Distributed Operating Systems

Cluster Systems

Ewa Niewiadomska-Szynkiewicz
ens@ia.pw.edu.pl

Institute of Control and Computation Engineering
Warsaw University of Technology

E&IT Department, WUT

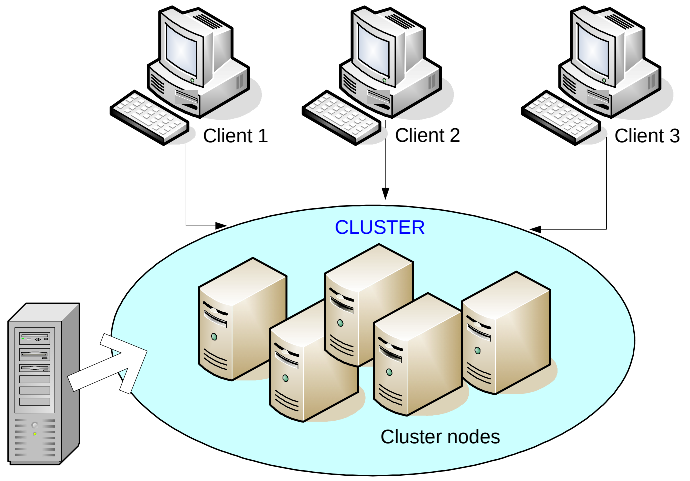
Cluster Systems

1. Cluster systems
    - Introduction
    - Clusters Categorization
    - Beowulf Cluster
    - Linux Virtual Server (LVS)
    - SSI Clustering Systems
2. MOSIX System
3. OpenSSI System
4. Kerrighed System
5. Clusters – Comparison

Cluster Systems – Introduction

Computer Cluster

A group of **loosely coupled computers** (often homogenous) commonly, but not always, connected to each other through **fast local area networks.** The components of a cluster work together closely so that in many respects they can be viewed as a single computer. Clusters are usually deployed to improve performance and/or availability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.



Cluster nodes – cluster components (computers)

Communication model – *peer-to-peer* (typical)

Clusters Categorization

High Performance Computing Clusters (HPC)

The purpose – **improving performance**. The components of a cluster work together as a single supercomputer. The programs to be executed should be implemented in parallel versions.
Example: **Beowulf cluster**.

Load Balancing Cluster – a group of HP clusters. Clusters operate by having all workload come through one or more load-balancing front ends, which then distribute it to a collection of back end.
Example: **Server cluster**: the goal – balancing of the load of all servers in the system.

High Available Computing Clusters (HA)

The purpose – **improving the availability** of services which the cluster provides. They operate by having **redundant nodes**, which are then used to provide service when system components fail.

HA cluster implementations attempt to manage the redundancy inherent in a cluster to eliminate single points of failure. The most common size for an HA cluster is two nodes, which is the minimum requirement to provide redundancy.

Beowulf Style Cluster

Beowulf – technology of connecting computers to provide "virtual supercomputer" capable of performing tightly coupled parallel HPC computation. The components of a cluster work together as a single computer with single IP address.

- Goal: high performance parallel computing cluster on inexpensive computer hardware and software (usually identical PC computers running a free and open source Unix-like system, such as: BSD, Linux, Solaris, networked into a small TCP/IP LAN).
- Typical access to the client nodes via remote login.
- Two network interfaces, for:
  - *internal communication*
  - *external communication.*

Advantage: easy to build.

Drawback: connection to the outer network via a single node, lack of shared memory and load-balancing algorithms.

Linux Virtual Server (LVS)

The purpose was to build a high-performance and highly available server for Linux using clustering, which provides good scalability, reliability and serviceability.

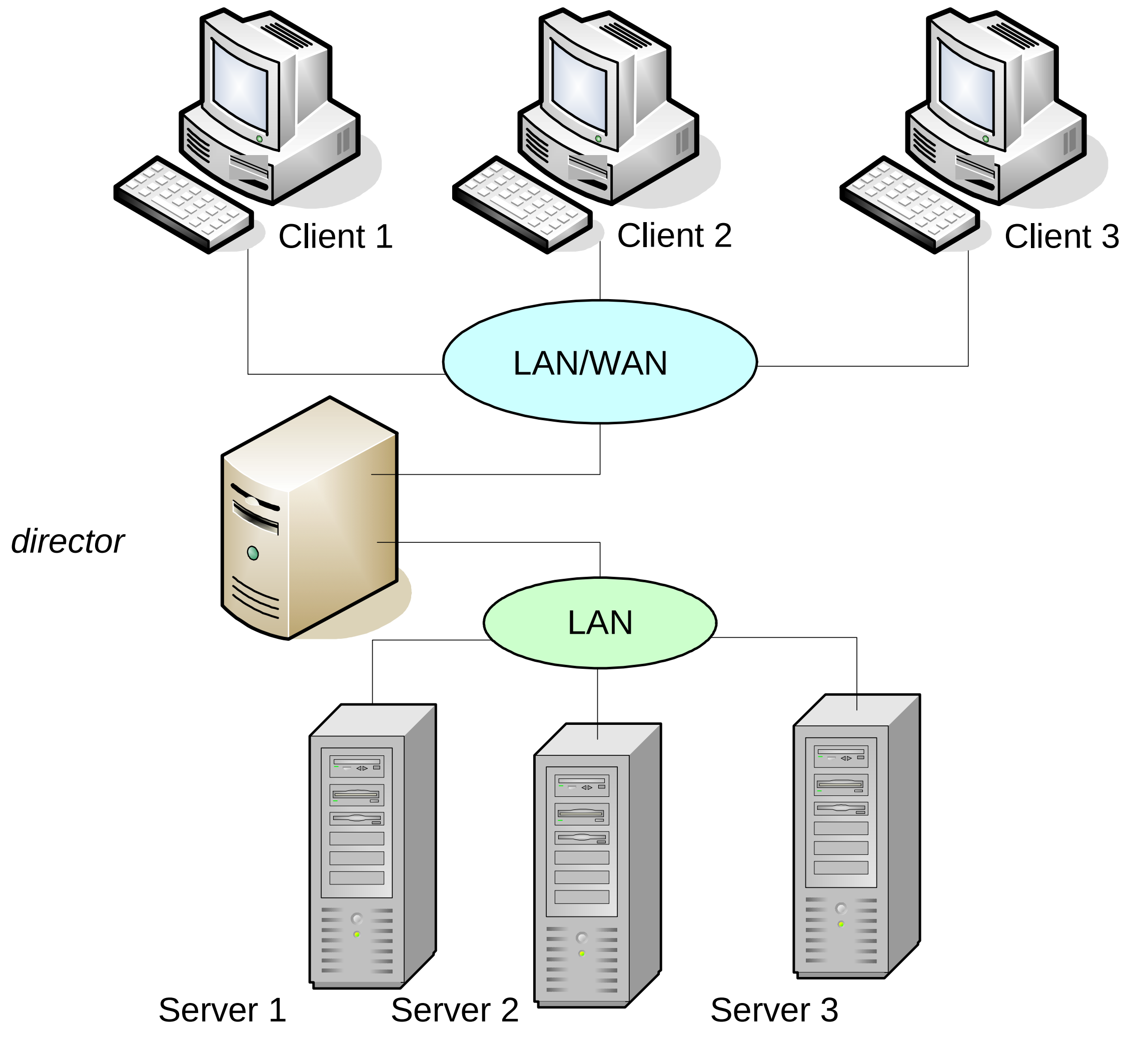
- LVS software allows to distribute the tasks to all servers in the cluster.

- Specialized algorithms for load-balancing and execution monitoring.

Three-layer architecture of the system:

1. **set of servers**
2. **data store system**
3. **load-balancer (*director*)**

LVS – Example of Configuration



SSI Clustering System

Single System Image (SSI)

The property of a system that hides the heterogeneous and distributed nature of available resources and presents them to users and applications as a single unified computing resource
(*the operating system environment is shared by all nodes in the system*).

- User has a global view of the resources available to him irrespective of the node to which they are physically associated.

- SSI can ensure that a system continues to operate after some failure (high availability).

- SSI can ensure that the system is evenly loaded (resource management and scheduling).

- SSI design goals for cluster systems are mainly focus on:
  - *complete transparency of resource management*,
  - *scalability,*
  - *high performance and availability*.

Key Services of SSI

- Single entry point.
- Single user interface.
- Single process space.
- Single memory space.
- Single file hierarchy.
- Single I/O space.
- Single virtual networking.
- Single job management system.
- Single control point and management.
- Checkpointing and process migration.

Checkpointing is a software mechanism to periodically save the process state in memory or disks (this allows the rollback recovery after a failure and dynamic cluster reconfiguration).
Process migration is needed in dynamic load balancing.

MOSIX Systems

MOSIX (***M**ulticomputer **O**perating **S**ystem for un**IX***)

A management system targeted for high performance computing on Linux clusters and multi-clusters (http://www.mosix.org/)

(*Amnon Barak, The Hebrew University of Jerusalem, since 1977*)

OpenMOSIX

The Open Source cluster management system originally forked from MOSIX

(*Moshe Bar, 2002*)

MOSIX2 (MOSIX version 2)

MOSIX2 can manage a cluster and a multicluster (Grid) as well as workstations and other shared resources.

(*Amnon Barak, The Hebrew University of Jerusalem*)

MOSIX – Main Futures

- **No central control or master-slave relationship between nodes:**

Each node operates as an autonomous system, makes its control decisions independently

- **Provides <u>some aspects</u> of a Single System Image:**
  - ***Users can login on any node and do not need to know where their programs run.***
  - ***No need to link applications with special libraries.***

- **Automatic resource discovery and dynamic load distribution:**
  - ***Automatic load-balancing.***
  - ***Automatic process migration from slower to faster nodes and from nodes that run out of free memory.***

- **Migratable sockets** for direct communication between migrated processes.
- **Load balancing.**
- **Checkpoint and recovery.**
- **Tools for automatic installation, configuration and monitoring**
- **Support scalable file systems.**

MOSIX2 – Main Futures

- Can manage a **cluster** and an **organizational grid** **(flexible configurations)**
- Secure run time environment (**sandbox**) for remote (guest) processes.
- Live queuing – queued jobs preserve their full generic Linux environment.
- Supports **batch jobs**, checkpoint and recovery.

Grid features of MOSIX2

- **Collectively administrated**
- Each owner maintains its private cluster
  (determine the priorities vs. other clusters to allow/block process migration)
- Dynamic configuration – clusters can join or leave the grid at any time.
- Dynamic partition of nodes to private virtual clusters.
- Each cluster and the whole grid perform like a single computer with multiple processors.
- Cluster owner can assign different priorities to guest (remote) processes.

Grid: *a federation of homogenous clusters, servers and workstations whose owners wish to cooperate from time to time.*

The System Image Model

Each process has a ***Unique Home-Node*** (**UHN**) –
where it was created (usually the Login node of the user).

The system image model is a cluster, in which every process seems to run at its **UHN** and all the processes of a user's session share the run-time environment of **UHN**.

Process migration – the home node model:

Process that migrate to a remote node **uses its local resources** but **interact with the process' environment via UHN.**

Migrated processes seem to be running in their respective UHN.
(for example: invoking ***gettimeofday()*** get the current time at the UHN)

Migration Control

Decision about migration of processes is made based on:

- Statistics about communication with the kernel
- Statistics about all node's workloads
- Processors' speed (measured after system start up)

Migration processes from slower to faster nodes and from nodes that run out of free memory.

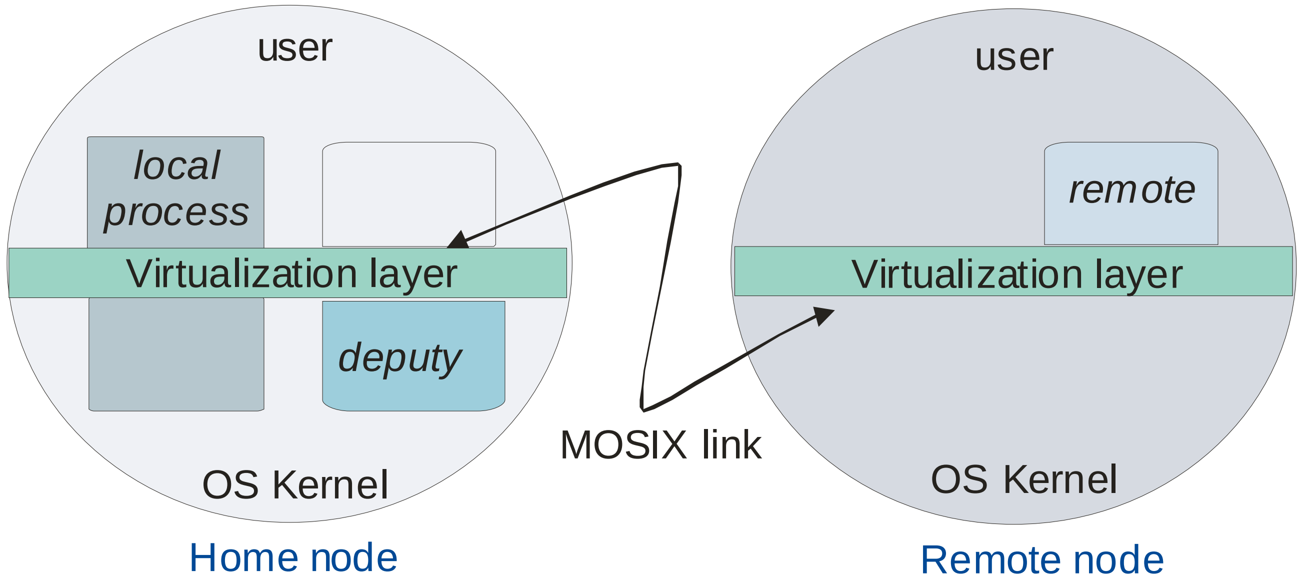
Load balancing algorithms

Migration in MOSIX inefficient for processes with intensive I/O and/or file-system operations.

Migration – drawback: an increased communication overhead.

Process Migration (MOSIX)

Preemptive process migration: to evacuate migrated process from a disconnected cluster
Virtualization software: allows a migrated process to run in remote node, away from its UHN



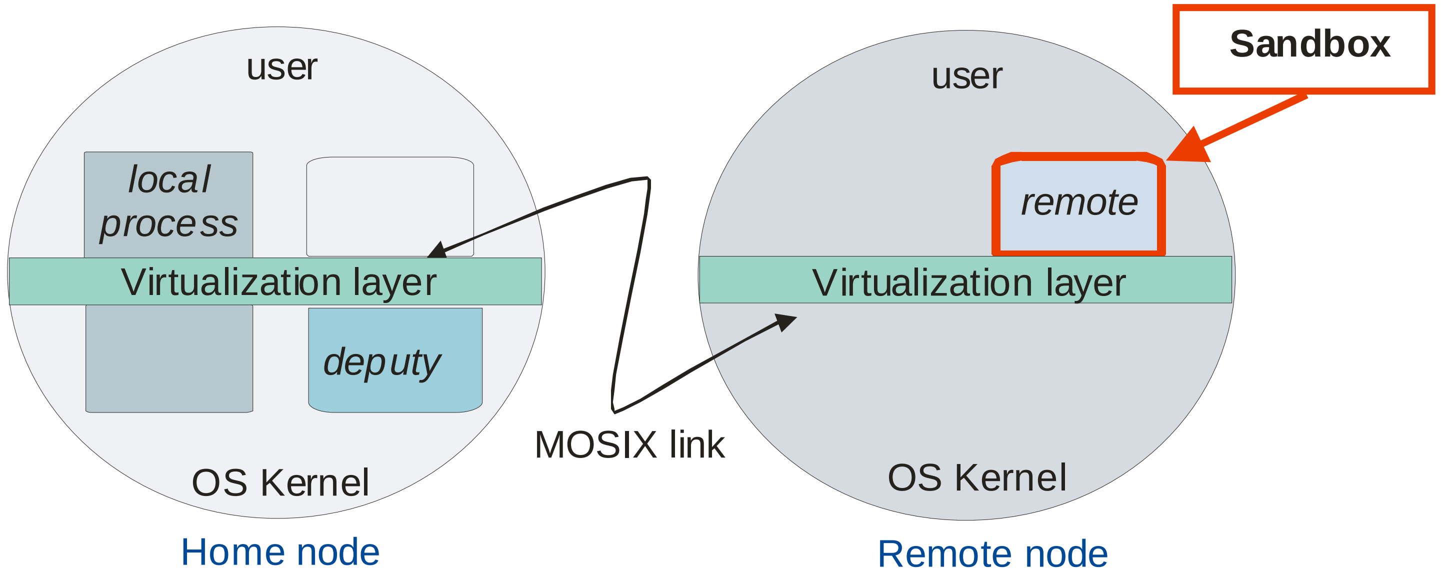
Two contexts of migrated process:

remote (user context) – that can be migrated,
deputy (system context) – UHN dependent, may not be migrated

remote: program code, stack, data, memory maps and register of the process.
deputy: kernel stack, description of the resources attached to the process.

Process Migration (MOSIX2)

A secure run-time environment



The home node model implies:

- Migrated processes run in a **sandbox**
- Guests (remote) process cannot access or modify local resources

<u>Outcome</u>: **remote node is protected**

MOSIX API: /proc

The API is implemented by extending the Linux **/proc** file system tree with a new directory **/proc/mosix**

Files	**Information**
/proc/{pid}/where	Node number where process *pid* is executed (0 – home)
/proc/{pid}/lock	Process *pid* is prevented from migration (1– yes, 0– no)
/proc/{pid}/nmigs	Number of process *pid* migrations
/proc/{pid}/goto	Points the node number for process *pid* to be transfered
/proc/mosix/remote	Directory with information about remote processes curried out on a given node
/proc/mosix/nodes	Information about cluster nodes
/proc/mosix/admin	Directory with configuration files

MOSIX Management

For all users

`mon` – MOSIX monitor (displays current information about MOSIX nodes)

`mosrun` – runs a program under the MOSIX discipline
(i.e. program can migrate;
programs run in “native” Linux mode cannot migrate)

`migrate` – manually migrates a given MOSIX process (*pid*)
Example: **`migrate pid MOSIX_ID`**, **`migrate pid 0`**,
`migrate pid balance`

For administrators

`setpe` – defines the configuration of the local MOSIX cluster
(usually the cluster map is obtained from **`/etc/mosix.map`**)

`mosctl` – group of functions for MOSIX administration
(get information about the status of the node, migration blocking/unblocking, etc.)
Example: **`mostcl stay`** – prevents processes from migrating

MOSIX Management – Examples

1. Run a program on node No 2. Two possibilities:

   - function **mosrun** with adequate option:

     ```
     mosrun –j2 our_program
     ```

   - script execution:

     ```
     runon 2 our_program
     ```

2. Run a program on randomly selected node from nodes with numbers: 2-4, 6, 20-30

   ```
   mosrun –j2-4,6,20-30 our_program
   ```

3. Run a program that mainly performs calculations and allow for its migration:

   ```
   mosrun –h -l -c our_program
   ```

Process Migration – Example

Example:

1) We want to move the process with *pid* = **1231** to the node with number **2.**
We can write in the command line:

```
echo ″2″ > /proc/1231/goto
```

2) The destination node for the migrated process with *pid*=**1231** will be chosen by the load balancing algorithm:

```
echo ″-1″ > /proc/1231/goto
```

Script with MOSIX Application

```
echo /proc/self/where > /proc/self/selected

for i in 1 2 3 4 5 6 7;
do
  mosrun -j1-7 -l ./licz $i 7 > /mfs/selected/tmp/wynik_licz$i &
done;

wait
cat /mfs/selected/tmp/wynik_licz* | sort –n –r head –1> wynik.txt
rm /mfs/selected/tmp/wynik_licz*
```

OpenSSI Clustering System

OpenSSI (***Open S****ingle* ***S****ystem* ***I****mage*) – comprehensive clustering solution offering a full, highly available SSI environment for Linux. OpenSSI clusters can be built from standard servers. (http://www.openssi.org/)

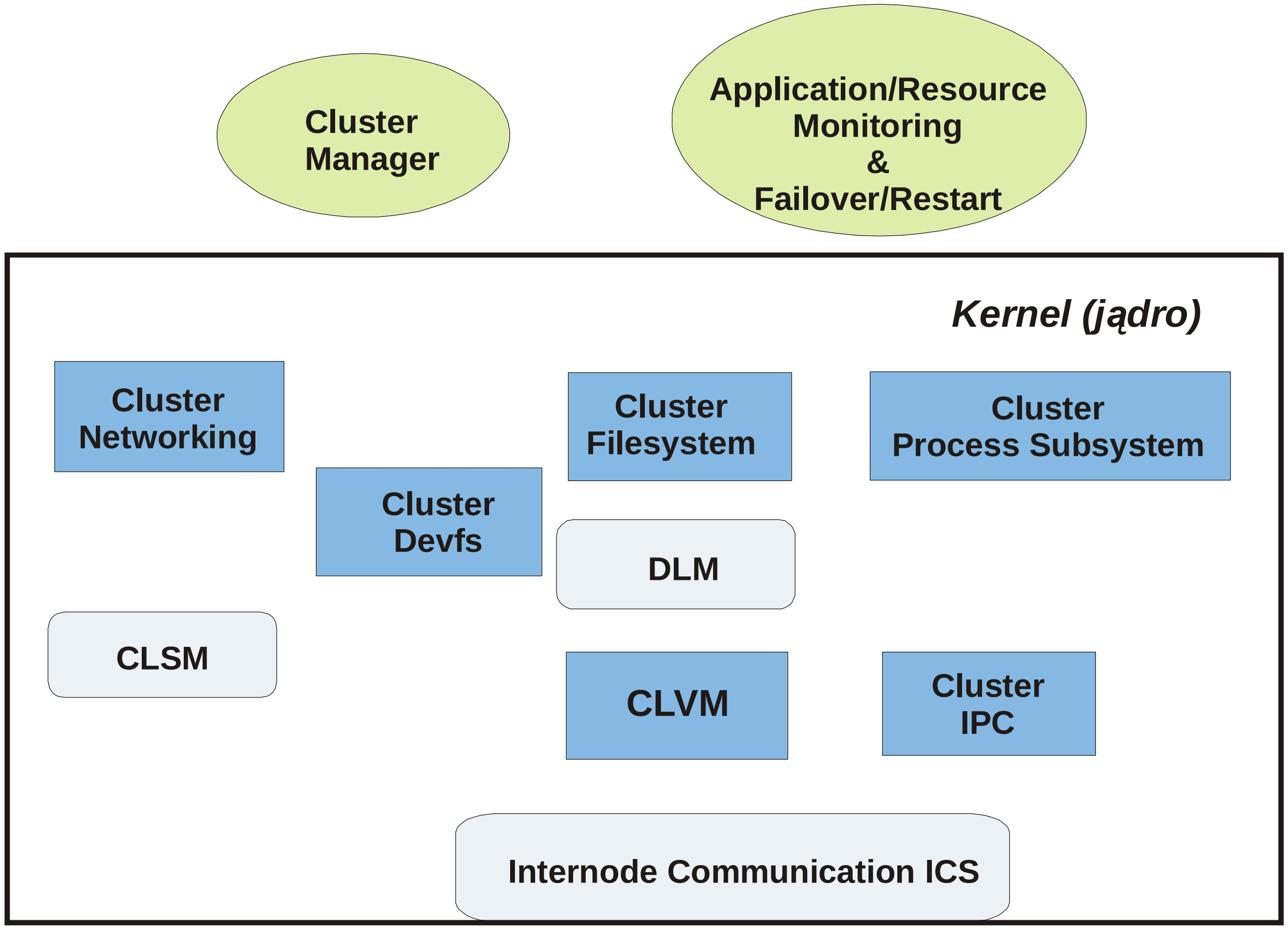
OpenSSI goals: available, scalable and manageable cluster.

Operating systems: all POSIX (Linux/BSD/UNIX-like OSes)

Technology includes: membership, single IP address to the external world and single init, cluster file system, distributed lock manager (runs on each node to enable various forms of cache coherency), single process space, single and shared IPC space, load balancing and leveling, process migration…

License: OpenSSI code released under the GNU ***G****eneral* ***P****ublic* ***L****icense* (GPL).

OpenSSI Architecture



Cluster Manager
Application/Resource Monitoring & Failover/Restart
Kernel (jądro)
Cluster Networking
Cluster Filesystem
Cluster Process Subsystem
Cluster Devfs
DLM
CLSM
CLVM
Cluster IPC
Internode Communication ICS

Networking Model

Consists of two parts:

- **CVIP (*C*luster *V*irtual *IP*), or cluster alias – address on an external network** (it makes the cluster look like a single, highly available machine).
- **Addresses for kernel-to-kernel communication:** (each node has one or more IP addresses that act in a per-node manner; that address can be used for MPI or other cross-node application communication).

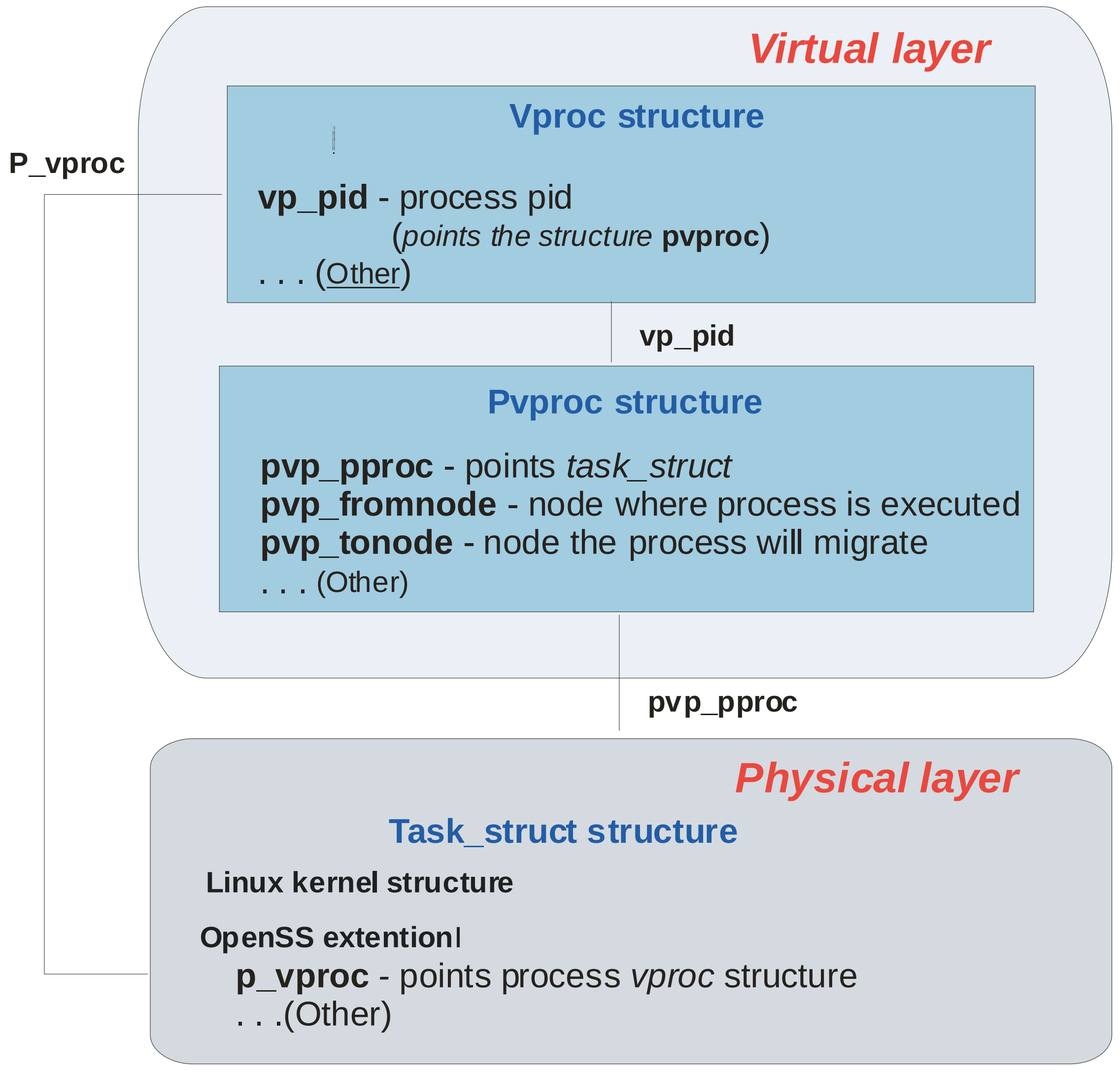
SSI model: uses LVS (***L****inux* ***V****irtual* ***S****erver*) project technology.

Process Management - Completely SSI

- Each process has cluster wide unique **pid**.
- Transparent access to all processes on all nodes:
  single unique list of processes,
  cluster wide **`/proc`** (**`ps`** shows all processes on all nodes).
- Cluster wide job control.
- Transparent migration of processes (during execution).
  Migration is via:
  - *servicing* **`/proc/<pid>/goto`** (*done transparently by kernel*)
  - *manual migration*
- Transparent debugging.

System **Vproc** (***Virtual Processes***) – Linux kernel extension.

Vproc Implementation



Process Migration Technology

.

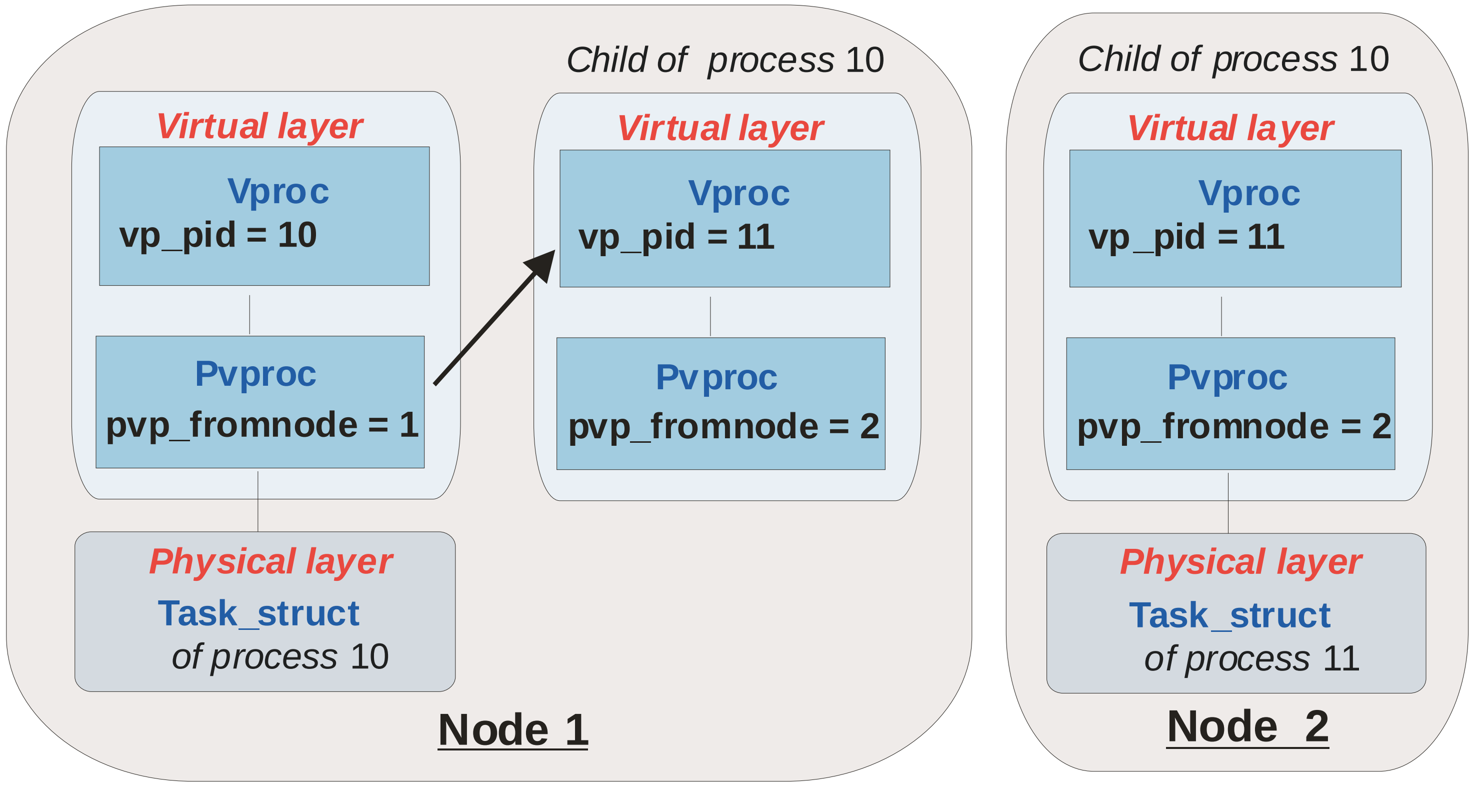
Migration technology is different than in MOSIX project.

- **Origin node** (creation node; node whose number is in the ***pid***) is responsible for knowing if the process exists and where it is executed).
- **Origin node** controls the migration process.
- Cluster wide unique processes ***pids*** **retain** if the process is moved to the other node.
- Information about changes in process allocation is provided to **origin** and **destination nodes**.
- **Whole process** and its context is moved to the destination node.
- The updated ***task_struct*** is copied to the destination node. The ***vproc*** and updated ***pvproc*** stay in the origin node (a field in the ***pvproc*** indicates the destination node).
- If the origin node goes away the ***surrogate origin node*** (whose identity is well known to all nodes) is assigned to the process.

OpenSSI is using load balancing algorithm from the MOSIX project.

Vproc: Parent/child Relationship

Example: Parent process with *pid* = 10 that was executed at the node 1 started child process with *pid* = 11. Process 11 was moved to the node 2.



System wide Device Naming and Access

- Each node creates a device space through **`devs`** and mount it in **`/cluster/nodenum#/dev.`**
- Global naming space.
- Each node sees it's devices in **`/dev`**.

File System Strategy

Support:

- parallel physical files systems (like **GFS**),
- layered **Cluster File Systems CFS** (which allows SSI cluster coherent access to non-parallel physical files systems like: JFS, XFS, ext3, etc.),
- parallel distributed (eg. **Lustre**).

User Interface - /proc

Directory	Files
cluster/node{n}	Node **n** control. Available files: **load** – load balancing of the node, **loadlevel** – load balancing algorithm switching for node **n** (**0** – No, **1** – Yes).
cluster/loadlevellist	List of processes that can migrate.
cluster/events	Information about nodes connection and disconnection.
PID	Information about process: **where** – node on which process is executed, **loadlevel** – load balancing status (**1** – Yes), **goto** – number of node to which process should be moved.

.

Vproc Implementation – API

1) Semantically identical to exec() and fork() but with node number argument.

```
rexec()
rfork()
```

2) Move process to the pointed node

```
migrate()
```

3) Way to ask on which node a process is executed

```
where_pid()
```

Migration Control

Automatic migration: file **proc/<pid>/loadlevel** :
„**1**” – allows for migration
„**0**” – blocks migration

<u>Example:</u> Different approaches to **manual migration**:

Allows for migration of process *pid*:

loadlevel –p <pid>

Allows for migration of process with the name *program*:

loadlevel <program>

Moves the process *pid* to the node with the number 2:

migrate <pid> 2

<u>inside the program</u> – function from the library ***libcluster***:

migrate(clusternode_t node)

System Kerrighed

Kerrighed (http://www.kerrighed.org/) provides the view of a single SMP machine on a top of cluster (SSI cluster).

Global management of cluster resources (unique *pid* for each process, global list of processes).

A configurable global process scheduler (dedicated scheduling policies can be easily written an plugged in the cluster) .

Migration mechanism based on several approaches: ghosting, containers, migrable streams and distributed.

Individual thread migration.

Kerrighed – system under development.

Comparison of Covered SSI Features in:
MOSIX, OpenSSI i Kerrighed

	MOSIX	OpenSSI	Kerrighed
Cluster wide unique PID space	No	Yes	Yes
Cluster wide process view (***global ps***)	No	Yes	Yes
Global memory and process load (***global top***)	No	No	Yes
Global device view (***global /dev***)	No	Yes	No
Processes migration	Yes	Yes	Yes
Migration of processes using System V memory segment	No	Yes	Yes
Thread application migration	No	Yes	Yes
Individual thread migration	No	No	Yes
Global process scheduler	Yes	Yes	Yes
Dynamic cluster reconfiguration (hot node addition and removal)	Yes	Yes	No
Tolerance to node failure	Yes	Yes	No