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**

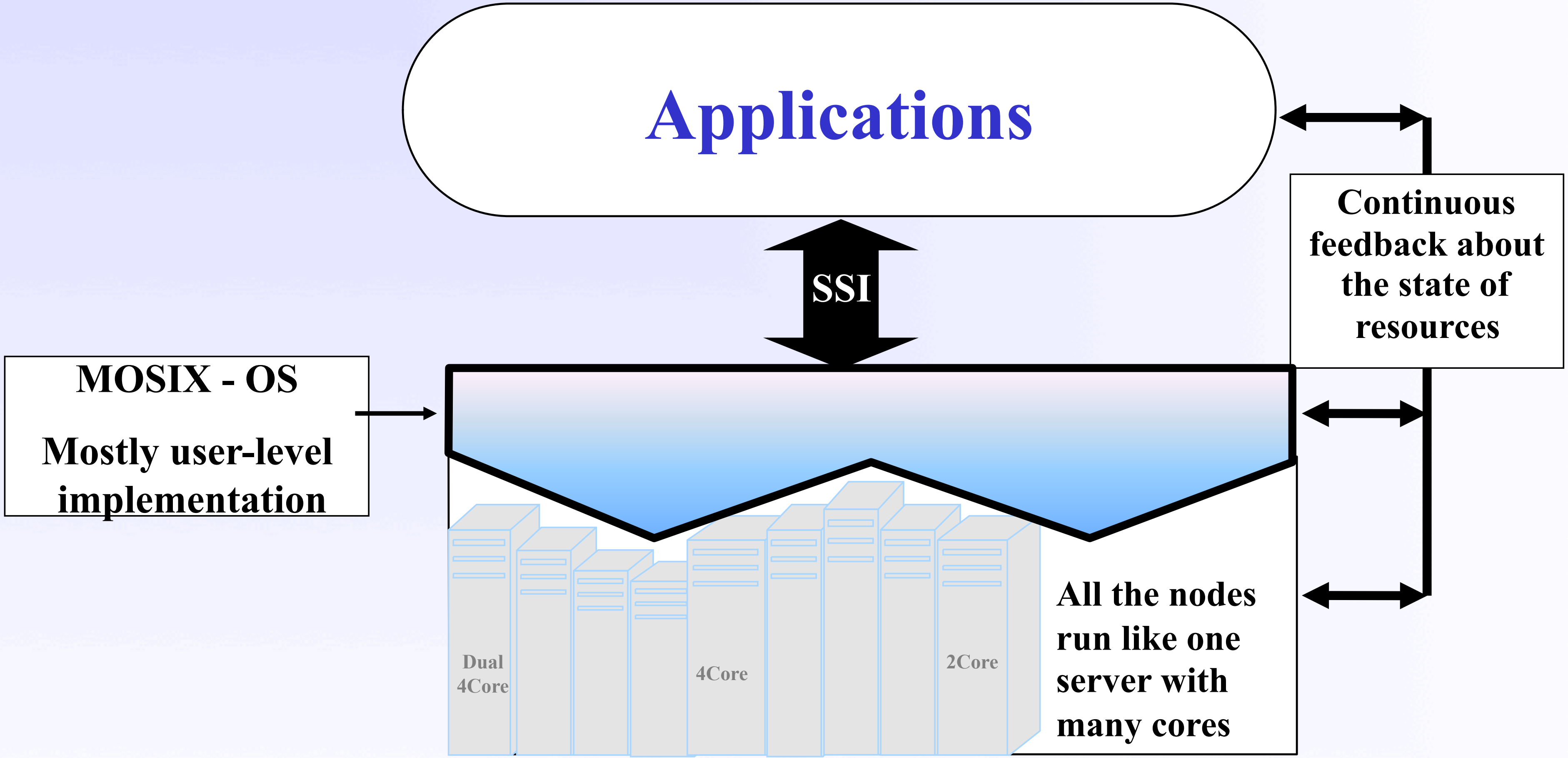
Goal: a production system that people can use

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

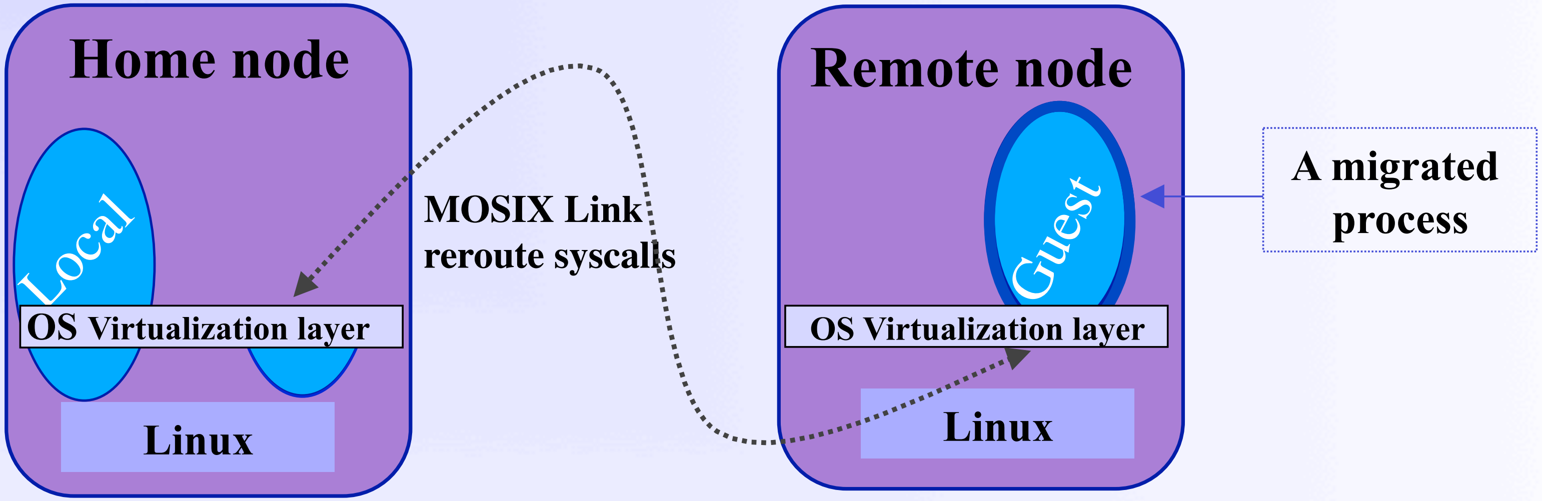


The main software components

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"

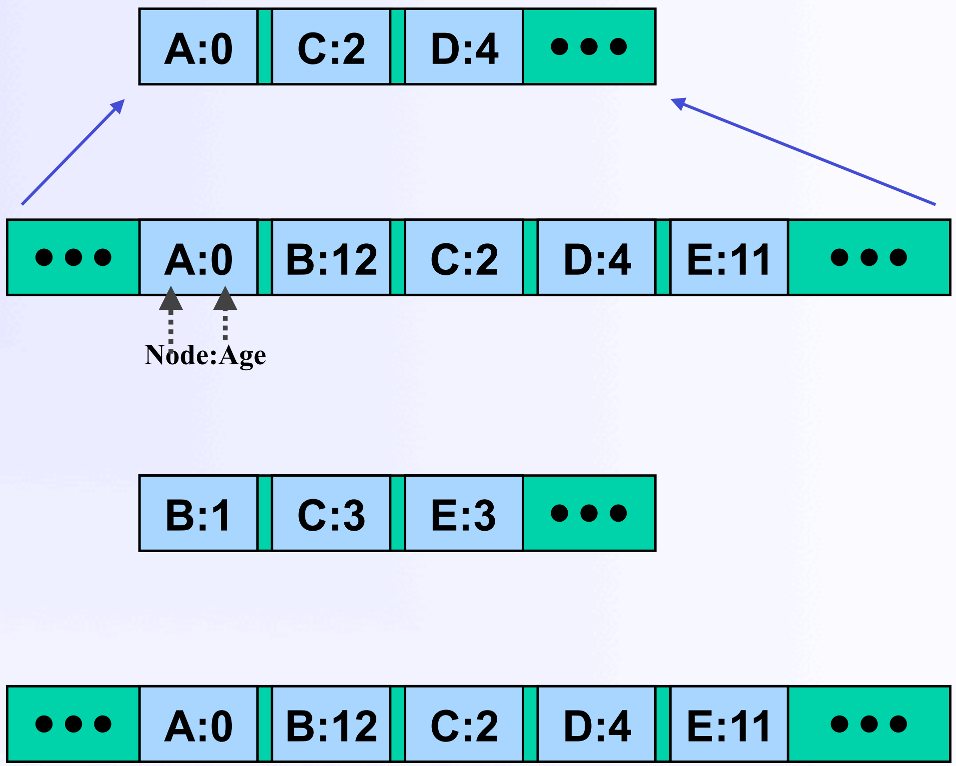
- All the nodes disseminate information about relevant resources: CPU speed, load, memory, IPC, I/O local/ remote
  - Info exchanged in a random fashion - to support scalable configurations and overcome node failures
- Useful for initial allocation and process migration
  - Example: a compilation farm - 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 :**

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

- **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**

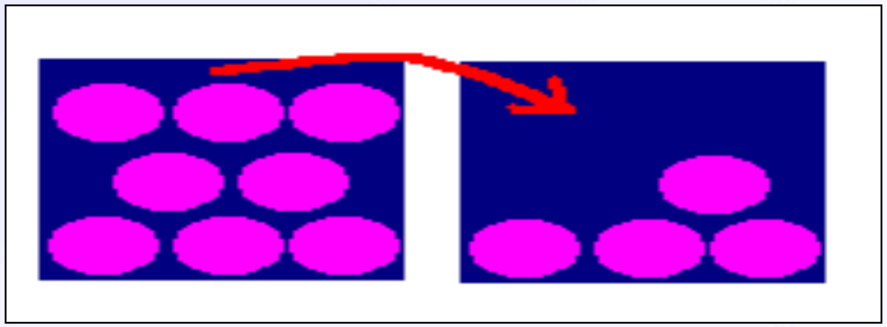
Load balancing algorithms

- 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)

Administrating a multi-cluster

Model: a federation of clusters, servers and workstations whose owners wish to cooperate from time to time

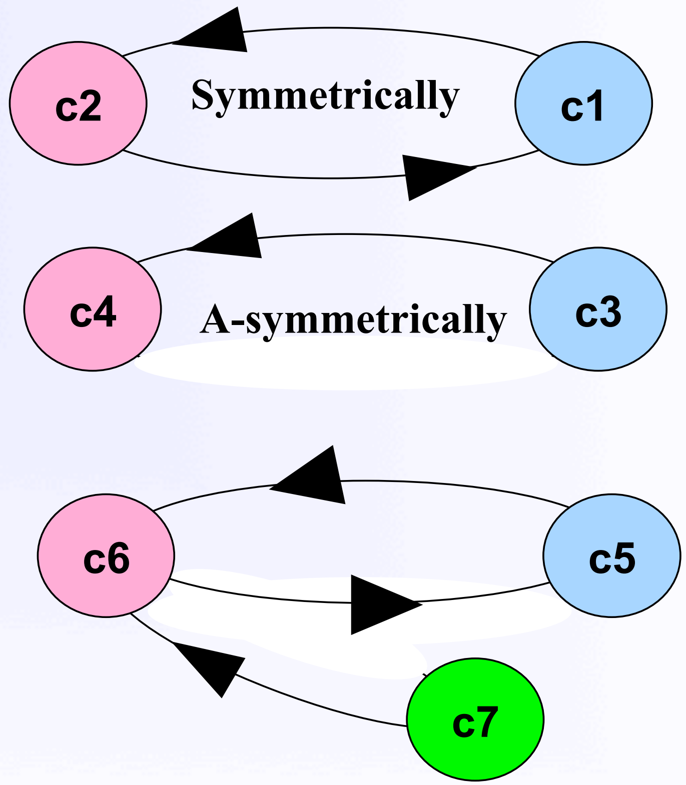
- Collectively administrated
  - Each owner maintains its private cluster
    - Determine the priorities vs. other clusters
  - Clusters can join or leave at any time
  - Dynamic partition of nodes to private virtual clusters
  - Users of a group access the Cloud via their private cluster and workstations

Outcome: each cluster and the whole Cloud perform like a single computer with many processors

The priority scheme

- Cluster owners can assign priorities to processes from other clusters
  - Local and higher priority processes force out lower priority processes
- Pairs of clusters could be shared, symmetrically(C1-C2) or asymmetrically(C3-C4)
- A cluster could be shared (C6) among other clusters (C5, C7) or blocked for migration from other clusters (C7)
- Dynamic partitions of nodes to private virtual clusters

Outcome: flexible use of nodes in shared clusters



When priorities are needed

- Scenario 1: one cluster, some users run many jobs, depriving other users from their fair share
- Scenario 2: some users run long jobs while other user need to run short jobs
- Scenario 3: several groups share a common cluster
- Solution: partition the cluster to several sub-clusters and allow each user to login to only one sub-cluster
  - Processes of local users (in each sub-cluster) has higher priority over all guest processes from other sub-clusters
  - Users in each sub-cluster can still benefit from idle nodes in other sub-clusters

Support disruptive configuration

When a private cluster is disconnected:

- **All guest processes move out**
  - **To available nodes or to the home cluster**
- **All migrated processes from that cluster move back**
  - **Returning processes are frozen (image stored) on disks**
    - **Try to do that for 100 jobs of 2GB each**
  - **Frozen processes are reactivated gradually**

Goal:

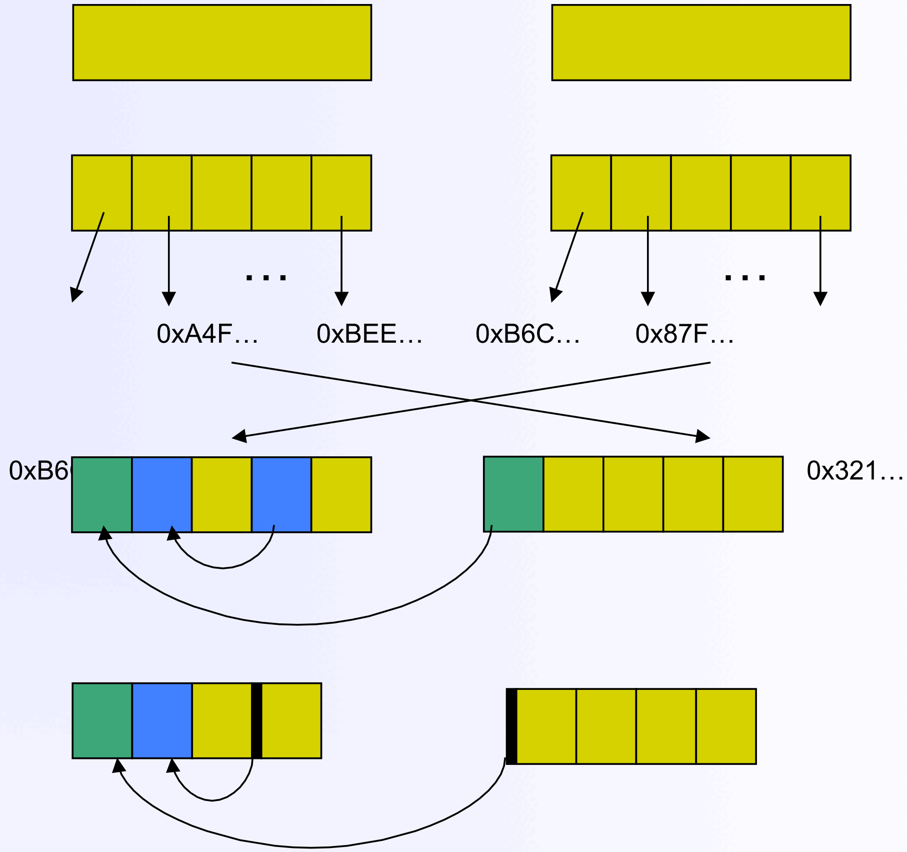
- **Preserve long running processes**

Parallel compression of correlated files

- Method 1: concurrent serial compressors - simultaneously compress the memory images at each node, then send to the repository
  - Problem: takes longer to compress and send a memory image than sending it uncompressed
- Method 2: Assumption: memory images of a parallel job are correlated:
  - The processes use the same code and libraries
  - Typically, these processes share the same database
    - There are large substrings common to these images
- Idea: Eliminate inter-file redundancy

Parallel algorithm

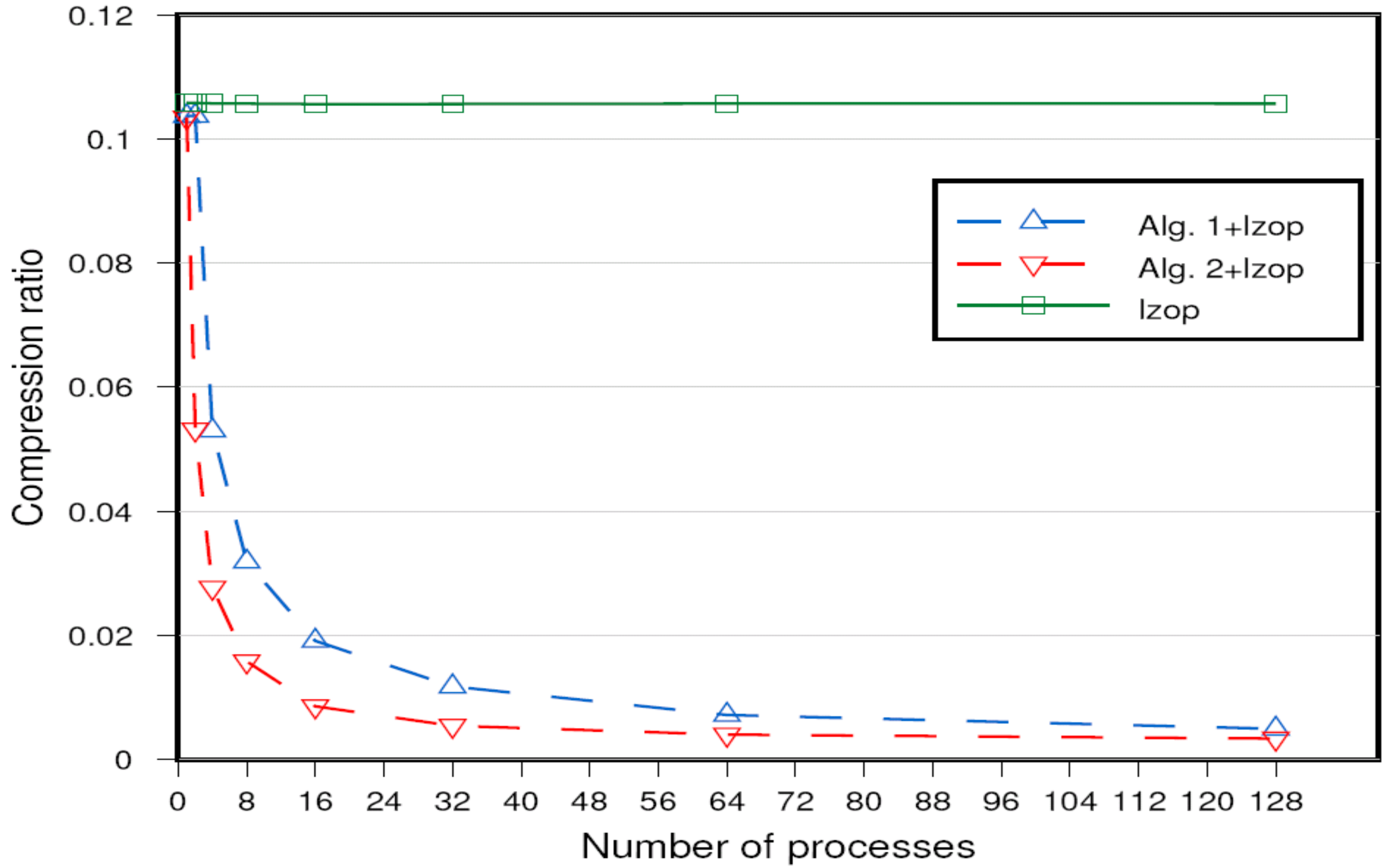
- **For each memory image:**
  - **Partition the file into equal chunks**
  - **Obtain hash value for each chunk**
  - **Exchange hash values with the other nodes to find duplicate chunks**
  - **Compress the file, replacing duplicate chunks with pointers**
    - Advantage: no need to transfer the whole file to compare chunks, just the hash values
    - The basis of the rsync protocol



- **Improvement: use serial compressors on results to further compress each file**

Example: RxRySpace compression ratios

- **Medical application creates 2D projections of 3D CT data**
- ***Average image size: 509MB, Total size 99GB***
- ***Run on 64 dual-core nodes with 2GB RAM***

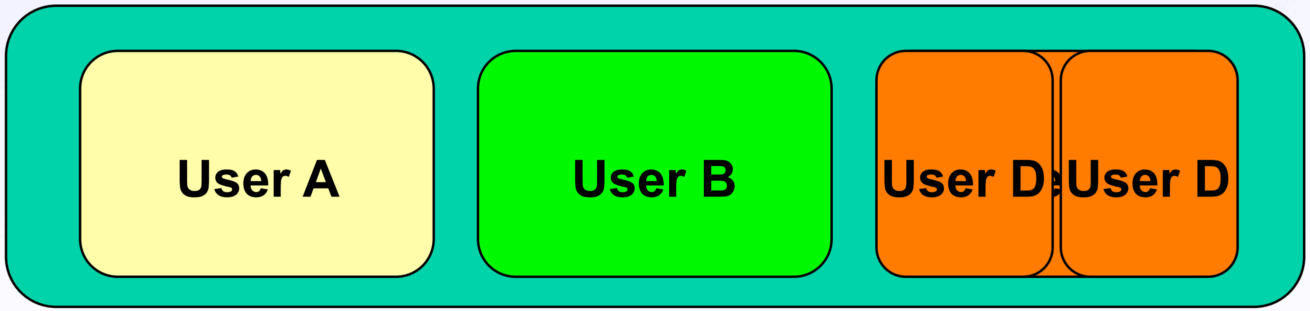


Fair-share node allocation

- **Most cluster and Cloud management systems do not provide adequate means for fair share allocation, e.g. as in multi-core systems**
  - **New users may need to wait a long time until scheduled to run**
- We developed on-line algorithms and a runtime environment for fair share scheduling in a cluster

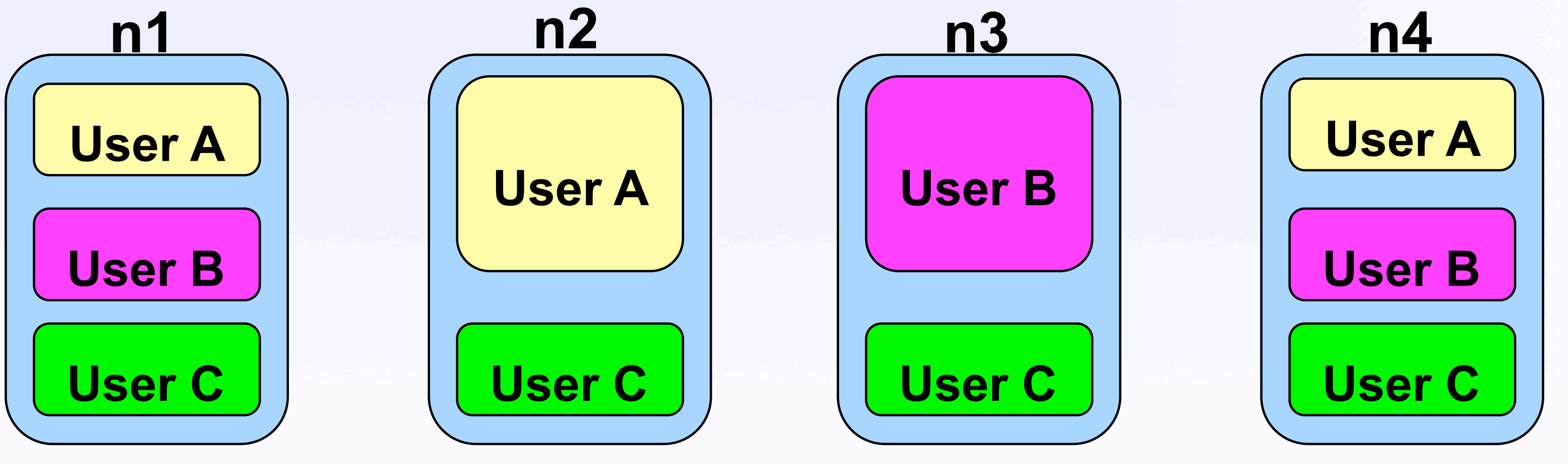
Single-node Fair-Share (FS) scheduling

- **A scheduling strategy for proportional allocation of the CPU to users**
  - **Users get a predefined percentage of the CPU**
    - **As opposed to the OS default which is equal distribution among processes**
  - ***Lottery* and *Stride* are two well known algorithms for FS scheduling in a single-node**
  - **VMware & Xen supports proportional share scheduling of VMs**



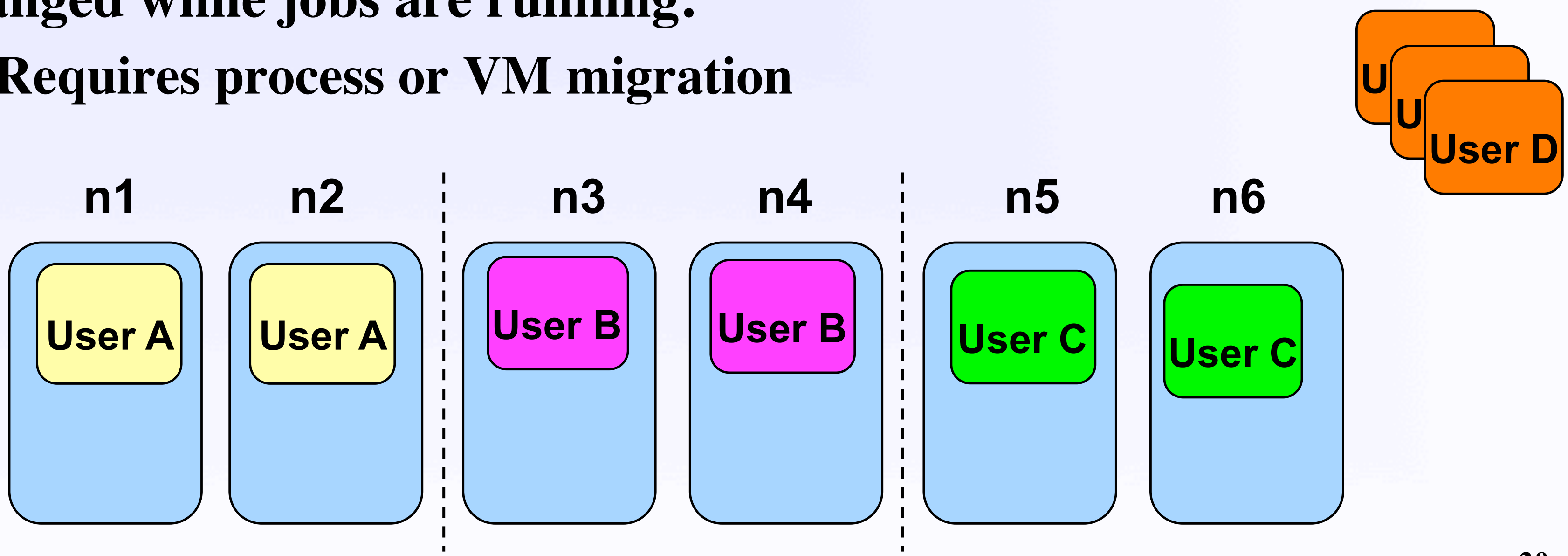
Cluster FS by time-sharing *(Horizontal Partitioning)*

- Cluster-wide proportional resource allocation to all users
- Time sharing *[Arpaci-Dusseau et al PDPTA 1997]*
  - Resources are allocated proportionally within each node using a single node scheduler (like *stride*)
  - Based on the desired proportions and the current allocation, a supervisor algorithm determines the local proportion allocated to each user on each machine



Cluster FS by space-sharing (vertical partitioning)

- Proportional allocation of disjoint sets of nodes to users (one user per node)
  - Non-preemptive: size of sets can be changed only when jobs are started or finished
    - Common in batch systems
  - Preemptive space-sharing: size of sets can be dynamically changed while jobs are running.
    - Requires process or VM migration

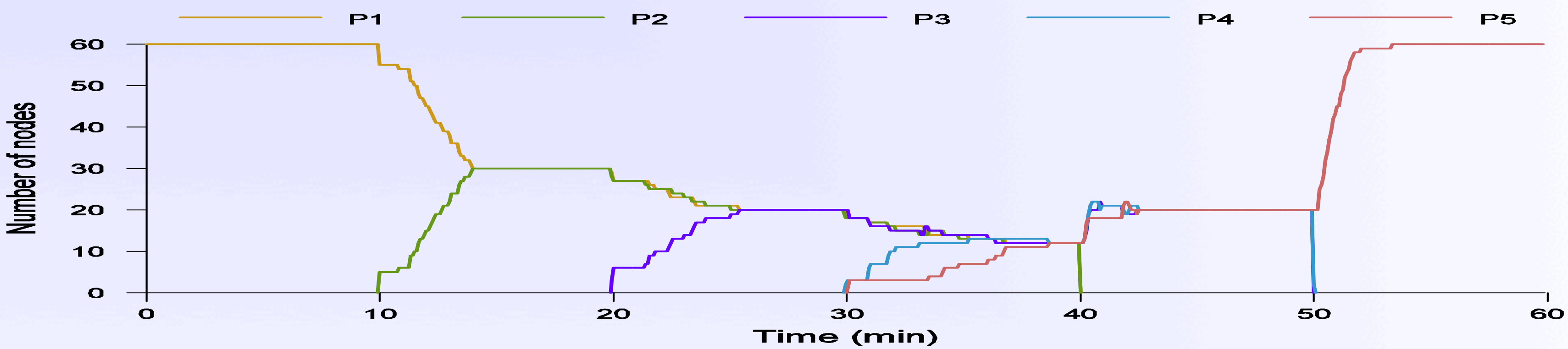


A distributed dynamic proportional-share scheduler

- A distributed, preemptive space-sharing scheduler was developed
- A central algorithm, maintains one queue for all the users
- Our distributed algorithm (without a single queue)
  - Each node continuously monitors the current allocation of nodes to users
  - The nodes with the highest id that is already allocated to a user which is using more nodes than its entitled share becomes a potential candidates to be reallocated
    - This node adjust the local MOSIX priority, to allow users which deserve more nodes to obtain nodes if in need
  - In case of non integer shares, the algorithm circulate some nodes among different users
    - 2 users 3 nodes

Example on a 60 nodes cluster

- Gradually adding up to 5 users
- Then gradually removing 2 partners at a time



Reach the clouds

Cloud computing allows user to run applications and store data on remote clusters/data-centers via the internet

- Some providers: Amazon, Google, IBM
- Relevant issues: cost, convenience, trust
- The MOSIX "reach the clouds" (MRC) tools:
  - Users can run applications clouds, while still using local files
    - By exporting local file systems to remote clusters
    - No need to store or copy files in the clouds

Our campus multi-cluster (HUGI)

- 18 production MOSIX clusters ~730 nodes, ~950 CPUs
  - In life-sciences, med-school, chemistry and computer science
  - Priorities among users from different departments
- Sample applications:
  - Nano-technology
  - Molecular dynamics
  - Protein folding, Genomics (BLAT, SW)
  - Weather forecasting
  - Navier-Stokes equations and turbulence (CFD)
  - CPU simulator of new hardware design (SimpleScalar)S

Current project: MOSIX GPU cluster

- Heterogeneous computing systems can dramatically increase the performance of parallel applications
- Currently, applications that utilize GPU devices, run their device code only on local devices, were they started
- The MOSIX Virtual OpenCL (VCL) cluster platform can run unmodified OpenCL applications transparently on clusters with many devices.
- VCL provides an OpenCL platform in which all the cluster devices are seen as if they are located in the hosting-node
  - Benefits OpenCL applications that can use many devices concurrently

VCL highlights

- Geared for running applications on clusters
  - Applications can make use of both multi-core CPUs and many GPUs
- Especially benefits parallel applications that can use multiple devices concurrently, e.g. HPC
  - Supports an OpenMP-like programming environment and MPI-like concurrent access to cluster-wide devices
- Provides a shared pool of devices for many users
  - Applications can even be started from workstations without GPU devices

The VCL run-time model

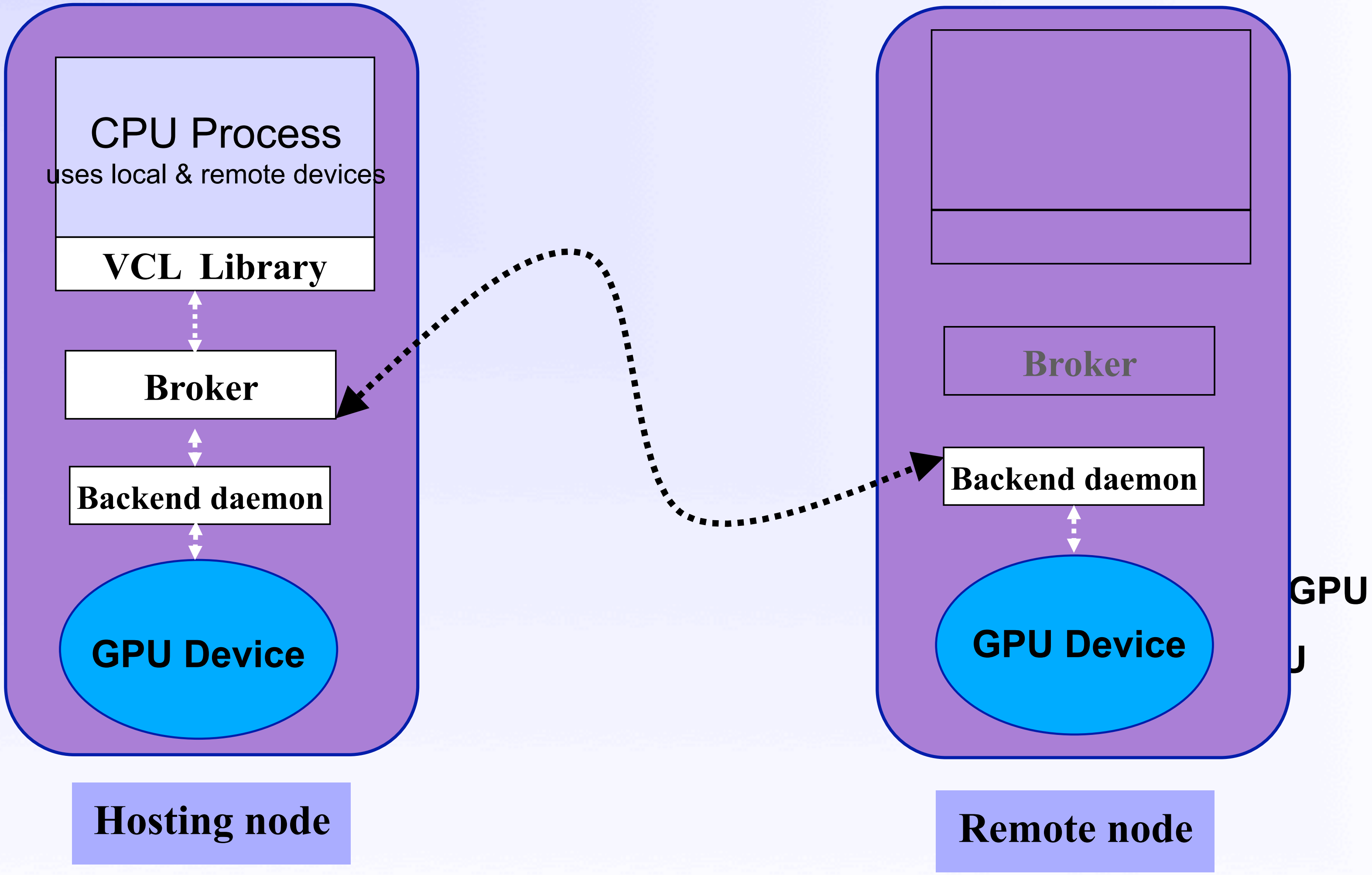
- VCL is designed to run applications that combine a CPU process with parallel computations on many GPUs

- The CPU process runs on a single “hosting” node
  - Responsible for the overall program flow
    - May perform some computation
    - Can be multi-threaded, to utilize available cores in the hosting node
- The GPU programs (kernels) can run on multiple devices, e.g. GPUs, CPUs, APUs
    - The locations of the devices is transparent to the program

Combines benefits of OpenMP and MPI

- **Applications benefit from:**
  - **Reduced programming complexity of a single computer, as in OpenMP**
    - **Availability of shared-memory, multi-threads and lower level parallelism**
    - **Recall: development of parallel applications is simpler in OpenMP than in MPI**
  - **Concurrent access to cluster-wide devices, as in MPI**
- **Outcome:**
  - **Full benefit of VCL manifest with applications that utilize many devices**
  - **The VCL model is particularly suitable for applications that can make use of shared-memory on many-core computers**

Using multiple GPUs in a cluster



SHOC - FFT performance on a cluster

- 256 MB buffer, 1000 – 8000 iterations on 1, 4 and 8 nodes

Number of Iterations	Native OpenCL Time (Sec.) no VCL	VCL - 4 Nodes		VCL - 8 Nodes	
		Time (Sec.)	Speedup	Time (Sec.)	Speedup
1000	42.34	19.27	2.19	16.29	2.60
2000	82.25	30.11	2.73	22.03	3.73
4000	162.17	52.58	3.08	33.37	4.86
8000	321.91	97.53	3.29	55.95	5.74