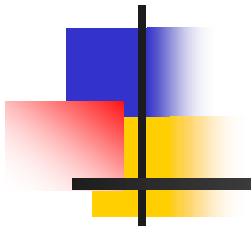
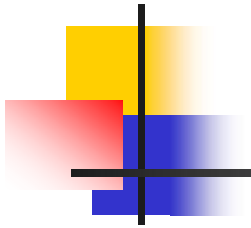


MOSIX: High performance Linux farm

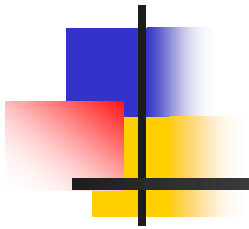
Paolo Mastroserio [mastroserio@na.infn.it]
Francesco Maria Taurino [taurino@na.infn.it]
Gennaro Tortone [tortone@na.infn.it]

Napoli

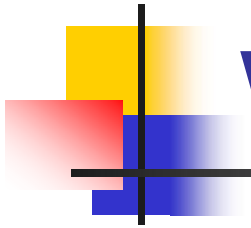


Index

- overview on Linux farm
- farm setup: Etherboot and Cluster-NFS
- farm OS: Linux kernel + MOSIX
- performance test (1): PVM on MOSIX
- performance test (2): molecular dynamics simulation
- performace test (3): MPI on MOSIX
- future directions: DFSA and GFS
- conclusions
- references



Overview on Linux farm



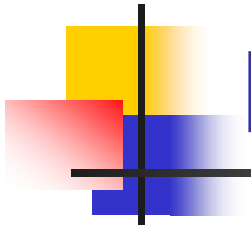
Why Linux farm ?

- high performance
- low cost

Problems with big supercomputers

- high cost
- low and expensive scalability

  (CPU, disk, memory, OS, programming tools, applications)



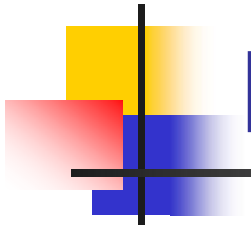
Linux farm: common hardware

Node devices

- CPU + SMP motherboard (Pentium IV)
- RAM (512 Mb ÷ 4 Gb)
- more fixed disks ATA 66/100 or SCSI

Network

- Fast Ethernet (100 Mbps)
- Gigabit Ethernet (1Gbps)
- Myrinet (1.2Gbps), ...



Programming environments

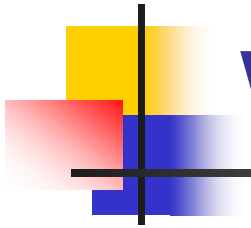
- MPI - Message Passing Interface

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

- PVM - Parallel Virtual Machine

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

- Threads



What makes clusters hard ?

Setup (administrator)

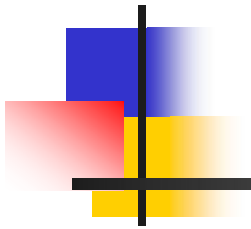
- setting up a 16 node farm by hand is prone to errors

Maintenance (administrator)

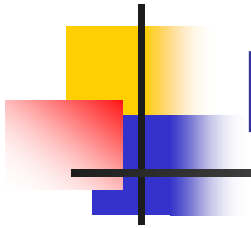
- ever tried to update a package on every node in the farm

Running jobs (users)

- running a parallel program or set of sequential programs requires the users to figure out which hosts are available and manually assign tasks to the nodes

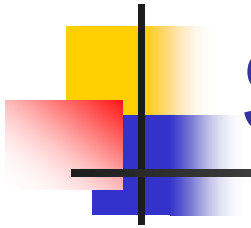


Farm setup: Etherboot and ClusterNFS



Diskless node

- low cost
- eliminates install/upgrade of hardware, software on diskless client side
- backups are centralized in one single main server
- zero administration at diskless client side



Solution: Etherboot (1/2)

Description

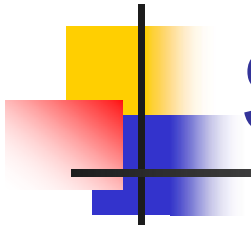
Etherboot is a package for creating ROM images that can download code from the network to be executed on an x86 computer

Example

maintaining centrally software for a cluster of equally configured workstations

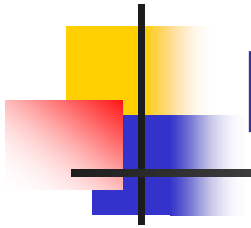
URL

http://www.etherboot.org



Solution: Etherboot (2/2)

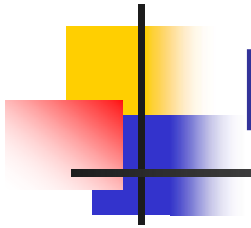
- The components needed by Etherboot are
  - A bootstrap loader, on a floppy or in an EPROM on a NIC card
  - A Bootp or DHCP server, for handing out IP addresses and other information when sent a MAC (Ethernet card) address
  - A tftp server, for sending the kernel images and other files required in the boot process
  - A NFS server, for providing the disk partitions that will be mounted when Linux is being booted.
  - A Linux kernel that has been configured to mount the root partition via NFS



Diskless farm setup traditional method (1/2)

Traditional method

- Server
  - BOOTP server
  - NFS server
  - separate root directory for each client
- Client
  - BOOTP to obtain IP
  - TFTP or boot floppy to load kernel
  - rootNFS to load root filesystem

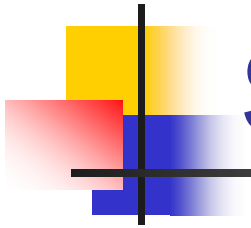


Diskless farm setup traditional method (2/2)

Traditional method – Problems

separate root directory structure for each node

- hard to set up
  - lots of directories with slightly different contents
- difficult to maintain
  - changes must be propagated to each directory



Solution: ClusterNFS

Description

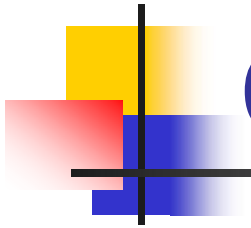
cNFS is a patch to the standard Universal-NFS server code that “parses” file request to determine an appropriate match on the server

Example

when client machine foo2 asks for file `/etc/hostname` it gets the contents of `/etc/hostname$$HOST=foo2$$`

URL

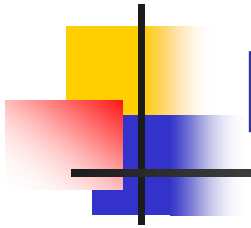
https://sourceforge.net/projects/clusternfs



ClusterNFS features

ClusterNFS allows all machines (including server) to share the root filesystem

- all files are shared by default
- files for all clients are named `filename$$CLIENT$$`
- files for specific client are named `filename$$IP=xxx.xxx.xxx.xxx$$` or `filename$$HOST=host.domain.com$$`

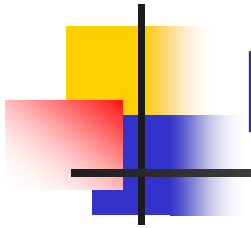


Diskless farm setup with ClusterNFS (1/2)

ClusterNFS method

- Server
  - BOOTP server
  - **ClusterNFS** server
  - **single root directory for server and clients**

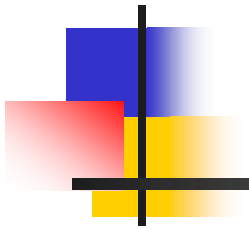
- Clients
  - BOOTP to obtain IP
  - TFTP or boot floppy to load kernel
  - rootNFS to load root filesystem



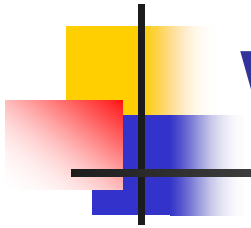
Diskless farm setup with ClusterNFS (2/2)

ClusterNFS method – Advantages

- easy to set up
  - just copy (or create) the files that need to be different
- easy to maintain
  - changes to shared files are global
  - easy to add nodes



Farm operating system: Linux kernel + MOSIX



What is MOSIX ?

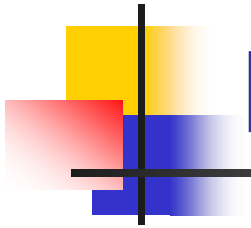
Description

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

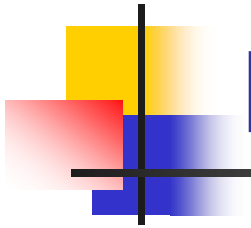
- virtual machine with a lot of memory and CPU

Granularity

- Process

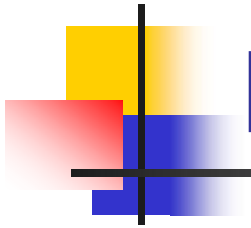
Goal

- 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



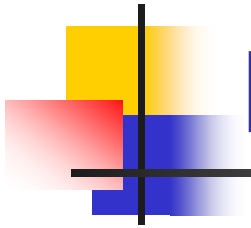
MOSIX architecture (2/9)

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

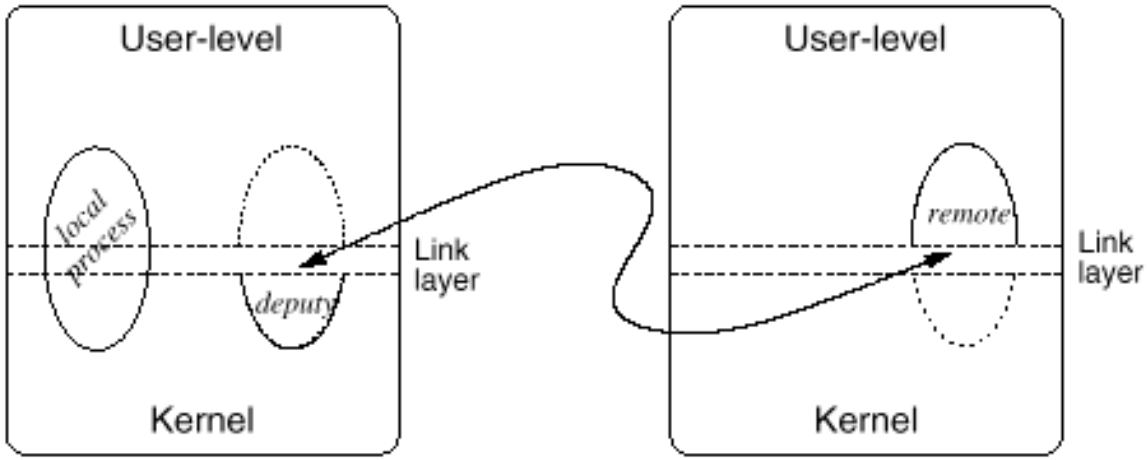
any user's process, trasparently and at any time, can migrate to any available node.

The migrating process is divided into two contexts:

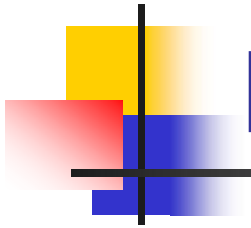
- system context (deputy) that may not be migrated from "home" workstation (UHN);
- user context (remote) that can be migrated on a diskless node;

MOSIX architecture (4/9)

Preemptive process migration



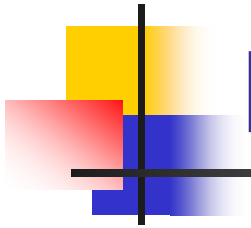
master node diskless node



MOSIX architecture (5/9)

Dynamic load balancing

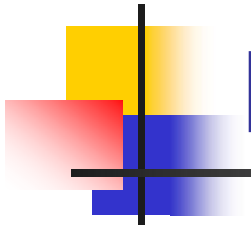
- initiates process migrations in order to balance the load of farm
- responds to variations in the load of the nodes, runtime characteristics of the processes, number of nodes and their speeds
- makes continuous attempts to reduce the load differences between pairs of nodes and dynamically migrating 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

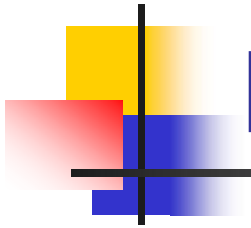
- places the maximal number of processes in the farm main memory, even if it implies an uneven load distribution among the nodes
- delays as much as possible swapping out of pages
- makes the decision of which process to migrate and where to migrate it is based on the knoweldge of the amount of free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



MOSIX architecture (7/9)

Efficient kernel communication

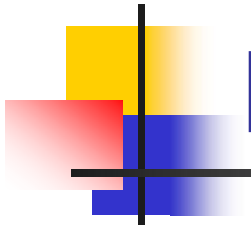
- is specifically developed to reduce the overhead of the internal kernel communications (e.g. between the process and its home site, when it is executing in a remote site)
- fast and reliable protocol with low startup latency and high throughput



MOSIX architecture (8/9)

Probabilistic information dissemination algorithms

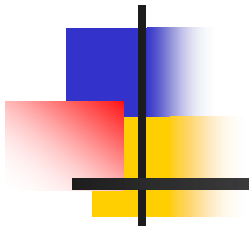
- provide each node with sufficient knowledge about available resources in other nodes, without polling
- measure the amount of the available resources on each node
- receive the resources indices that each node send at regular intervals to a randomly chosen subset of nodes
- the use of randomly chosen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



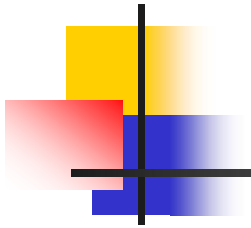
MOSIX architecture (9/9)

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



Performance test (1): PVM on MOSIX



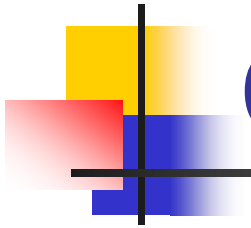
Introduction to PVM

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

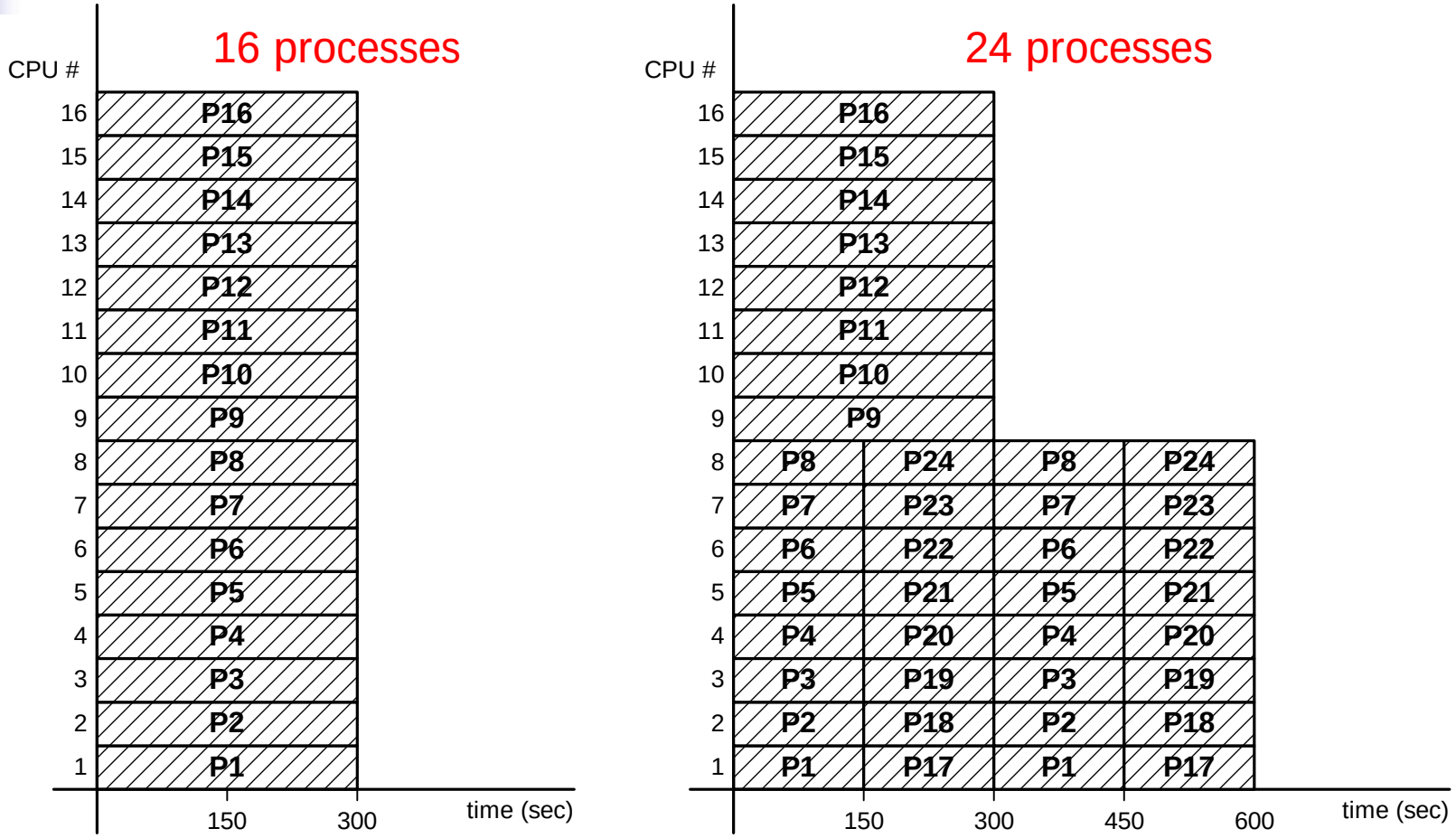
  16 Pentium 90 Mhz that were connected by an Ethernet LAN
- benchmark description

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

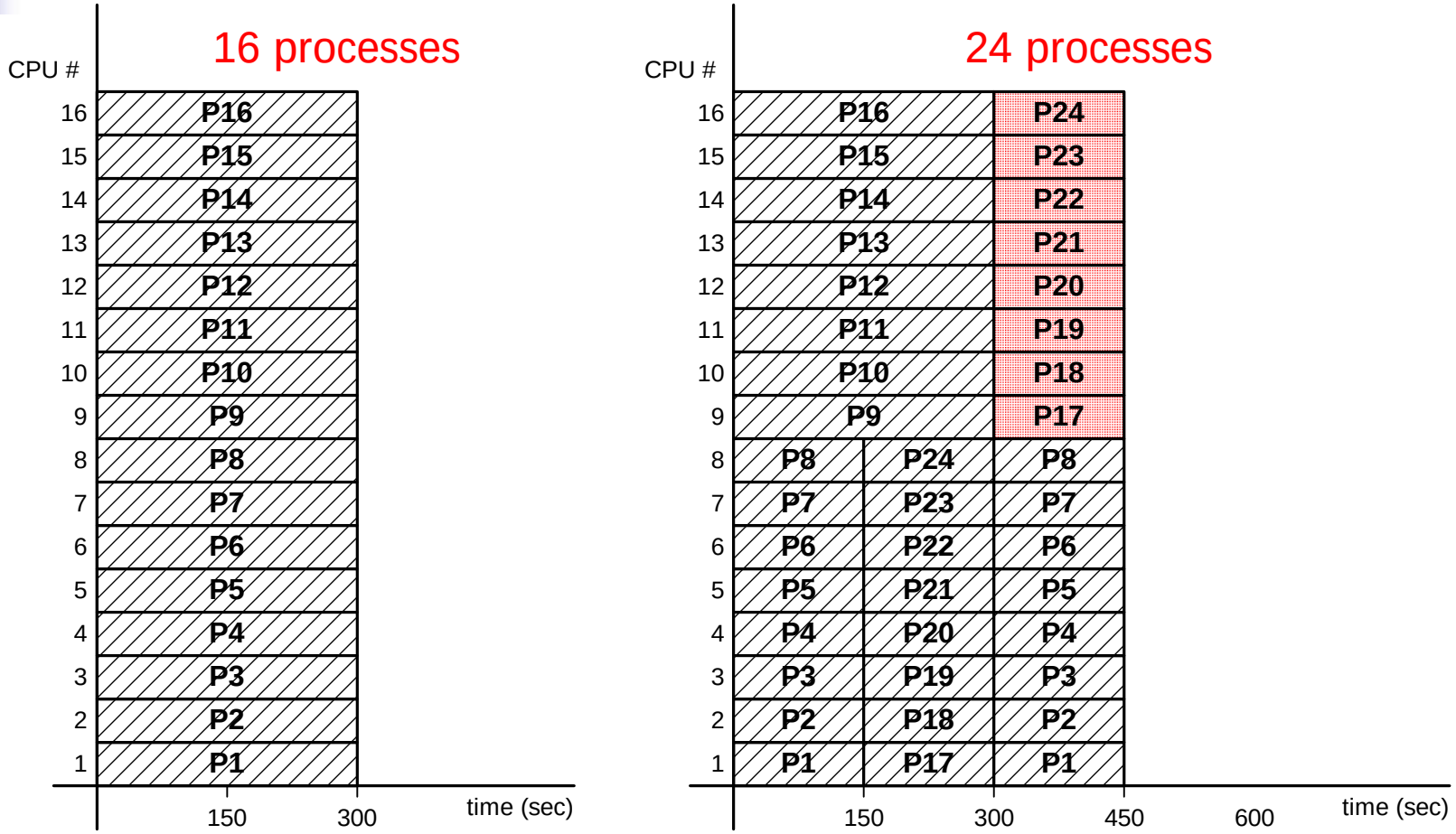
  2) a set of identical CPU-bound processes that were executed for random durations in the range 0-600 sec.

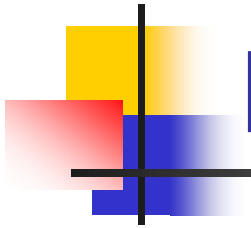
  3) a set of identical CPU-bound processes with a background load

Scheduling without MOSIX



Scheduling with MOSIX





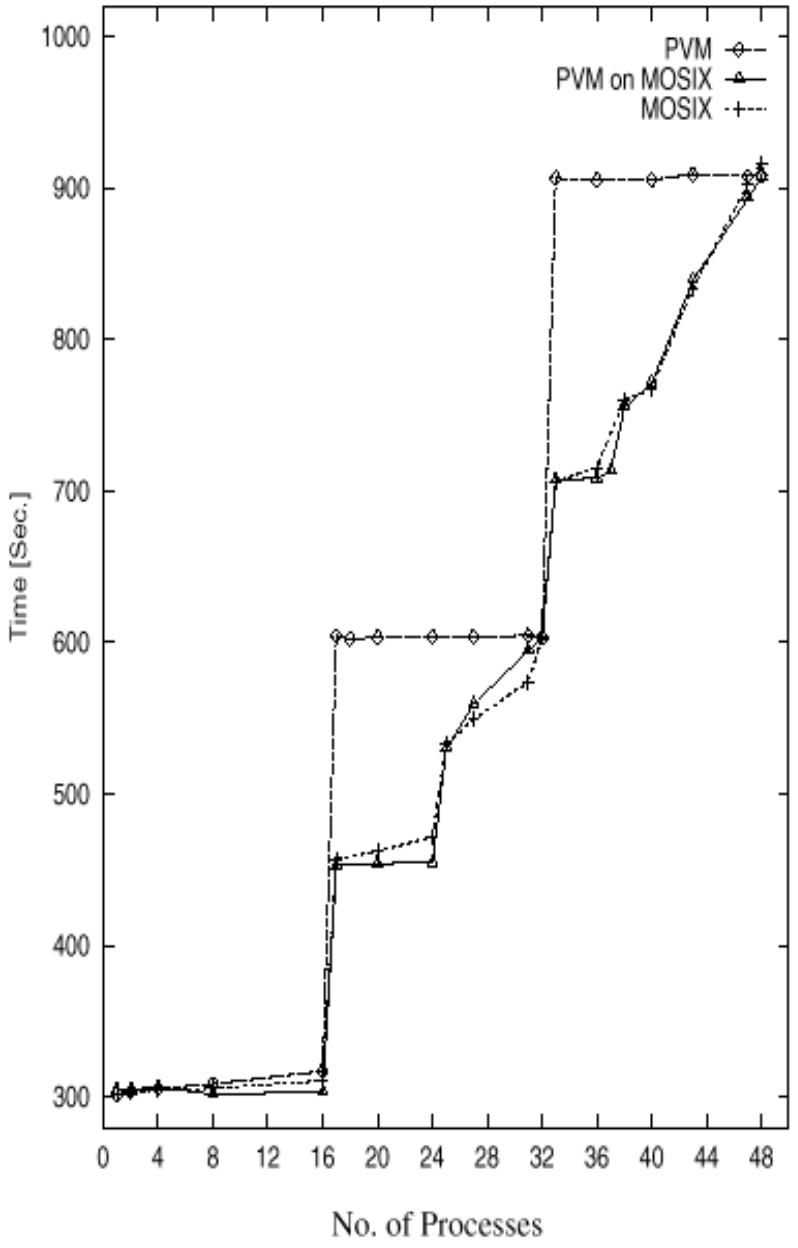
Execution times

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)

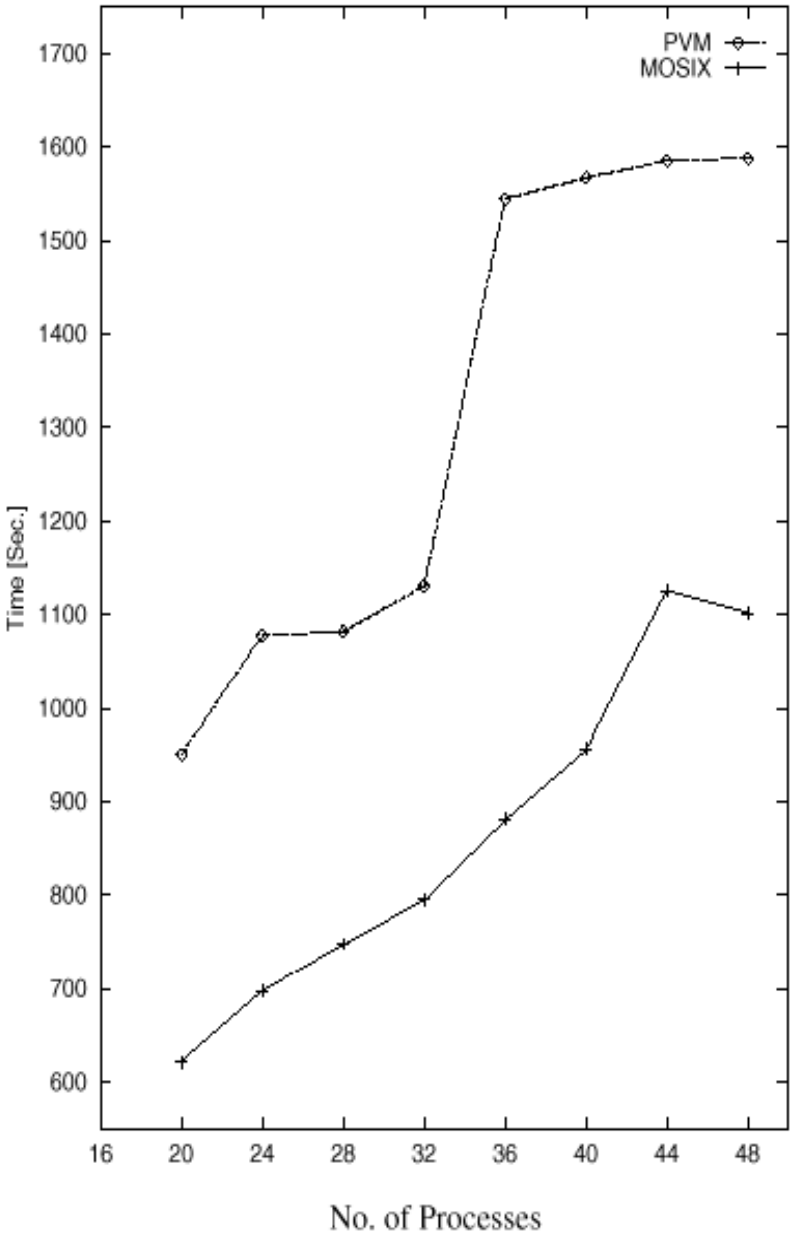
Test #1 results

MOSIX, PVM and PVM on MOSIX execution times



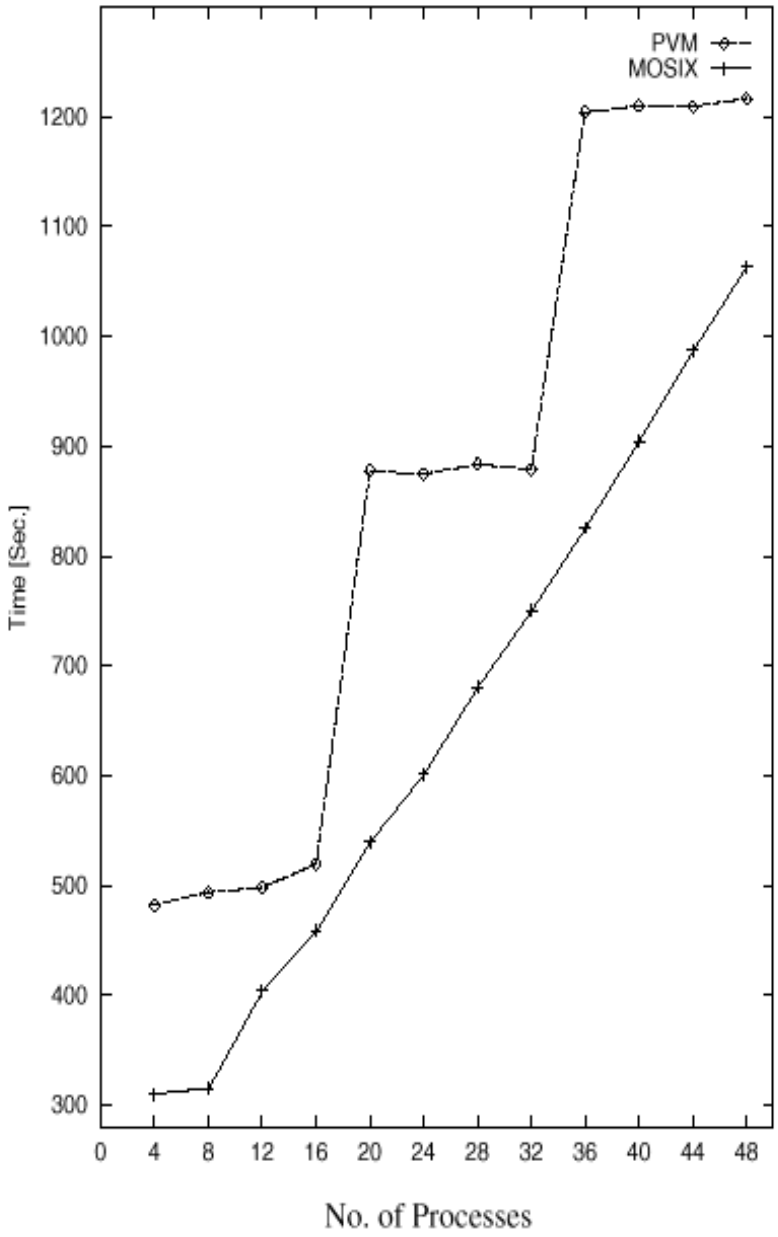
Test #2 results

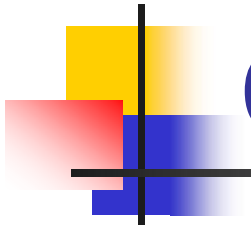
MOSIX vs. PVM random execution times



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

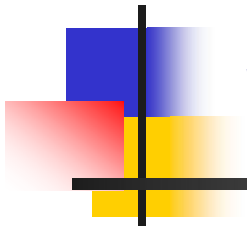
- benchmark description

  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.

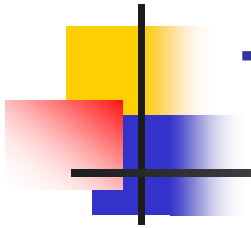
Comm-bound test results

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

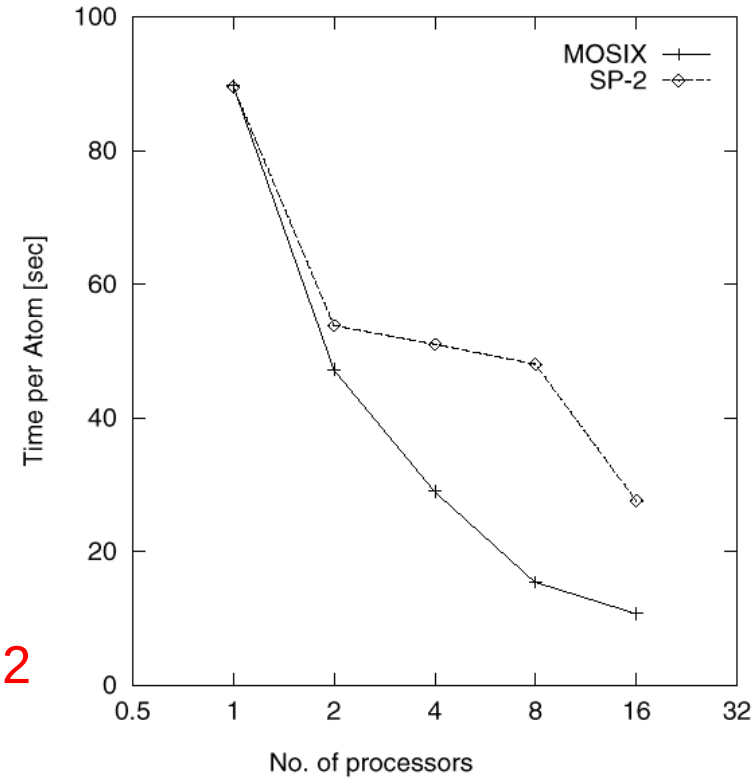


Test description

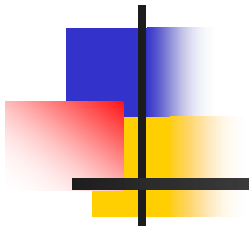
- molecular dynamics simulation has been used as a tool to study irradiation damage
- the simulation consists of a physical system of an energetic atom (in the range of 100 kev) impacting a surface
- simulation involves a large number of time steps and a large number ($N > 10^6$) of atoms
- most of calculation 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

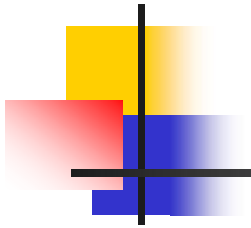
- Hardware used for test
  - 16 nodes Pentium-Pro 200 Mhz with MOSIX
  - Myrinet network



MD performance
of MOSIX vs. the IBM SP2



Performance test (3): MPI on MOSIX



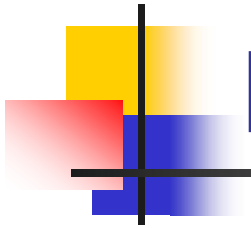
Introduction to MPI

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

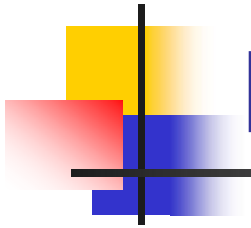
URL

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



MPI environment description

- Hardware used for test
  - 2 nodes Dual Pentium 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

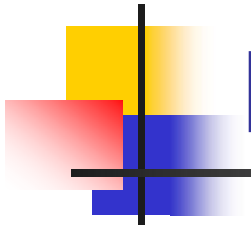
$$I = \int dx_1 dx_2 dx_3 dx_4 dx_5 \, f\left(x_1, x_2, x_3, x_4, x_5; \alpha, \beta\right)$$

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)

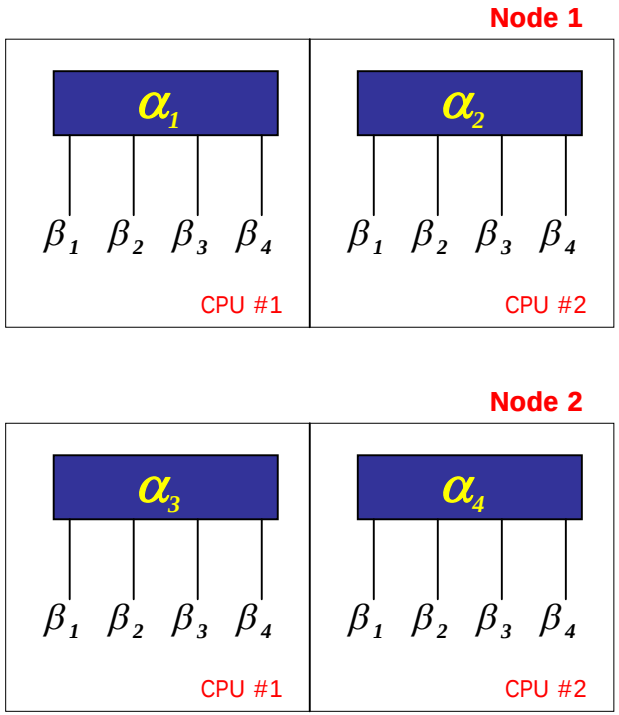
While with the command

```
mpirun –np 8 intprog
```

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

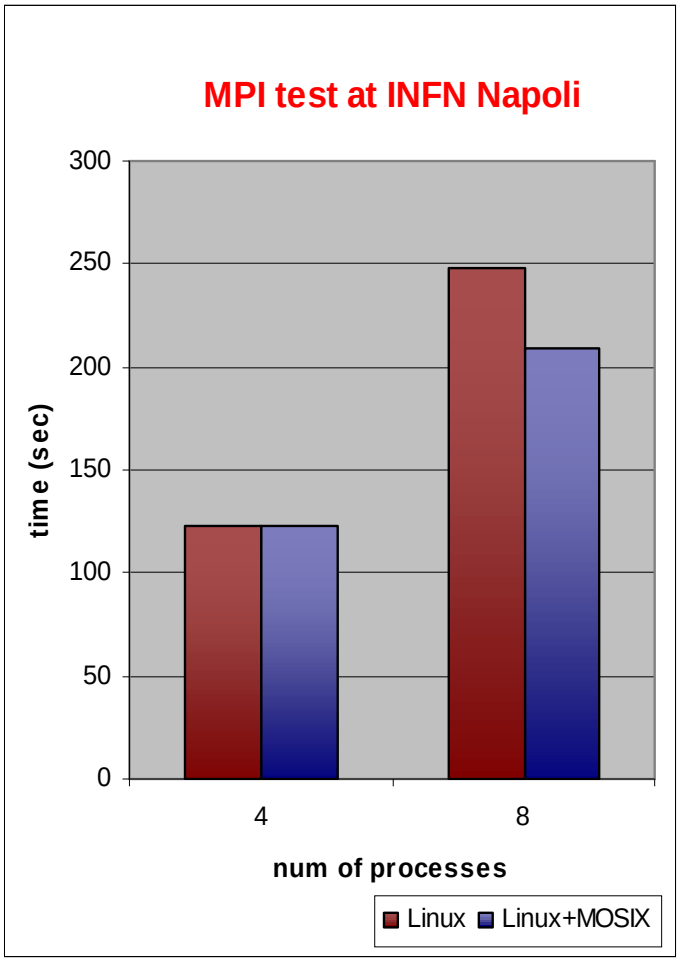
The time employed in this last case is expected to be **two times** the time employed in the first case.

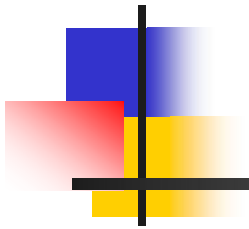
MPI test results



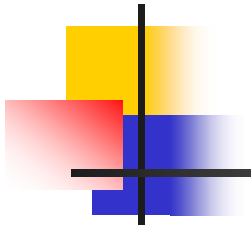
	num. of processes	
Operating System	4	8
Linux	**123**	**248**
Linux+MOSIX	**123**	**209**

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



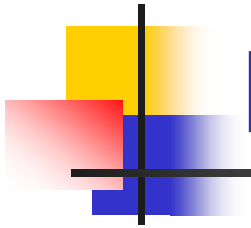


Future directions: DFSA and GFS



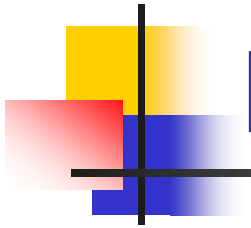
Introduction

- 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

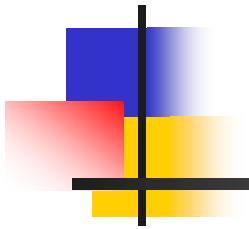
- DFSA was designed to reduce the extra overhead of executing I/O oriented system-calls of a migrated process
- The Direct File System Access (DFSA) provision extends the capability of a migrated process to perform some I/O operations locally, in the current node.
- This provision reduces the need of 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)



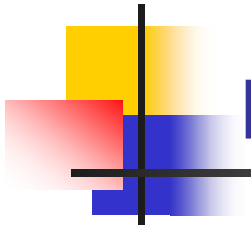
DFSA-enabled filesystems

- 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

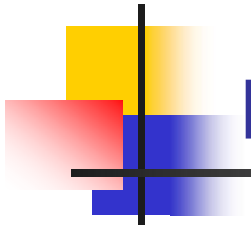


Conclusions



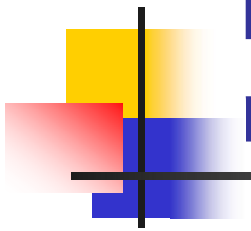
Environments that benefit from MOSIX (1/2)

- **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



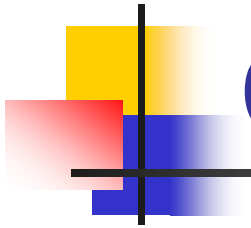
Environments that benefit from MOSIX (2/2)

- **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



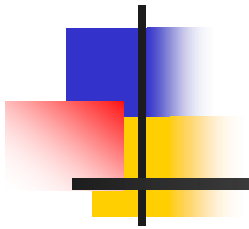
Environments currently not benefit much from MOSIX

- **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

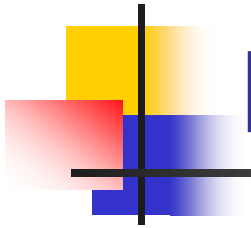


Conclusions

- 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



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**