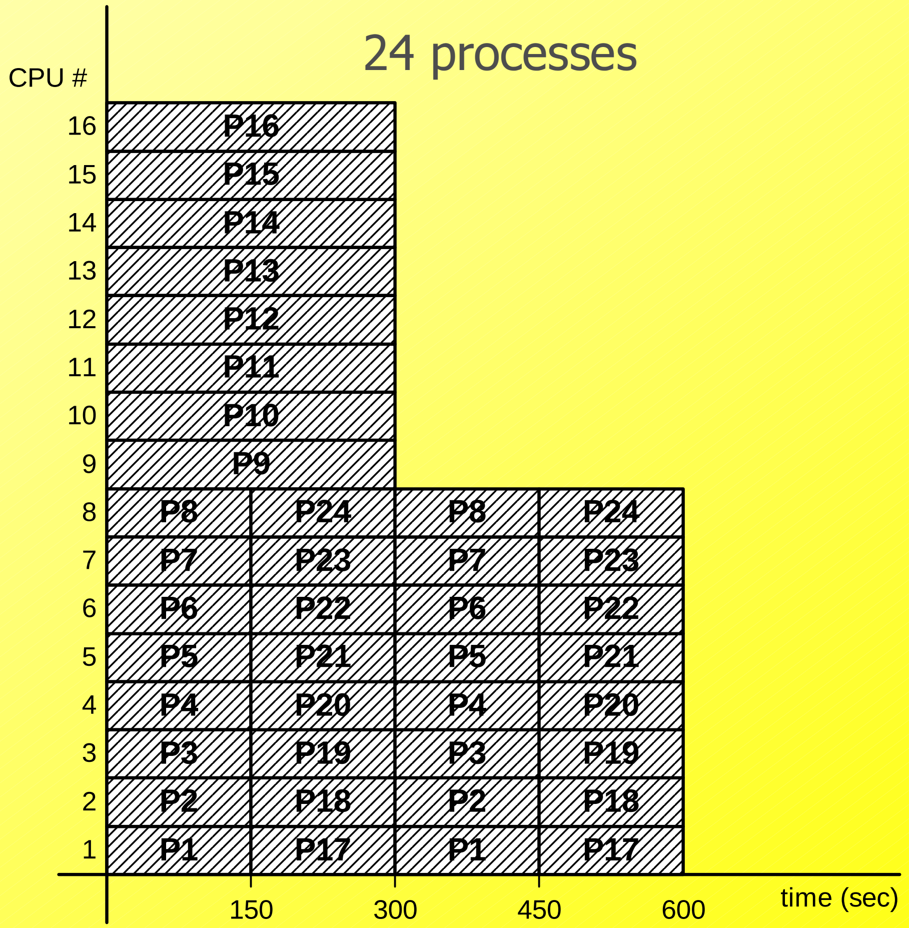
  1) a set of identical CPU-bound processes, each requiring 300 sec.

  2) a set of identical CPU-bound processes that were executed for random durations in the range 0-600 sec.

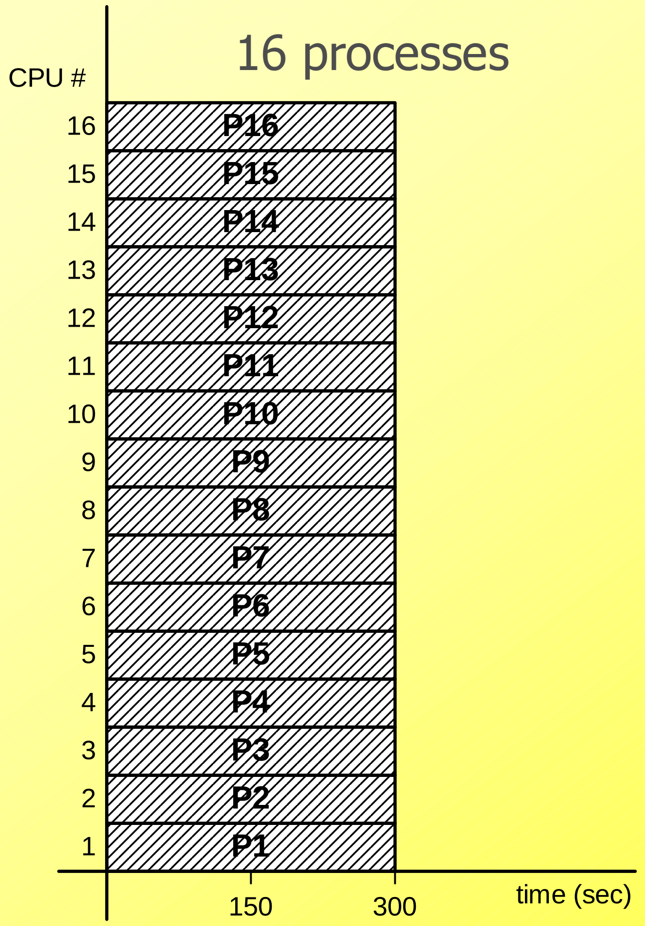
  3) a set of identical CPU-bound processes with a background load

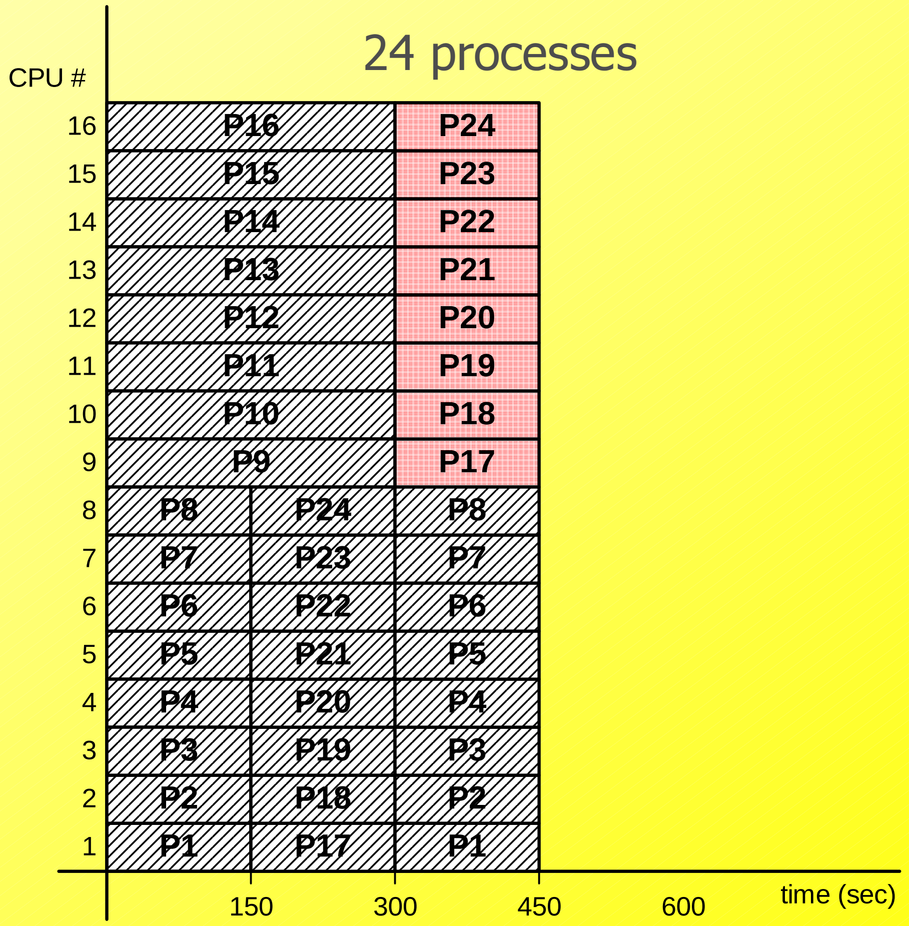
Scheduling without MOSIX



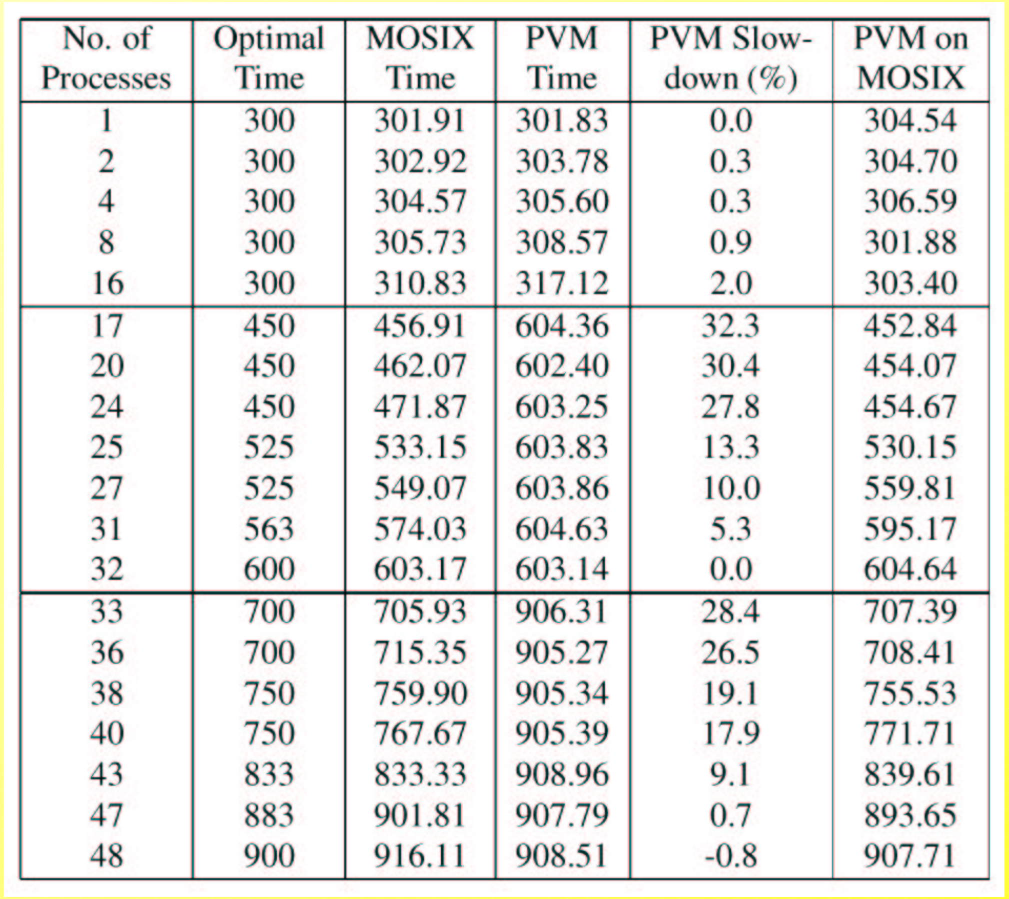


Scheduling with MOSIX





Execution times

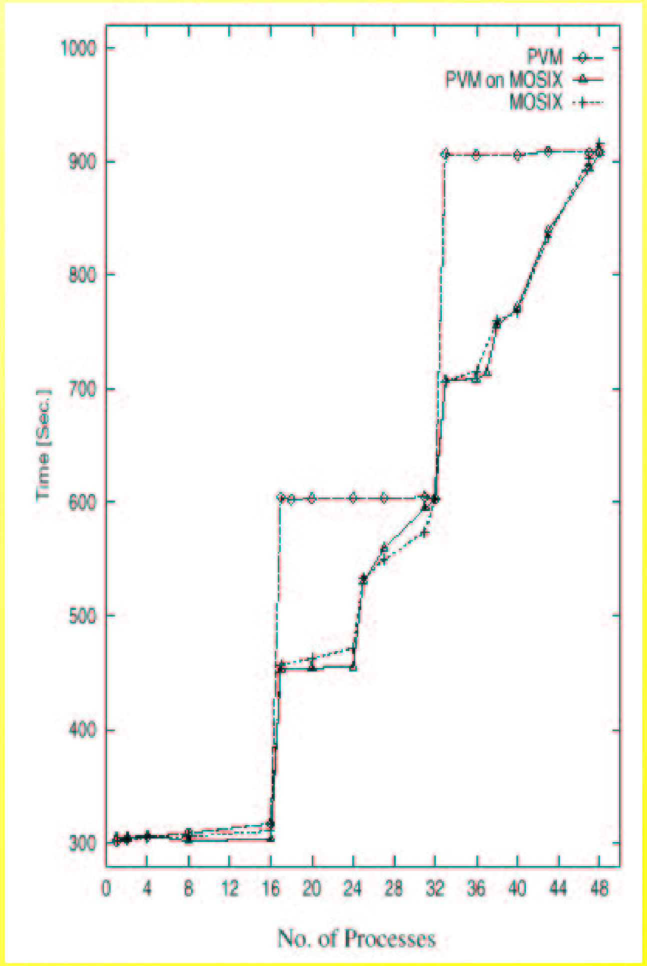


No. of Processes	Optimal Time	MOSIX Time	PVM Time	PVM Slow-down (%)	PVM on MOSIX
1	300	301.91	301.83	0.0	304.54
2	300	302.92	303.78	0.3	304.70
4	300	304.57	305.60	0.3	306.59
8	300	305.73	308.57	0.9	301.88
16	300	310.83	317.12	2.0	303.40
17	450	456.91	604.36	32.3	452.84
20	450	462.07	602.40	30.4	454.07
24	450	471.87	603.25	27.8	454.67
25	525	533.15	603.83	13.3	530.15
27	525	549.07	603.86	10.0	559.81
31	563	574.03	604.63	5.3	595.17
32	600	603.17	603.14	0.0	604.64
33	700	705.93	906.31	28.4	707.39
36	700	715.35	905.27	26.5	708.41
38	750	759.90	905.34	19.1	755.53
40	750	767.67	905.39	17.9	771.71
43	833	833.33	908.96	9.1	839.61
47	883	901.81	907.79	0.7	893.65
48	900	916.11	908.51	-0.8	907.71

Optimal vs. MOSIX vs. PVM vs. PVM on MOSIX execution times (sec)

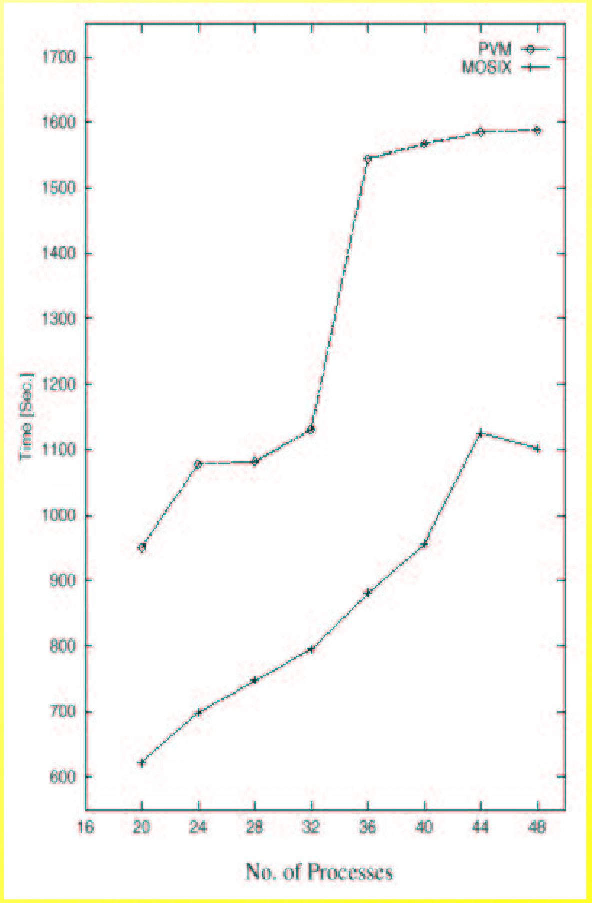
Test #1 results

MOSIX, PVM and PVM on MOSIX execution times



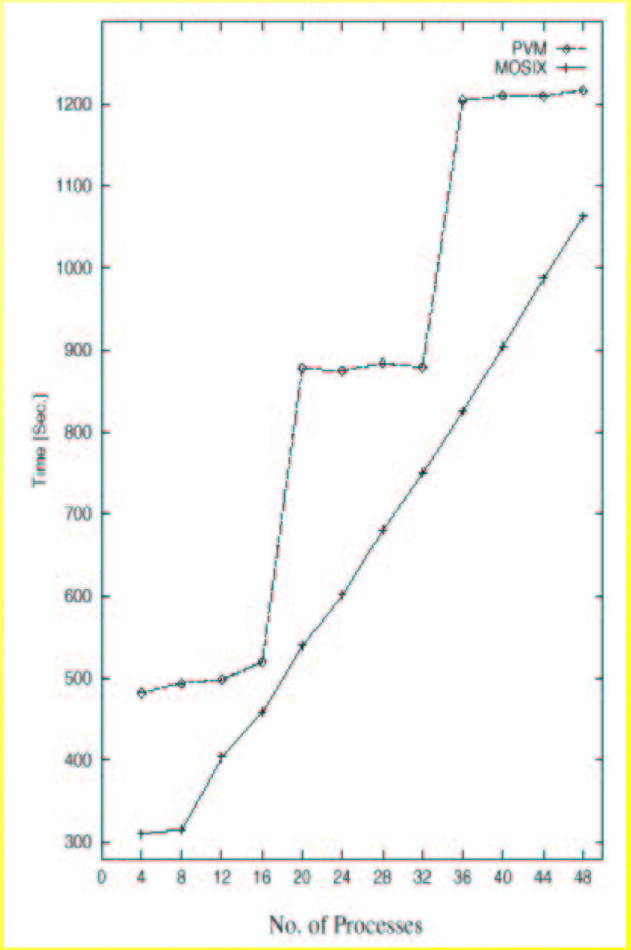
Test #2 results

MOSIX vs. PVM random execution times



Test #3 results

MOSIX vs. PVM with background load execution times



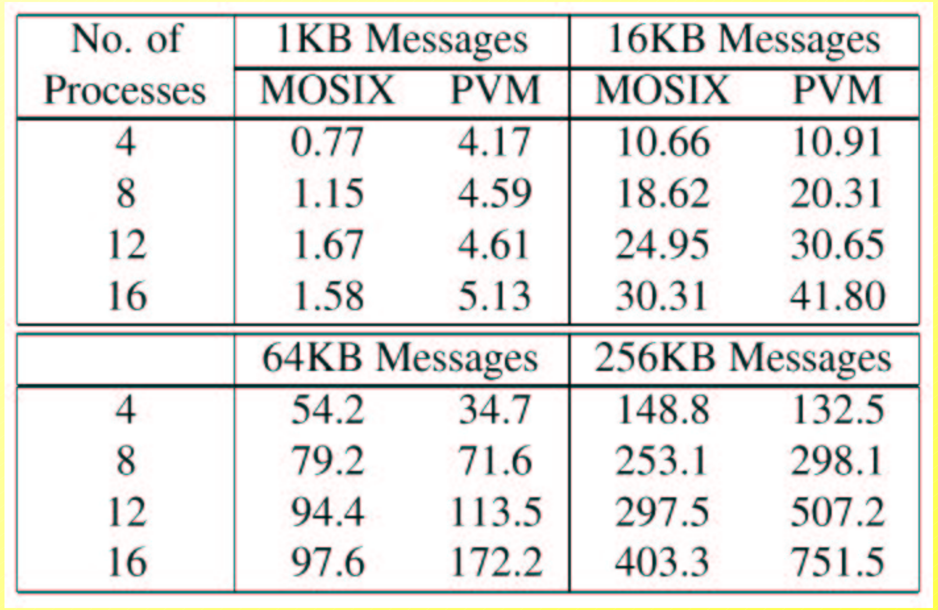
Comm-bound test description

- this test compares the performance of inter-process communication operations between a set of processes
  under PVM and MOSIX

- benchmark description

  each process sends and receives a single message to/from each of its two adjacent processes, then it proceeds with a short CPU-bound computation. In each test, 60 cycles are executed and the net communication times, without the computation times, are measured.

Comm-bound test results



No. of Processes	1KB Messages		16KB Messages	
	MOSIX	PVM	MOSIX	PVM
4	0.77	4.17	10.66	10.91
8	1.15	4.59	18.62	20.31
12	1.67	4.61	24.95	30.65
16	1.58	5.13	30.31	41.80

	64KB Messages		256KB Messages	
4	54.2	34.7	148.8	132.5
8	79.2	71.6	253.1	298.1
12	94.4	113.5	297.5	507.2
16	97.6	172.2	403.3	751.5

MOSIX vs. PVM communication bound processes execution times (sec) for message sizes of 1K to 256K

Performance test (2): molecular dynamics simulation

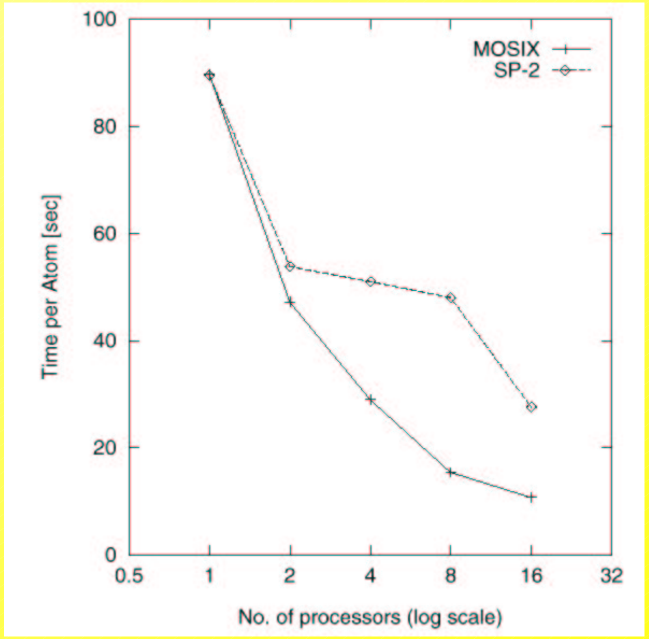
Test description

- molecular dynamics simulation has been used as a tool to study irradiation damage
- the simulation consists of a physical system of an energetic atom (in the range of 100 kev) impacting a surface
- simulation involves a large number of time steps and a large number $(N > 10^6)$ of atoms
- most of calculation is local except the force calculation phase; in this phase each process needs data from all its 26 neighboring processes
- all communication routines are implemented by using the PVM library

Test results

- Hardware used for test
  - 16 nodes Pentium-Pro 200 Mhz with MOSIX
  - Myrinet network

MD performance
of MOSIX vs. the IBM SP2



Performance test (3): MPI on MOSIX

Introduction to MPI

Description

MPI (Message-Passing Interface) is a standard specification for message-passing libraries.
MPICH is a portable implementation of the full MPI specification for a wide variety of parallel computing environments, including workstation clusters

URL

http://www-unix.mcs.anl.gov/mpi/mpich

MPI environment description

- Hardware used for test
  - 2 nodes Dual Pentium III 800 Mhz with MOSIX
  - fast-ethernet network

- Software used for test
  - Linux kernel 2.2.18 + MOSIX 0.97.10
  - MPICH 1.2.1
  - GNU Fortran77 2.95.2
  - NAG library Mark 19

MPI program description (1/2)

The program calculates

$$I = \int dx_1 dx_2 dx_3 dx_4 dx_5 \, f\left(x_1, x_2, x_3, x_4, x_5; \alpha, \beta\right)$$

where α and β are two parameters.

For each value of α, a do loop is performed over four values of β.
MPI routines are used to calculate I for as many values of α as the number of processes. This means that, for example, with a four units cluster with the command

```
mpirun -np 4 intprog
```

each processor performs the calculation of I for the four values of β and a given value of α (the value of α being obviously different for each processor).

MPI program description (2/2)

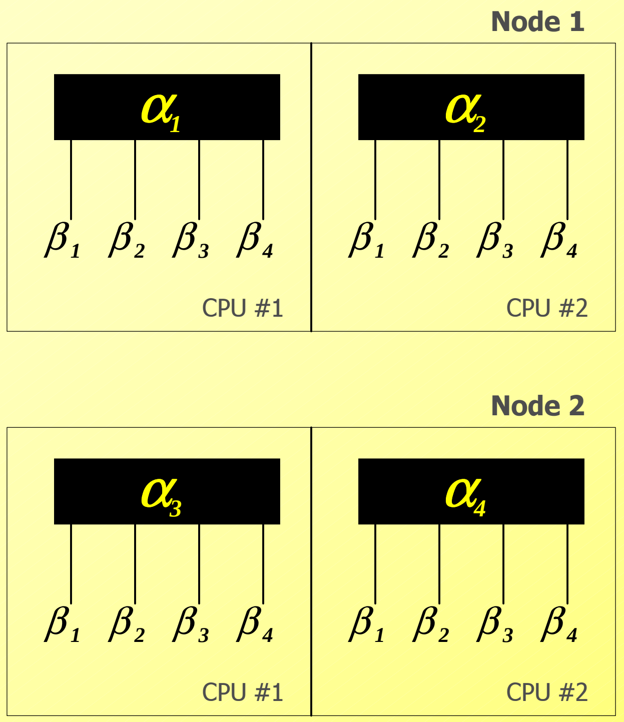
While with the command

```
mpirun –np 8 intprog
```

each processor performs the calculation of I for the four values of β and a couple of values of α.

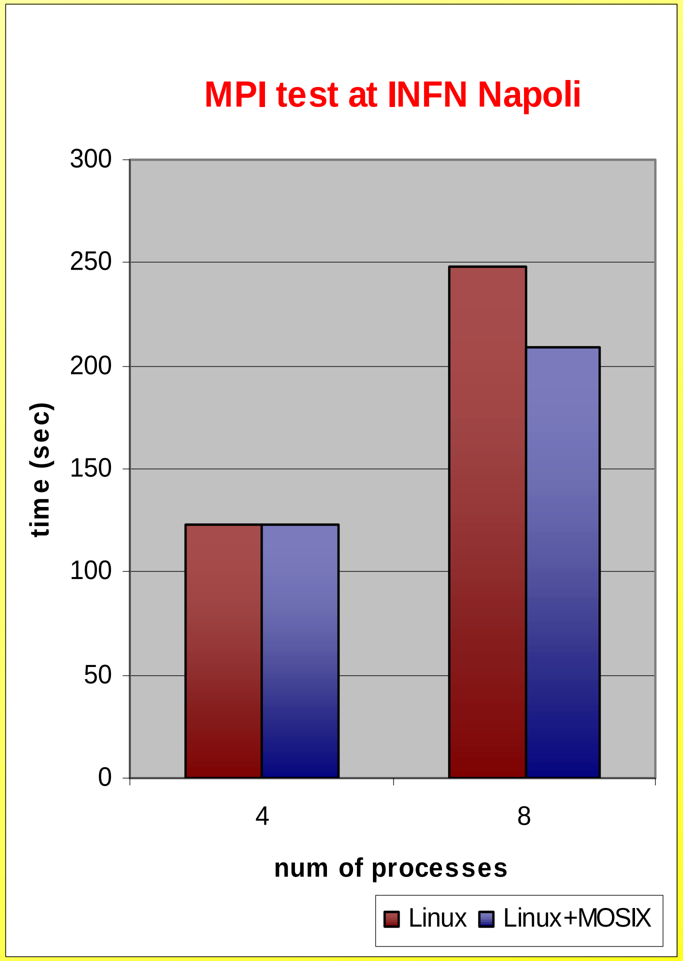
The time employed in this last case is expected to be **two times** the time employed in the first case.

MPI test results


Node 1
α_1
β_1 β_2 β_3 β_4
CPU #1
α_2
β_1 β_2 β_3 β_4
CPU #2
Node 2
α_3
β_1 β_2 β_3 β_4
CPU #1
α_4
β_1 β_2 β_3 β_4
CPU #2

	num. of processes	
Operating System	4	8
Linux	**123**	**248**
Linux+MOSIX	**123**	**209**

(*) each value (in seconds) is the average value of 5 execution times



…continued

- **MOSIX for the 2.2.19 kernel:**
  - 80 new files (40,000 lines)
  - 109 modified files (7,000 lines changed/added)
  - About 3,000 lines are load-balancing algorithms
- **openMOSIX for Linux 2.4.17**
  - **47 new files (38,500 lines)**
  - **126 kernel files modified (5,200 lines changed/added)**
  - **48 user-level files (12,000 lines)**

…continued

- Some ancillary tools
  - Kernel debugger for 2.2. and 2.4
  - Kernel profiler
  - Parallel make (all exec() become mexec()
  - openMosix pvm
  - openMosix mm5
  - openMosix HMMER
  - openMosix Mathematica

Cluster Administration(1)

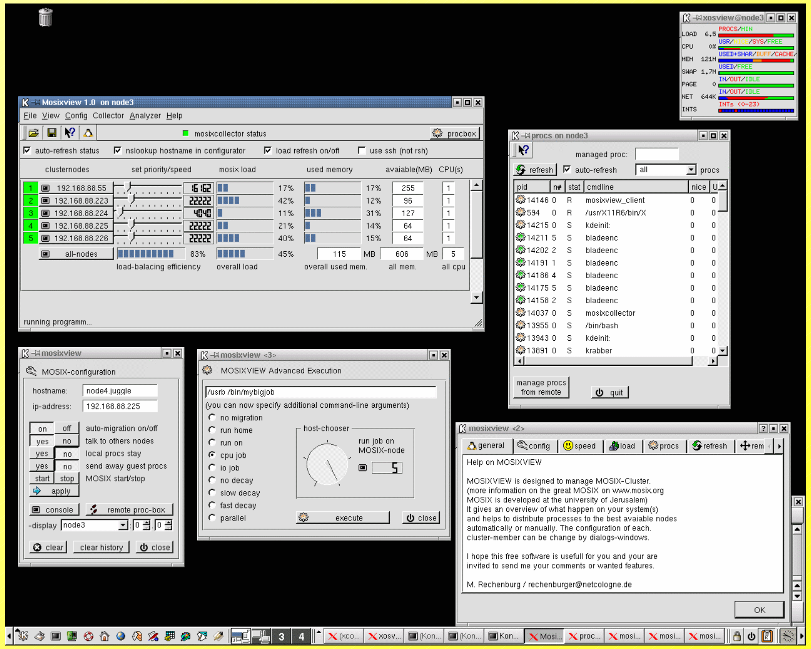
- Various installation options:
  1. LTSP (www.ltsp.org)
  2. ClumpOs (www.clumpos.org)
  3. Debian already includes openMosix
- Use 'mps' & 'mtop' for more complete process status information, ‘mosctl' for node administration

Monitoring

- **Cluster monitor - ‘mosmon’(or ‘qtop’)**
  - Displays load, speed, utilization and memory information across the cluster.
  - Uses the /proc/hpc/info interface for the retrieving information
- **Applet/CGI based monitoring tools - display cluster properties**
  - Access via the Internet
  - Multiple resources
- **openMosixview with X GUI**

openMosixview

- Developed by Mathias Rechemburg
- www.mosixview.com (and its mirror)



☑ auto-refresh status ☑ nslookup hostname in configurator ☑ load refresh on/off ☐ use ssh (not rsh)

	clusternodes	set priority/speed	mosix load		used memory		avaiable(MB)	CPU(s)
1	192.168.88.55	16162		17%		17%	255	1
2	192.168.88.223	22222		42%		12%	96	1
3	192.168.88.224	4040		11%		31%	127	1
4	192.168.88.225	22222		21%		14%	64	1
5	192.168.88.226	22222		40%		15%	64	1
	all-nodes	83%		45%	115 MB		606 MB	5
		load-balacing efficiency	overall load		overall used mem.		all mem.	all cpu

running programm...

managed proc:

☑ auto-refresh all procs

pid	n#	stat	cmdline	nice	U
14146	0	R	mosixview_client	0	0
594	0	R	/usr/X11R6/bin/X	0	0
14215	0	S	kdeinit:	0	0
14211	5	S	bladeenc	0	0
14202	2	S	bladeenc	0	0
14191	1	S	bladeenc	0	0
14186	4	S	bladeenc	0	0
14175	5	S	bladeenc	0	0
14158	2	S	bladeenc	0	0
14037	0	S	mosixcollector	0	0
13955	0	S	/bin/bash	0	0
13943	0	S	kdeinit:	0	0
13891	0	S	krabber	0	0

```
LOAD  6.5   PROCS/MIN
CPU         USR/NICE/SYS/FREE
MEM   121M  USED+SHAR/BUFF/CACHE
SWAP  1.7M  USED/FREE
PAGE     0  IN/OUT/IDLE
NET   644K  IN/OUT/IDLE
INTS        INTs (0-23)
```

MOSIX-configuration

hostname: node4.juggle
ip-address: 192.168.88.225

on	off	auto-migration on/off
yes	no	talk to others nodes
yes	no	local procs stay
yes	no	send away guest procs
start	stop	MOSIX start/stop

-display node3 : 0 . 0

MOSIXVIEW Advanced Execution

/usrb /bin/mybigjob
(you can now specify additional command-line arguments)

- ( ) no migration
- ( ) run home
- ( ) run on
- (•) cpu job
- ( ) io job
- ( ) no decay
- ( ) slow decay
- ( ) fast decay
- ( ) parallel

host-chooser: run job on MOSIX-node 5

Help on MOSIXVIEW

MOSIXVIEW is designed to manage MOSIX-Cluster.
(more information on the great MOSIX on www.mosix.org
MOSIX is developed at the university of Jerusalem)
It gives an overview of what happen on your system(s)
and helps to distribute processes to the best avaiable nodes
automatically or manually. The configuration of each.
cluster-member can be change by dialogs-windows.

I hope this free software is usefull for you and your are
invited to send me your comments or wanted features.

M. Rechenburg / rechenburger@netcologne.de

Application Fields

- **Scalable storage area cluster (SAN + Cluster) for parallel file access**
  - **Scalable transaction processing systems**
- **Scalable web servers: assign new incoming requests to the least loaded node**
  - **Scalable to any number of nodes by IP rotation**
  - **Higher availability**
- **Misc. applications - parallel make**

Example: Parallel Make

- **Assign the next file to the least loaded node**
- **A cluster of 52 4-way 550MHz Xeon nodes**
  - **Runs over a 40 builds of entire code of SAP R/3 (4.7 million lines of code) *concurrently***
  - **Got much better performance vs. LSF cluster for less cost in computing nodes**

People behind openMosix

- Copyright for openMosix, Moshe Bar
- Barak and Moshe Bar were co-project managers of Mosix until Nov 2001
- Team Members
  - Danny Getz (migration)
  - Avraham Ben Yehudah (MFS and 2.5.x)
  - David Santo Orcero (user-space utilities)
  - Michael Farnbach (extern. Patch matching, ie XFS, JFS etc.)
  - Many others, including help from Ingo Molnar, Alan Cox, Andrea Arcangeli and Rik van Riel

HPC Applications

Demanding applications:

- Protein classification
- Molecular dynamics
- Weather forecasting (MM5)
- Computational fluid dynamics
- Car crash numerical simulations (parallel Autodyn)
- Military applications

Current Projects

- Make GFS work with **load-balancing on Fibre channel**
  - A prototype is already working, using SCSI disks
- High availability
  - Recovery in a client/server architecture
  - A distributed lock manager (FS, Cache consistency, shared memory)
  - Monitor lock managers activities to optimize allocation of processes to nodes

Current Projects

- Migrating sockets
- Network RAM
- Distributed Shared Memory
- Checkpoint / Restart
- Queue Manager / Scheduler

Future Plans

- **Inclusion in Linux 2.6**
- **Re-writing MFS**
- **Stub-less migration**
- **Increase developers to 20-30**

Conclusions

- openMosix is today still the most advanced HPC clustering option
- A file system like NFS is not really an option in a cluster, MFS, pvfs, GPFS(perhaps) and GFS (…) are.
- openMosix is much more open than the predecessor
- Over 300 installations already switched to openMosix
  - University of Pisa
  - STM
  - Intel
  - INFN Napoli
  - SISSA

Publications

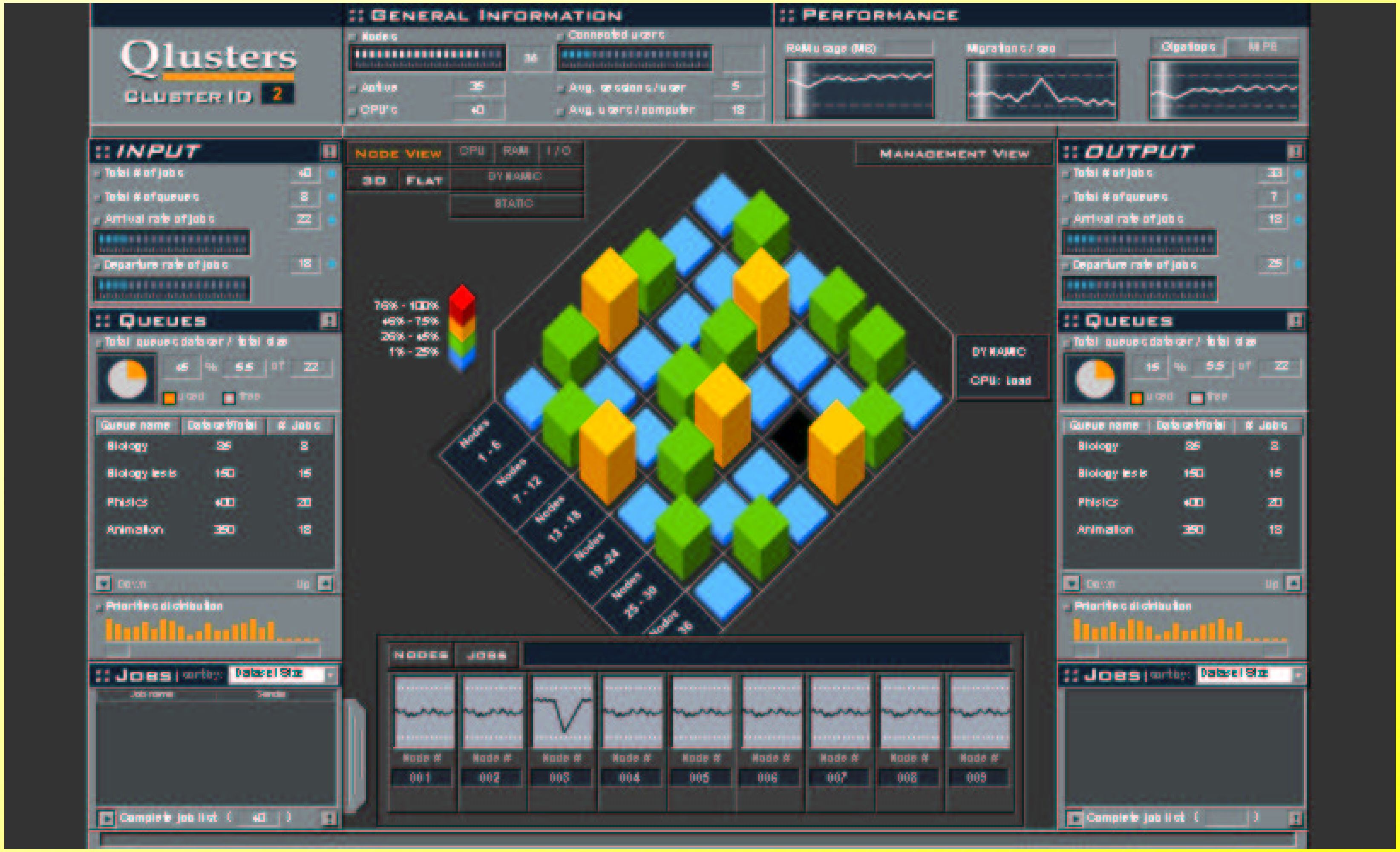
- Amar L., Barak A., Eizenberg A. and Shiloh A.
  The MOSIX Scalable Cluster File Systems for LINUX
  July 2000
- Barak A., La'adan O. and Shiloh A.
  Scalable Cluster Computing with MOSIX for LINUX
  Proc. Linux Expo '99, pp. 95-100, Raleigh, N.C., May 1999
- Barak A. and La'adan O.
  The MOSIX Multicomputer Operating System
  for High Performance Cluster Computing
  Journal of Future Generation Computer Systems, Vol. 13, March 1998

- Postscript versions at: **http://www.mosix.org**

openMosix / Qlusters OS

- Based in part on openMosix technology
- Migrating sockets
- Network RAM already implemented
- Cluster Installer, Configurator, Monitor, Queue Manager, Launcher, Scheduler
- Partnership with IBM, Compaq, Red Hat and Intel
- First sales to Italy

QlusterOS Monitor



openMosix Clusters in Pisa

- **Anubis** cluster
- 13 SuperMicro 6010H Dual PIII 1Ghz,1GB RAM,18 SCSI disk
- RedHat 7.2
- openMosix 1.5.4

openMosix Clusters in Pisa

- **Seth** Cluster
- 27 Appro 1124 Dual AMD Athlon MP 1800+,1GB RAM,18GB SCSI disk
- RedHat 7.2
- openMosix 1.5.4

openMosix Cluster in Pisa

- **Amon** cluster
- 5 dual AMD Athlon 1900+, 1GB RAM,18 GB scsi disk Evolocity Cluster
- RedHat 7.2
- openMosix 1.5.4
- **Donated by AMD Italy**

Linux NetworX

AMD

Farm operating system: Linux kernel + MOSIX

What is MOSIX ?

Description

MOSIX is an **OpenSource** enhancement to the Linux kernel providing adaptive (on-line) load-balancing between x86 Linux machines. It uses preemptive process migration to assign and reassign the processes among the nodes to take the best advantage of the available resources

MOSIX moves processes around the Linux farm to balance the load, using less loaded machines first

URL

http://www.mosix.org

MOSIX introduction

- Execution environment
  - farm of [diskless] x86 based nodes both UP and SMP that are connected by standard LAN
- Implementation level
  - Linux kernel (no library to link with sources)
- System image model
  - virtual machine with a lot of memory and CPU
- Granularity
  - Process
- Goal
  - improve the overall (cluster-wide) performance and create a convenient multi-user, time-sharing environment for the execution of both sequential and parallel applications

MOSIX architecture (1/9)

- network transparency
- preemptive process migration
- dynamic load balancing
- memory sharing
- efficient kernel communication
- probabilistic information dissemination algorithms
- decentralized control and autonomy

MOSIX architecture (2/9)

Network transparency

the interactive user and the application level programs are provided by with a virtual machine that looks like a single machine

Example

disk access from diskless nodes on fileserver is completely transparent to programs

MOSIX architecture (3/9)

Preemptive process migration

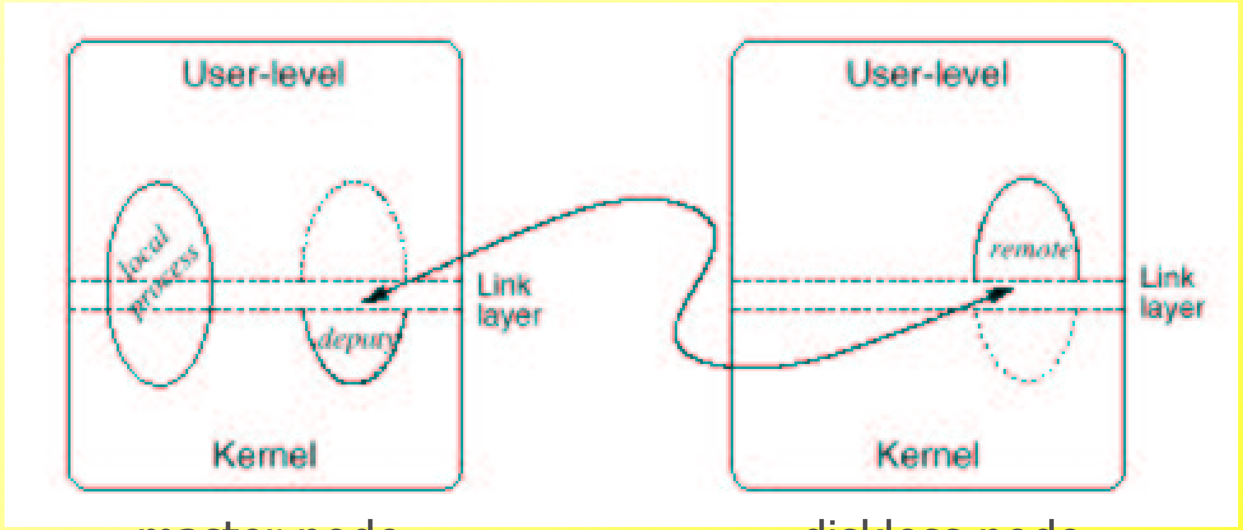
any user’s process, trasparently and at any time, can migrate to any available node.

The migrating process is divided into two contexts:

- system context (deputy) that may not be migrated from “home” workstation (UHN);
- user context (remote) that can be migrated on a diskless node;

MOSIX architecture (4/9)

Preemptive process migration



master node diskless node

MOSIX architecture (5/9)

Dynamic load balancing

- initiates process migrations in order to balance
  the load of farm
- responds to variations in the load of the nodes, runtime characteristics of the processes, number of nodes and their speeds
- makes continuous attempts to reduce the load differences between pairs of nodes and dynamically migrating processes from nodes with higher load to nodes with a lower load
- the policy is symmetrical and decentralized; all of the nodes execute the same algorithm and the reduction of the load differences is performed indipendently by any pair of nodes

MOSIX architecture (6/9)

Memory sharing

- places the maximal number of processes in the farm main memory, even if it implies an uneven load distribution among the nodes
- delays as much as possible swapping out of pages
- makes the decision of which process to migrate and where to migrate it is based on the knoweldge of the amount of free memory in other nodes

MOSIX architecture (7/9)

Efficient kernel communication

- is specifically developed to reduce the overhead of the internal kernel communications (e.g. between the process and its home site, when it is executing in a remote site)
- fast and reliable protocol with low startup latency and high throughput

MOSIX architecture (8/9)

Probabilistic information dissemination algorithms

- provide each node with sufficient knowledge about available resources in other nodes, without polling
- measure the amount of the available resources on each node
- receive the resources indices that each node send at regular intervals to a randomly chosen subset of nodes
- the use of randomly chosen subset of nodes is due for support of dynamic configuration and to overcome partial nodes failures

MOSIX architecture (9/9)

Decentralized control and autonomy

- each node makes its own control decisions independently and there is no master-slave relationship between nodes
- each node is capable of operating as an independent system; this property allows a dynamic configuration, where nodes may join or leave the farm with minimal disruption

Future directions:
DFSA and GFS

Introduction

- MOSIX is particularly efficient for distributing and executing CPU-bound processes
- however the MOSIX scheme for process distribution is inefficient for executing processes with significant amount of I/O and/or file operations
- to overcome this inefficiency MOSIX is enhanced with a provision for Direct File System Access (DFSA) for better handling of I/O-bound processes

How DFSA works

- DFSA was designed to reduce the extra overhead of executing I/O oriented system-calls of a migrated process
- The Direct File System Access (DFSA) provision extends the capability of a migrated process to perform some I/O operations locally, in the current node.
- This provision reduces the need of I/O-bound processes to communicate with their home node, thus allowing such processes (as well as mixed I/O and CPU processes) to migrate more freely among the cluster's node (for load balancing and parallel file and I/O operations)

DFSA-enabled filesystems

- DFSA can work with any file system that satisfies some properties (cache consistency, syncronization, unique mount point, etc.)
- currently, only GFS (Global File System) and MFS (Mosix File System) meets the DFSA standards

NEWS: The MOSIX group has made considerable progress integrating GFS with DFSA-MOSIX

Conclusions

Environments that benefit from MOSIX (1/2)

- **CPU-bound processes**
  with long (more than few seconds) execution times and low volume of IPC relative to the computation, e.g., scientific, engineering and other HPC demanding applications.
  For processes with mixed (long and short) execution times or with moderate amounts of IPC, we recommend PVM/MPI for initial process assignments
- **multi-user, time-sharing environment**
  where many users share the cluster resources. MOSIX can benefit users by transparently reassigning their more CPU demanding processes, e.g., large compilations, when the system gets loaded by other users

Environments that benefit from MOSIX (2/2)

- **parallel processes**
  especially processes with unpredictable arrival and execution times - the dynamic load-balancing scheme of MOSIX can outperform any static assignment scheme throughout the execution
- **I/O-bound and mixed I/O and CPU processes**
  by migrating the process to the "file server", then using DFSA with GFS or MFS
- **farms with different speed nodes and/or memory sizes**
  the adaptive resource allocation scheme of MOSIX always attempts to maximize the performance

Environments currently not benefit much from MOSIX

- **I/O bound applications with little computation**
  this will be resolved when we finish the development of a "migratable socket"
- **shared-memory applications**
  since there is no support for DSM in Linux. However, MOSIX will support DSM when we finish the "Network RAM" project, in which we migrate processes to data rather than data to processes
- **hardware dependent applications**
  that require direct access to the hardware of a particular node

Conclusions

- the most noticeable features of MOSIX are its load-balancing and process migration algorithms, which implies that users need not have knowledge of the current state of the nodes
- this is most useful in time-sharing, multi-user environments, where users do not have means (and usually are not interested) in the status (e.g. load of the nodes)
- parallel application can be executed by forking many processes, just like in an SMP, where MOSIX continuously attempts to optimize the resource allocation

References