A Global Operating System for HPC Clusters

Emiliano Betti\textsuperscript{1}  Marco Cesati\textsuperscript{1}  Roberto Gioiosa\textsuperscript{2}  Francesco Piermaria\textsuperscript{1}

\textsuperscript{1}System Programming Research Group, University of Rome “Tor Vergata”
\textsuperscript{2}BlueGene Software Division, IBM TJ Watson Research Center

IEEE Cluster 2009
Many clusters... running Linux!

- Many modern supercomputers are built as *cluster* of thousands of independent nodes
- The OS commonly used in these supercomputers runs a version of the Linux kernel (more than 80% in the last top500)
- Each node runs an independent OS kernel, then:

  There is no temporal synchronization among the kernel’s nodes
Many modern supercomputers are built as *cluster* of thousands of independent nodes.

The OS commonly used in these supercomputers runs a version of the Linux kernel (more than 80% in the last top500).

Each node runs an independent OS kernel, then:

*There is no temporal synchronization among the kernel’s nodes.*
Do we need synchronization?

A strict temporal synchronization would allow or simplify activities, like:

- performance analysis,
- application debugging,
- data checkpointing.

Moreover, performance and scalability of an HPC application could be considerably improved just by synchronizing the system activities among the nodes in order to avoid the effects of operating system noise and noise resonance.
Do we need synchronization?

A strict temporal synchronization would allow or simplify activities, like:

- performance analysis,
- application debugging,
- data checkpointing. . .

Moreover, **performance** and **scalability** of an HPC application could be considerably **improved** just by synchronizing the system activities among the nodes in order to avoid the effects of **operating system noise** and **noise resonance**
In our paper we introduce

**CAOS, Cluster Advanced Operating System**

a prototype for a HPC Operating System based on Linux

**Main idea**

Synchronize the system activities of all cluster nodes
In the following slides:

- Motivations for our work, reviewing the concepts of OS noise and noise resonance
- Introduce CAOS:
  - general idea,
  - HPCSCHED,
  - NETTick
- Show the experiments’ results
HPC applications

Most of HPC applications are

- *Single Process-Multiple Data (SPMD)*,
- implemented using the *MPI protocol*

MPI applications can be characterized by a cyclic alternation of two phases.

- **Computing phase**: each process performs some computation on its local portion of data
- **Synchronization phase**: processes communicate by exchanging data among themselves

*Crucial for performance is ensuring that all the nodes start and terminate each phase at the same moment*
HPC applications

Most of HPC applications are

- **Single Process-Multiple Data (SPMD),**
- implemented using the **MPI protocol**

MPI applications can be characterized by a cyclic alternation of two phases.

- **Computing phase**: each process performs some computation on its local portion of data
- **Synchronization phase**: processes communicate by exchanging data among themselves

**Crucial for performance is ensuring that all the nodes start and terminate each phase at the same moment**
Operating System noise

Assuring phases’ synchronization is not easy mainly because of the so-called *Operating System noise*

In other words, OS activities like:

- timer interrupt,
- disk cache flush,
- and so on...

slow down the computing phases consuming CPU cycles

Noise resonance

Since **nodes’ kernels are not synchronized**

OS noise effects in a large scale cluster **may decrease** considerably **performance** and **scalability**
OS noise in a single node has been analyzed and measured in the past and is about $1 - 2\%$ on average.

Anyway if, on each computational phase, all the processes have to wait for one slow process, the whole parallel job is delayed.
Synchronization with CAOS

On large clusters the probability that during each computational phase one process is delayed due to the OS noise approximates 1.
Synchronization with CAOS

On large clusters the probability that during each computational phase one process is delayed due to the OS noise approximates 1

A possible solution is forcing all cluster’s nodes to perform system activities at the same time

This is one among CAOS’ main goals
Global Operating System services

Besides improved performance with a Global OS we could:

- simplify performance analysis
- realize data checkpointing for OS level fault tolerance
- improve application debugging

Example
Capability to stop the parallel application (at the same moment all the nodes) obtaining a coherent global snapshot
Besides improved performance with a **Global OS** we could:

- simplify performance analysis
- realize data checkpointing for OS level fault tolerance
- improve application debugging

**Example**

Capability to stop the parallel application (at the same moment all the nodes) obtaining a coherent global snapshot
CAOS is a global operating system for HPC cluster based on Linux

The core layer essentially consists of two main agents:

- **Global conductor**
- **Local instrumentalist**

In our prototype we realized two main components:

- **HPCSCHED**: a new scheduler design for HPC cluster
- **NETTICK**: the global heartbeat mechanism of CAOS
CAOS is a global operating system for HPC cluster based on Linux

The core layer essentially consists of two main agents:

- **Global conductor**
- **Local instrumentalist**

In our prototype we realized two main components:

- **HPCSched**: a new scheduler design for HPC cluster
- **NETTick**: the global heartbeat mechanism of CAOS
The *Global conductor (GC)* runs on a selected **master node** and is in charge to schedule cluster activities. It

- provides a common time source (**heartbeat**) to the cluster nodes (**NETTick**, already implemented)
- notifies global and local system activities

**Example**

The GC may command the nodes to check for:

- decayed software timers
- reclaim pages from the disk caches

The Global conductor could be implemented as a **user mode program running on the Master node**, i.e., modifying an available job scheduler
The Global conductor (GC) runs on a selected master node and is in charge to schedule cluster activities. It
- provides a common time source (heartbeat) to the cluster nodes (NETTICK, already implemented)
- notifies global and local system activities

Example

The GC may command the nodes to check for:
- decayed software timers
- reclaim pages from the disk caches

The Global conductor could be implemented as a user mode program running on the Master node, i.e., modifying an available job scheduler.
The Global conductor (GC) runs on a selected master node and is in charge to schedule cluster activities. It:

- provides a common time source (heartbeat) to the cluster nodes (NETTick, already implemented)
- notifies global and local system activities

**Example**

The GC may command the nodes to check for:

- decayed software timers
- reclaim pages from the disk caches

The Global conductor could be implemented as a user mode program running on the Master node, i.e., modifying an available job scheduler.
Local instrumentalist

The *Local instrumentalist (LI)* runs on each cluster node and:

- performs the operations scheduled by the global conductor
- when needed, could notify the GC when the operations have been completed

**Main goal:** not interfere with the computing phase of the HPC application

**Main component:** a kernel’s task scheduler (our prototype now uses `HPCSCHED`)
HPCSched is a new scheduler designed for HPC Cluster and presented at ACM/IEEE Supercomputing, 2008

Main features:

- balancing MPI applications
- minimizing OS noise caused by other user and kernel threads

It uses:

- hardware resource allocation mechanism provided by an underneath processor (like IBM POWER6),
- heuristic functions to select the amount of hardware resources to assign to the task.
HPC_SCHED is a new scheduler designed for HPC Cluster and presented at ACM/IEEE Supercomputing, 2008

Main features:

- balancing MPI applications
- minimizing OS noise caused by other user and kernel threads

It uses:

- hardware resource allocation mechanism provided by an underneath processor (like IBM POWER6),
- heuristic functions to select the amount of hardware resources to assign to the task.
HPCsched and OS noise

HPCsched introduces a new scheduling class and a new policy to select and balance the tasks.

Many kernel activities should not be executed if and HPC application is ready to run.

Main result

HPCsched is able to execute the system activities during the communication phase of the MPI application.
**NetTick** is our prototype implementation of the heartbeat mechanism of CAOS

It’s implemented as a patch for the Linux kernel v. 2.6.24

### Main idea

- Substitute the nodes’ local tick device with a **common network tick device**

- All the nodes execute the **time interrupt** at the **same moment** and the **same number of time**, then they have the same perception of the **flow of time**
NetTick is our prototype implementation of the heartbeat mechanism of CAOS.

It’s implemented as a patch for the Linux kernel v. 2.6.24

**Main idea**

- Substitute the nodes’ local tick device with a **common network tick device**

- All the nodes execute the **time interrupt** at the **same moment** and the **same number of time**, then

  *they have the same perception of the flow of time*
Local tick device vs common tick device

Why do we need a common tick device?
First of all, Network Time Protocol (NTP) is NOT a solution!
It synchronizes only the Wall Clock Time...

Instead we need to synchronize the system tick, and:

- Normally each node uses his own timer device to measure time and raise a periodic interrupt (i.e. Local APIC for Intel/AMD, Decrementer for PowerPC)

- Even timer devices of the same type oscillate at slightly different frequencies

- Tick period is slightly different from node to node

Machines have different perception of the flow of time
Local tick device vs common tick device

Why do we need a common tick device?

First of all, Network Time Protocol (NTP) is NOT a solution!
It synchronizes only the Wall Clock Time...

Instead we need to synchronize the system tick, and:

- Normally each node uses his own timer device to measure time and raise a periodic interrupt (i.e. Local APIC for Intel/AMD, Decrementer for PowerPC)
- Even timer devices of the same type oscillate at slightly different frequencies
- Tick period is slightly different from node to node

Machines have different perception of the flow of time
Local tick device vs common tick device

Why do we need a common tick device?

First of all, Network Time Protocol (NTP) is NOT a solution!
It synchronizes only the Wall Clock Time...

Instead we need to synchronize the system tick, and:

- Normally each node uses his own timer device to measure time and raise a periodic interrupt (i.e. Local APIC for Intel/AMD, Decrementer for PowerPC)

- Even timer devices of the same type oscillate at slightly different frequencies

- Tick period is slightly different from node to node

Machines have different perception of the flow of time
Why do we need a common tick device?
First of all, Network Time Protocol (NTP) is NOT a solution!
It synchronizes only the \textit{Wall Clock Time}...

Instead we need to synchronize the \textit{system tick}, and:

- Normally each node uses his \textbf{own timer device} to measure time and raise a periodic interrupt (i.e. Local APIC for Intel/AMD, Decretementer for PowerPC)
- Even timer devices of the same type oscillate at slightly \textbf{different frequencies}
- \textbf{Tick period} is slightly \textbf{different} from node to node

Machines have different perception of the flow of time
Patch of Linux v. 2.6.24:

- **No changes in drivers**, only the core has been patched
- A **very small part** of the patch, only needed to support SMP system, is **architecture-dependent**. Actually we support *Intel IA32, Intel64, PowerPC64*
- **sysfs** interface to activate/deactivate **NETTick**
- Based on **Ethernet** communication channel
  - Our test bed used Gigabit Ethernet
  - According to Top500 list more than 56% of top ranking cluster are based on Gigabit Ethernet
  - If **NETTick** performs well on Gigabit Ethernet, it can presumably work with any other network architecture having a similar or higher performance
Patch of Linux v. 2.6.24:

- **No changes in drivers**, only the core has been patched.
- A **very small part** of the patch, only needed to support SMP system, is **architecture-dependent**. Actually we support *Intel IA32, Intel64, PowerPC64*.
- **sysfs** interface to activate/deactivate NETTICK.
- Based on **Ethernet** communication channel:
  - Our test bed used Gigabit Ethernet.
  - According to Top500 list more then 56% of top ranking cluster are based on Gigabit Ethernet.
  - If NETTICK performs well on Gigabit Ethernet, it can presumably work with any other network architecture having a similar or higher performance.
**NETTICK: global heartbeat**

A Master node send periodic frame

*Capability to dynamically change node’s tick frequency*
**NETTick: timer interrupt emulation**

- **New tick frame**
  - Driver NIC
    - hard IRQ context
  - nettick_rcv()
    - soft IRQ context
    - send_tick_ipi()
  - (smp_)nettick_timer_interrupt()
    - hard IRQ context

**Timekeeping activities**

**Introduction**

**Motivation**

- OS noise
- Global OS services

**CAOS**

- HPC Schedule
- NETTick

**Experiments**

**Conclusion and future work**
Experiments

We ran 3 sets of different experiments:

- OS noise reduction in a single node using FTQ (*Fixed Time Quanta*):
  - NETTICK (timer interrupt noise)
  - HPCSCHED (scheduling noise)

- testing NETTICK scaling in a cluster (using NAS Parallel Benchmark): we can show that deferring system activities does not impact on the performance of parallel applications
Experiments’ test bed

24-nodes **Apple Xserve** cluster, generously offered by **Italian Defence’s General Staff**

- Each node is equipped of 2 *dual core* Intel Xeon 5150 2.66GHz processors (overall *96 cores*), 4GByte of RAM and 2 Gigabit Ethernet Network Interface Cards
- One of the 24 nodes was employed as *master node*
Noise measured with FTQ

We executed *Fixed Time Quanta (FTQ)* on a single node to analyze the OS noise introduced by the timer interrupt.

Test conditions, needed to isolate the timer interrupt:

- All the CPUs idle (only FTQ running)
- Reduced number of kernel threads and user daemons

FTQ measures the amount of work done in a fixed time quantum (1 ms) in terms of basic operations. The difference between the maximum number of basic operations ($N_{max}$) and the number of basic operations in a sample ($N_i$) can be considered OS noise.

$$\text{OS noise} = N_{\text{max}} - N_i$$
Noise measured with FTQ

We executed *Fixed Time Quanta (FTQ)* on a single node to analyze the OS noise introduced by the timer interrupt.

Test conditions, needed to isolate the timer interrupt:
- All the CPUs idle (only FTQ running)
- Reduced number of kernel threads and user daemons

FTQ measures the amount of work done in a fixed time quantum (1 ms) in terms of *basic operations*. The difference between the maximum number of basic operations ($N_{max}$) and the number of basic operations in a sample ($N_i$) can be considered OS noise:

$$OS \text{ noise} = N_{max} - N_i$$
Timer interrupt noise measured with FTQ

Standard system

NETTick
Scheduling noise measured with FTQ

In this test we introduced a periodic user daemon for cluster activities.

Standard system

HPCSCHED and NETTick
A Global OS for HPC Clusters

E. Betti, M. Cesati, R. Gioiosa, F. Piermaria

Introduction

Motivation

OS noise

Global OS services

CAOS

HPC Sched

NETTick

Experiments

Conclusion and future work

A cluster using NETTick

Curves of execution times of NAS EP varying the number of cores, comparing the system tick (100 MHz) with NETTick (10MHz, 50MHz, 100MHz)

We also ran NAS BT obtaining the same evidence: no delay introduced by NETTick
Conclusion and future work

We have proposed CAOS, an extension to the Linux kernel for HPC applications in cluster-based supercomputers.

Main goals of CAOS:
- schedule OS activities in order to reduce the OS noise
- simplify activities like debugging or checkpointing

In CAOS’ prototype, we developed two major components:
- \texttt{HPCSCHED}: a local task scheduler for HPC application
- \texttt{NETTICK}: a global heartbeat mechanism

We proved that:
- both components reduce the OS noise on a single node
- CAOS does not introduce overhead that impacts on HPC applications performances
A Global Operating System for HPC clusters

System Programming Research Group
University of Rome “Tor Vergata”
http://www.sprg.uniroma2.it

BlueGene Software Division
IBM TJ Watson Research center

{betti, cesati}@sprg.uniroma2.it, rgioios@us.ibm.com,
francesco.piermaria@gmail.com

We gratefully acknowledge support of this project by the
Italian Defence’s General Staff