The MOSIX Algorithms for Managing Cluster, Multi-Clusters, GPU Clusters and Clouds

Prof. Amnon Barak
Department of Computer Science
The Hebrew University of Jerusalem

http:// www . MOSIX . Org

Background

Most cluster and cloud packages evolved from batch dispatchers

- View the cluster/Cloud as a set of independent nodes
  - One user per node, cluster partition for multi-users
- Use static allocation of jobs to nodes
- Place the burden of management on the users

So far a cluster/Cloud OS has not been developed

- Reasons: no industry standards, complexity of development, massive investment, architecture and OS dependency

The MOSIX project

R&D of a Multi-computer Operating System (MOS)

- **Formally: multi-computers are distributed memory (shared nothing) architectures: clusters, multi-clusters, Clouds**

- **Geared for HPC**
- **Research emphasis: management algorithms**
- **Development: infrastructure and tools**

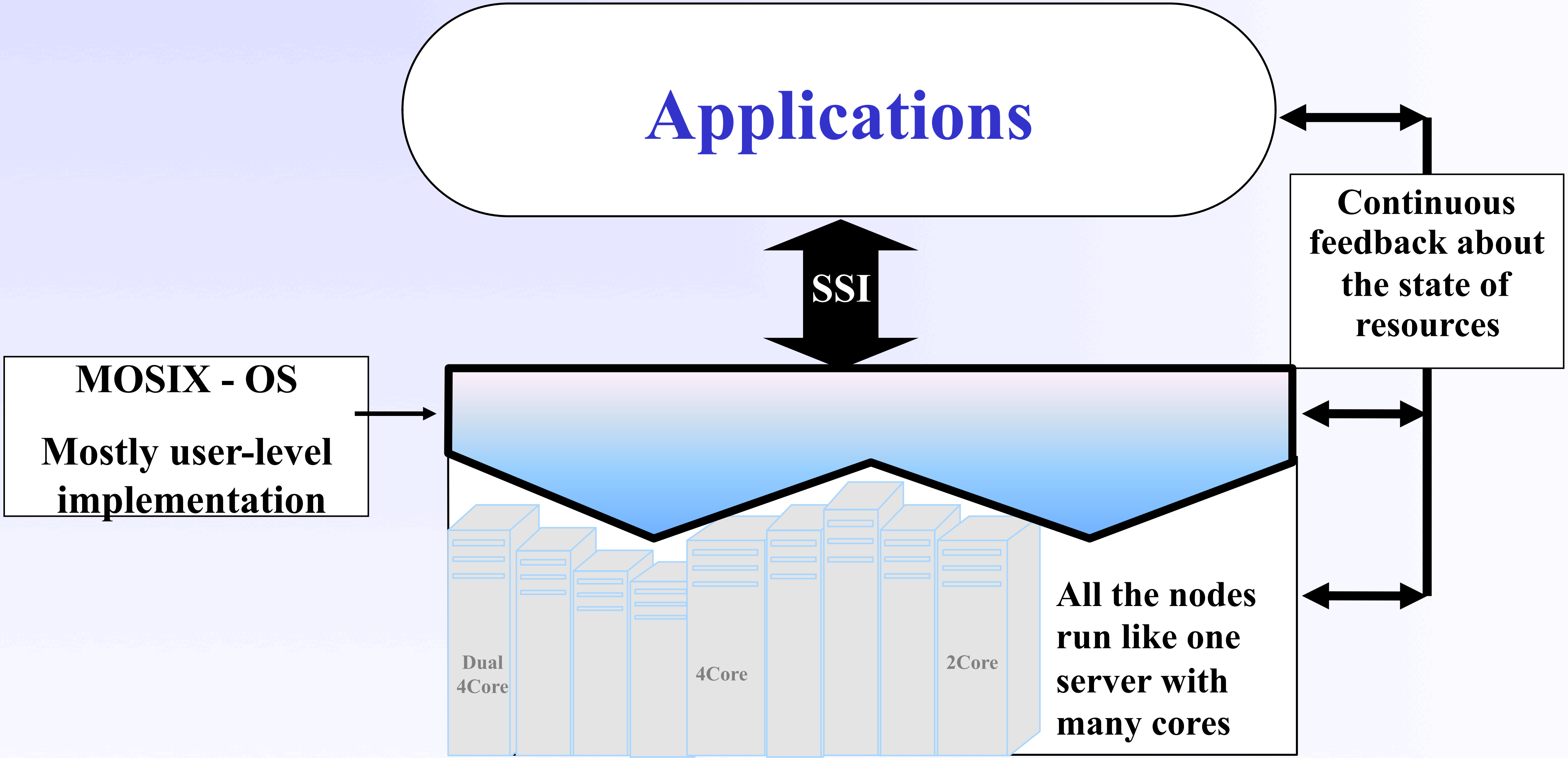
Goal: a production system that people can use

The MOS for UNIX (MOSIX)

A multi-computer OS with decentralized management

- Based on Unix (Linux)
- Provides a single-systems image
  - As if using one computer with multiple CPUs
- Geared to reduce the management complexity to users
  - The user's "login-node" environment is preserved
  - Automatic distribution of processes, e.g. load-balancing
  - No need to "login" or copy files to remote nodes
  - No need to link applications with special libraries
  - Limited support for shared-memory

MOSIX is a unifying management layer

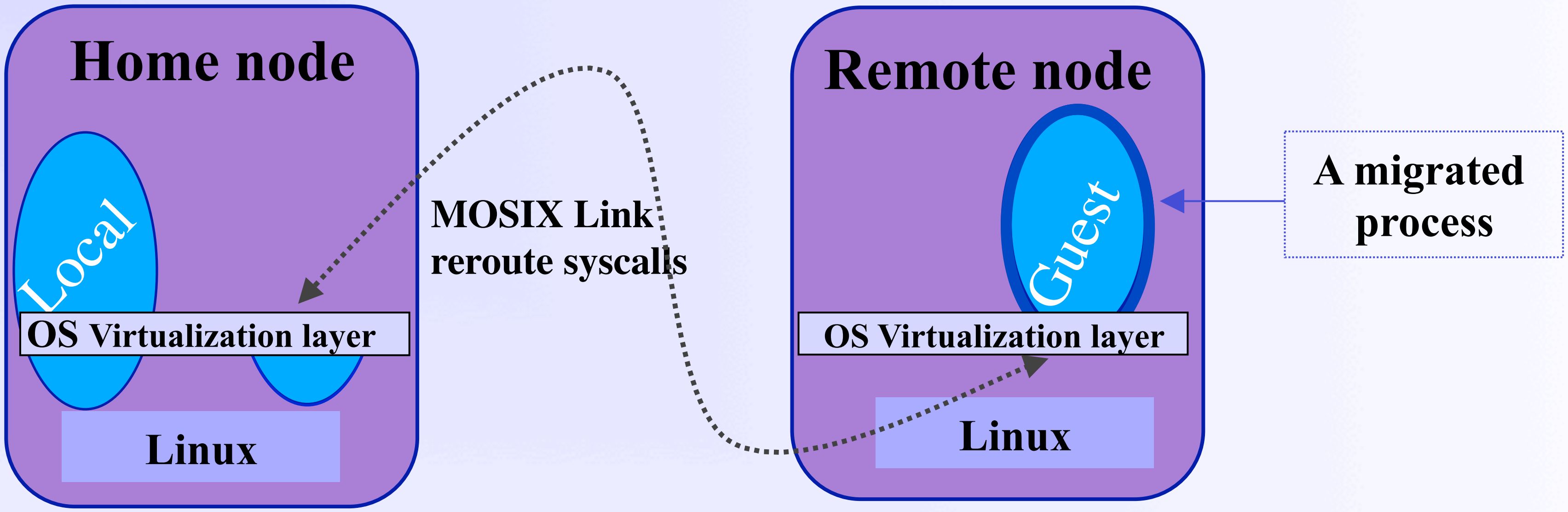


The main software components

1. Preemptive process migration
   - Can migrate a running processes anytime
   - Like a course-grain context switch
     - Implication on caching, scheduling, resource utilization
2. OS virtualization layer
   - Allows a migrated process to run in remote nodes
3. On-line algorithms
   - Attempt to optimize a given goal function by process migration
     - Match between required and available resources
   - Information dissemination – based on partial knowledge

Note: features that are taken for granted in shared-memory systems, are not easy to support in a cluster

Process migration - the home node model



- Process migration – move the process context to a remote node
  - System context stay at "home" thus providing a single point of entry
- Process partition preserves the user's run-time environment
  - Users need not care where their process are running

The OS virtualization layer

- A software layer that allows a migrated process to run in remote nodes, away from its home node
  - All system-calls are intercepted
    - Site independent sys-calls are performed locally, others are sent home
  - Migrated processes run in a sandbox
- Outcome:
  - A migrated process seems to be running in its home node
  - The cluster seems to the user as one computer
  - Run-time environment of processes are preserved - no need to change or link applications with any library, copy files or login to remote nodes
- Drawback: increased (reasonable) communication overhead

Reasonable overhead:

Linux vs. migrated MOSIX process times (Sec.), 1Gbit-Ethernet

Application	RC	SW	JEL	BLAT
Local - Linux process (Sec.)	**723.4**	**627.9**	**601.2**	**611.6**
Total I/O (MB)	0	90	206	476
Migrated process- same cluster	**725.7**	**637.1**	**608.2**	**620.1**
slowdown	**0.32%**	**1.47%**	**1.16%**	**1.39%**
Migrated process to another	**727.0**	**639.5**	**608.3**	**621.8**
cluster (1Km away) slowdown	**0.5%**	**1.85%**	**1.18%**	**1.67%**

Sample applications:

RC = CPU-bound job
JEL = Electron motion

SW = Proteins sequences
BLAT = Protein alignments

On-line management algorithms

- Competitive algorithms for initial assignment of processes to the best available nodes (2 papers in IEEE PDS)
  - Gossip algorithm to support a distributed bulletin board (Concurrency P&E)
- Process migration
    - For load-balancing and from slower to faster nodes (several papers)
    - From nodes that run out of free memory, IPC optimizations
    - Administration of a multi-cluster (CCGrid05)
    - Parallel compression of correlated files (Cluster07)
    - Fair (proportional) share node allocation (CCGrid07)
    - Cloud economy (AAMAS2008, GECON2008, Grid2008)
    - Job migration by combining process and VM migration (Cluster08)
  - Research in progress
    - GPU cluster computing

Resource discovery by a “gossip algorithm”

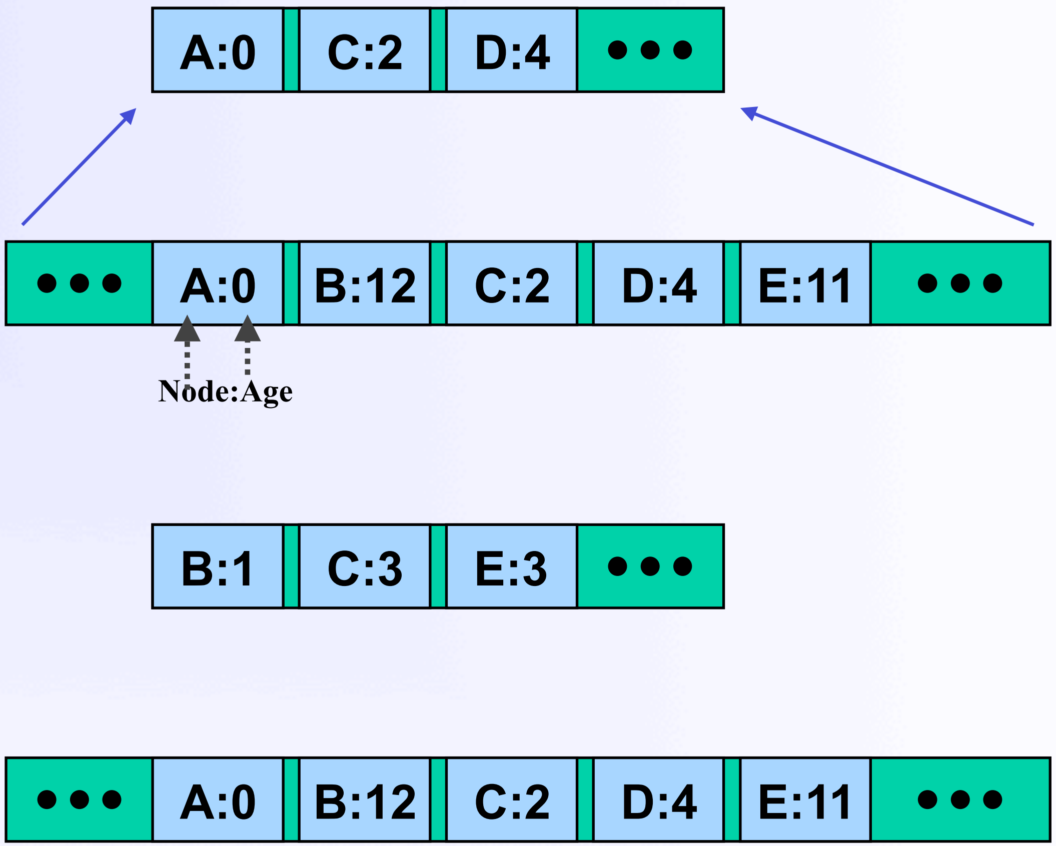
- **All the nodes disseminate information about relevant resources: CPU speed, load, memory, IPC, I/O local/ remote**
  - **Info exchanged in a random fashion - to support scalable configurations and overcome node failures**
- **Useful for initial allocation and process migration**
  - **Example: a compilation farm - assign the next job to least loaded node**
- **Main research issues:**
  - **How much/often info should be circulated**
  - **How long to use old information (Mitzenmacher)**
  - **How it scales up**

Distributed bulletin board

- **An n node cluster/Cloud system**
  - **Decentralized control**
  - **Nodes can fail at any time**
- ***Each node maintains a data structure (vector) with an entry about selected (or all) the nodes***
- **Each entry contains:**
  - **State of the resources of the corresponding node, e.g. load**
  - **Age of the information (tune to the local clock)**
- **The vector is used by each node as a distributed bulletin board**
  - **Provides information about allocation of new processes**

Information dissemination algorithm

- **Each time unit:**
  - **Update the local information**
  - **Find all vector entries that are up to age *t (a window)***
  - **Choose a random node**
  - **Send the window to that node**
- **Upon receiving a window**
  - **Update the received entries age**
  - **Update the entries in which the newly received information is newer**



Main results

For an n node system we showed how to find

- **The number of entries that poses information about node N with age up to T**

$$X(T) = \frac{ne^{nT/(n-1)}}{n-1+e^{nT/(n-1)}}$$

- **The expected average age of vector (A_w expected age of the window)**

$$A_v = \frac{1}{1-(1-1/(n-1))^{X(T)}} + A_w$$

- **The expected number of entries with age below t :**

$$\begin{cases} X(t) & t \le T \\ n\left[1-(1-1/(n-1))^{X(T)(t-A_w)}\right] & t > T \end{cases}$$

- **The expected maximal age**

$$\frac{\log n + \gamma}{X(T)\log(1-1/(n-1))}$$

Outcome: we can guarantee age properties of the vector entries

Load-balancing

Heuristics: reduce variance between pairs of nodes

- **Decentralized -** pair-wise decisions
- **Responds** to load imbalances
- **Migrate** from over-loaded to under-loaded nodes or form slower to faster nodes
- **Competitive** with the optimal allocation
- **Near optimal** performance
- **Greedy**, can get to a local minimum
  - **Why: placement problem is NP-hard**

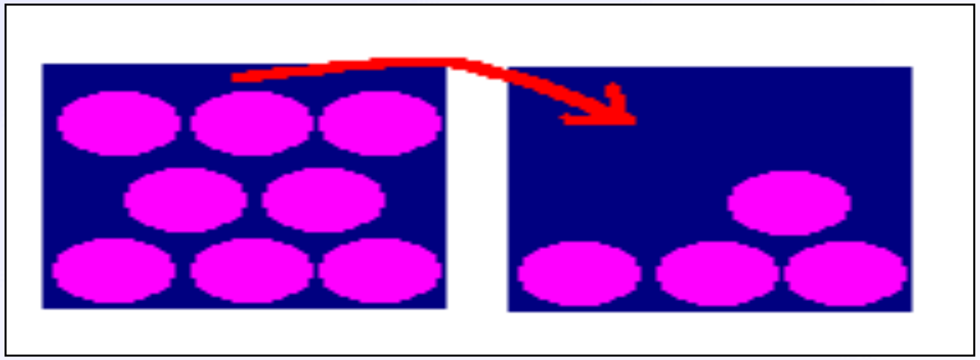
Load balancing algorithms

- **When** - Load difference between a pair of nodes is above a threshold value
- **Which** - Oldest process (assumes past-repeat)
- **Where** - To the known node with the lowest load
- Many other heuristics

- Performance: our online algorithm is only ~2% slower than the optimal algorithm (which has complete information about all the processes)

Memory ushering

- **Heuristics:** **initiate process migration from a node with no free memory to a node with available free memory**
- **Useful: when non-uniform memory usage (many users) or nodes with different memory sizes**
- **Overrides load-balancing**



- Recall: **placement problem is NP-hard**

Memory ushering algorithm

- **When** - free memory drops below a threshold
- **Where** - the node with the lowest load, to avoid unnecessary follow-up migrations
- **Which** - smallest process that brings node under threshold
  - To reduce the communication overhead

IPC optimizations

- **Reduce the communication overhead by migrating data intensive processes “near” the data**
- **Reduce IPC by migrating communicating processes to the same node (IPC via shared-memory)**