Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 31 (2005) 813–838
PARALLEL COMPUTING
www.elsevier.com/locate/parco

The I-Cluster Cloud: distributed management of idle resources for intense computing

Bruno Richard [a], Nicolas Maillard [b,*], César A.F. De Rose [c], Reynaldo Novaes [d]

[a] *HP Labs Grenoble, 5 Avenue Chanas, 38053 Grenoble Cedex 9, France*
[b] *Federal University of Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500, 91501-970 Porto Alegre, Brazil*
[c] *Pontifical Catholic University of Rio Grande do Sul (PUCRS), Faculty of Informatics, Av. Ipiranga, 6681, 90619-900 Porto Alegre, Brazil*
[d] *HP-Brazil, R&D, Av. Ipiranga 6681, Building 91A, 91501-970 Porto Alegre, Brazil*

Received 16 March 2004; accepted 2 June 2005
Available online 28 July 2005

Abstract

I-Cluster is an HP Laboratories Grenoble initiative, in collaboration with the ID-IMAG laboratory of INRIA Rhône-Alpes, HP Brazil and the Catholic University of Rio Grande do Sul (PUCRS). I-Cluster consists of a framework of tools that transparently takes advantage of unused network resources and federates them, in order to crystallize into specific virtual functions such as supercomputing. By doing this, I-Cluster enables automatic real-time analysis of the availability and workload of machines on an intranet. When the instantiation of a supercomputing function is requested by a user, I-Cluster determines the most appropriate set of machines for carrying out this function, allocates the machines into a virtual cluster and proceeds with the execution of the function. In order to address security issues, I-Cluster uses an "execution sandbox" on each machine of the intranet, which is transparent to the user and enables the use of local computing resources at idle periods, while securely protecting the local user data and jobs.

* Corresponding author.
E-mail address: nmaillard@inf.ufrgs.br (N. Maillard).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2005.06.001

In this paper we introduce the I-Cluster initiative and its overall architecture and present the main issues addressed in the conception of the I-Cluster framework, such as solving peer-to-peer computing security issues using OS sandboxing, self-organization and resilience to unanticipated disconnections of large and heterogeneous community of computers, as well as automatic resource collection. To validate the I-Cluster framework we both present experimental results obtained with a small scale prototype and simulated results for environments with a larger number of resources.

Keywords: Distributed computing; Idle cycles; Gossiping; Sandbox

1. Introduction

This article presents I-Cluster, a joint initiative between Hewlett-Packard Laboratories Grenoble and the INRIA (ID-IMAG laboratory, France), in collaboration with Hewlett-Packard Brazil and the Research Center in High Performance Computing (CPAD – PUCRS/HP) of the Catholic University of Rio Grande do Sul (PUCRS, Brazil). It aims at providing an environment, the Cloud, that federates computing resources of all sorts and aggregate idle machines into high-throughput virtual clusters. In this paper, such computing resources will be called "nodes" or "machines" without distinction.

The idea to use idle cycles of network connected computers dates back to the Condor project in the 1980s. The emergence of the world-wide web and the new possibilities offered by collaborative computing projects such as SETI@home[1] have incentivated new research on the theme, like the Holo Project [1], that offers support for parallel autonomic computing. The main concern in current works are:

- the new scale that distributed resources may reach, since a big company's intranet, for instance, may host up tens of thousands nodes distributed worldwide;
- security issues, since using idle cycles from such a wide pool of resources only can be considered if the normal users do not suffer any damage with their data.

The I-Cluster, which we present here, is a realistic attempt to tackle these issues. We show here how it has been conceived in the ID-IMAG laboratory and Grenoble's HP labs, and validated through a real implementation in Brazil. Simulations have also been used in both teams to evaluate I-Cluster's scalability on tens of thousands nodes.

This article is structured as follows: Section 2 presents some environments for distributed computing, such as Condor, one of the milestones in the use of idle cycles, or MOSIX, which introduced the gossiping algorithms that I-Cluster uses to propagate information in the Cloud. In Section 3 we introduce the I-Cluster environment, its key principles and the different modules it is made of. In order to evaluate I-Cluster we

[1] http://setiathome.ssl.berkeley.edu/

developed a prototype at CPAD – PUCRS/HP, which is the focus of Section 4. Due to the limited number of nodes of our prototype (25), we have been using simulation in order to validate I-Cluster's gossiping algorithm on a realistic number of nodes (Section 5). Finally we conclude in Section 6.

2. Related work

Many projects have tackled the issue of federating idle cycles of large scale distributed resources for high-performance computing. In this section we consider the research made on Condor, a problem-solving environment called DIET, the global computing project XtremWeb and the well-known MOSIX system of which I-Cluster is very much inspired.

2.1. Condor

Condor [2] is an on-going 15 years old project that aims at using idle cycles. The authors have tackled the problem of harnessing nodes from different pools to form an unique *flock* of resources on which jobs may be scheduled in a transparent way [3]. Since then Condor has embraced the Globus Computing effort [4].

Condor uses a system of remote procedure calls associated to checkpointing in order to execute (independent) tasks on the computing nodes. Currently Condor also provides a master/slave scheme to parallelize large applications. Nodes are organized in *pools*, one of them being the *central manager* (CM), which administers the other nodes and verifies their availability. Each node runs two daemons, one called "schedd" that queues the jobs that are assigned to this node, and the "started" which controls the ping–pongs with the CM and the starting and halting of the jobs. The CM also has a matching system to determine, based on the nodes' characteristics, on which node it has to run given jobs.

To handle inter-pools job exchanges, each pool has one or more nodes acting as Gateways. Gateways maintain information about each other, and regularly check availability of the nodes in their pool. The CM itself is informed about the available nodes in the other pools and takes them into account in order to schedule its own jobs. Inserting or removing a node simply requires to inform the pool's CM. It will then relay the information to other collaborating nodes.

Allocating a node to a job requires to match the job's requirements and possibilities. The ClassAd mechanism is used to describe both, and a distributed matching algorithm compares each one of the two descriptions. When the matcher finds a right needs/offer association, it sends the address of the potential correspondent back to the node and to the requester. It is then up to these ones to connect in a "peer-to-peer" way, without any more interaction with the Matcher. A processor allocation therefore costs 2 messages (the Matcher sends a message to both the requester and the node), and 1 more to transfer the job from requester to the executing node.

Our proposed platform, I-Cluster, differentiates from Condor in two ways: first we aim at treating fine-grained applications, not necessarily embarrassingly parallel.

This type of application is not currently well suited to the Condor system, which is based on preemption and migration of processes, although recent efforts to treat MPI applications may be a move in this direction. Second, I-Cluster includes a sandbox mechanism that tackles security issues not taken into consideration by Condor.

2.2. Diet

DIET [5] is an environment developed within the ReMap project, in the LIP laboratories (Lyon, France). Its goal is to provide a Grid-like environment for distributed computing (Network Enabled Server Applications) that could deal with:

- managing a pool of computing resources;
- providing a framework to describe a computing problem;
- looking for the data location;
- evaluating the execution time required on the various possible nodes of the environment;
- choosing the right nodes to run a computation that asks for resources (scheduling issue).

An environment close to DIET is NETSOLVE [6], since they both aim more at providing a Problem Solving Environment (PSE) rather than computational resources themselves.

DIET is made of the following components:

- The Master Agents (MA), that are in charge of receiving an application's requests and to chose the nodes on which it shall be executed. They are therefore responsible for the scheduling decisions.
- The Local Agents (LA) are intermediaries between the MAs and the nodes. They collect information about the available resources on each set of nodes.

Each computing node hosts a Computation Resources Daemon that informs about the computing capacity of the node (which can in turn host many processing units). It also hosts a Server Daemon (SeD) that manages the available data and the load of the processing units.

Both at MAs and at LAs levels, different distributed architecture may be faced by DIET. [5] does not give any detail on a specific architecture for the MAs neither for the LAs. The given example presents results for one MA and a tree of LAs.

DIET works with the help of two intermediary independent layers: the MAs use SLiM to describe and interpret the problem to be solved. SLiM classifies the problems and solutions within an LDAP compliant structure (such as Globus for instance). FAST, on the other hand, handles the performances forecasting, based on the information given by the SeD, and is used by the LAs which will transmit these forecasts to the MAs in order to decide a scheduling.

The construction of the computational Grid is made in a tree-like way, each LA being inserted to its "father". The hierarchy has to be described to the MA by the administrators and may, for instance, reflect some administrative or geographic criteria. In the case presented in [5] there is only one MA, which is therefore the root of the tree.[2] The article envisions the use of a Broadcast between the MAs regarding request processing, but does not explain how the architecture would be initialized.

The main differences to the I-Cluster are the focus on providing a Problem Solving Environment (PSE) rather than computational resources and the hierarchical way available resources are organized in the Grid (tree-like). This implies a significant management overhead specially in dynamic environments, with lots of resources entering and leaving the Grid. Security issues are also not taken into consideration in Diet, playing a major role in the I-Cluster.

2.3. XtremWeb

XtremWeb [7] is an experimental Global Computing platform. XtremWeb shares its architecture with SETI@home [8], but is not limited in the number of available applications (from genome exploration to ray tracing).

Fig. 1 shows a global view of the XtremWeb model. An XtremWeb System is composed of a network of *Servers* and *Workers*. Each Server owns some Workers. Upon user request, a Worker can submit an *Application* to the system by sending it (and the files needed for it) to the Server. Then the Server splits the Application into *tasks*. When a Worker owned by this server becomes idle (this is noticed by the use of the screen-saver for example), it asks the Server for a task to compute. When the computation is finished, the Server pushes the result in a Result Server. When a new Worker connects to a Server, it is included in a list of owned Workers. A disconnected Worker is suppressed from the Server's list of owned Workers and the aborted task is reassigned.

Security issues are also not taken into consideration in XtremWeb and SETI@home. The use of cycle steeling through screen-savers that are running in user space represent a major threat to local security. This was one of the main motivations to improve security in the I-Cluster through a sandbox.

2.4. MOSIX

MOSIX is a Multicomputer Operating System for Unix [9]. It is a several-decades-old project from Jerusalem's Hebraic University. MOSIX's goal is to provide a set of adaptive mechanisms to share resources in order to handle a processors farm as a single system, optimizing the performance through efficient use of the network. The processors farm is composed of standard machines that may be heterogeneous, linked by a standard LAN.

[2] Since there is only one MA, the scheduling decision is centralized, such as in Netsolve.

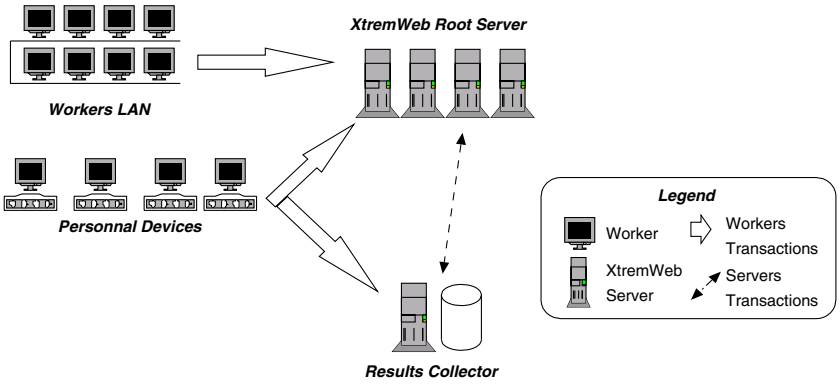

Fig. 1. XtremWeb communication structure.

MOSIX is based on a modified Unix kernel, so that it can migrate the processes between nodes in the processing farm. MOSIX also makes use of algorithms to distribute the nodes load information, to detect the load differences between nodes and to migrate the tasks from the most loaded nodes to the least loaded ones (PPM – Preventive Process Migration). The system is symmetric and distributed, without any centralized control.

Each process owns an initial node (UHN – Unique Home Node) which is generally the one on which the user is logged. The process's body contains its user context and some kernel level information which is independent from the site where it is run. The process's deputy owns the information depending from the site and resides on the UHN. A communication link is established between the deputy and the body to let the process access its local environment through the deputy and also in order to let other processes communicate with it. The migrated processes use local resources as much as possible, but resources such as network connections or open files have to stay on the UHN and will be handled by the deputy.

The load information is distributed using a gossiping algorithm [9]. Each node updates (typically every 2 s) a vector of information about the load of each of the other nodes in the processing farm. At regular time-steps (typically every 10 ms), it will send this vector to a randomly chosen other node. Upon receiving such a vector, its contents are combined with the local vector so as to keep only the most recent information. If the load between two machines is very different, a migration operation is done. Notice that the definition in MOSIX for a node's load is a direct function of the quantity of available memory, and not of the CPU load which is highly variable and hard to characterize. This avoids managing frequent variations of the load that would imply an important overload in the gossiping algorithm.

The diffusion by the gossiping mechanism has been shown to be quick. The MOSIX team proved [9] that the mean time of half-diffusion (mean time for an information

to go from one server to half the nodes in the farm) is of the order of $\log(n)$, n being the number of nodes.

We have been working on ways to improve the algorithm scalability (with a limited convergence speed however), and found that it was possible to store only a limited number of peers in each machine's local database in order to preserve scalability. Measuring the global system constituted by the Cloud starting from the simple description of the gossiping algorithm of a single machine could not be done on a real set of 10,000 machines, so we decided to build a simulation of the system to validate its properties.

We consider a simulation with n nodes, all fully functional. The time is discretized in synchronous steps. At step 0, one of the nodes updates its information. No other information is supposed to change during the duration of the simulation. At each step each of the n machines will connect to another one, randomically chosen between the $n - 1$ other ones, and send its information (if it already owns it) to the chosen node. We study the number of nodes that received the information as a function of the time-steps. Fig. 2 presents the results obtained for $N = 100{,}000$ nodes. The simulation was run 20 times and the mean, minimum and maximum values are drawn on the graph. It can be seen that the diffusion of the information is very fast, lasting only 19 time-steps in the worst case. In order to reach half the number of nodes, the diffusion takes 12 time-steps.

This result lets us anticipate valuable capabilities from the MOSIX derivated algorithms, which we expect to keep for the I-Cluster Cloud: self-organization, automatic information collection about the nodes and scalability.

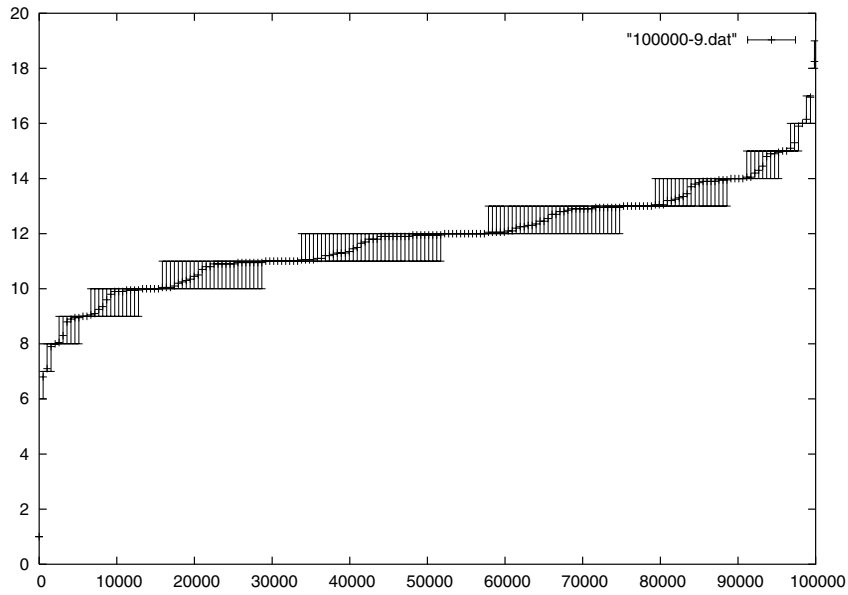

Fig. 2. Mean number of neighbors reached vs. timesteps.

3. The I-Cluster Cloud

3.1. Motivation

I-Cluster aims at organizing computing resources within a "Cloud" of processing units at the user's disposal. With this aim, it uses a MOSIX-like gossiping algorithm to propagate information about the nodes, in order to harness idle-cycles in the way Condor does. The integration of a gossiping algorithm in a scalable way and respecting the user's concern about security was a challenging task.

3.2. Services

Different types of problems must be dealt with in order to organize the I-Cluster Cloud:

- recognizing the nodes' capabilities (CPU power, memory, etc.);
- propagating information within the Cloud;
- offering a high security level to users (both those who own the resources of the Cloud and those who want to use them).

Node capabilities. Each machine belonging to the Cloud is provided with a sensor whose function is to instrument the local resources such as number, type and frequency of the processors, memory and disk size, and network connectivity. The node's IP configuration is automatically investigated, namely its current IP address, sub-network and default gateway. In case of multiple possible network connections, such as multiple Ethernet connectivity or concurrent Myrinet and Ethernet, each connection is instrumented. The instrumentation occurs after each node's state modification: upon boot, switch off, hibernation or even when network connectivity changes.

Propagating information. The Cloud has to disseminate information about the nodes it handles: when they are entering or leaving the system, nodes will be noticing other nodes to which they are connected. Information about nodes activity also has be propagated, in order to allocate new jobs to idle nodes. The algorithms to propagate the information have to be scalable in order to take into account hundreds of thousands computing nodes.

High-security level. The I-Cluster Cloud aims at harnessing nodes that may enter and leave the structure dynamically. Thus it provides a natural way to use idle cycles from machines loosely connected. In order to deal with implied security issues, we have chosen (see [10]) to use a sandboxing mechanism: the nodes can operate in two modes, the "user mode", in which the "normal" user (for instance, a student in an university) works on the machine; and the "cluster mode", in which the node enters when idle, to join the I-Cluster Cloud. Entering cluster mode, the node's OS shuts down and another OS image is booted, which has limited access to the hard disk i.e., only to the one partition where it has been installed (see Section 3.3.5). Thus, a physical sandbox prevents from any malicious code running and damaging

the node and the data it has in user-mode through isolation of user mode and cluster mode data and applications.

3.3. Architecture

Fig. 3 presents the I-Cluster Cloud architecture. From the entry point (“control point”) side, I-Cluster has to deal with job submission, to find a match between the job and the resources and to schedule the tasks on the nodes. On the resource side, each node has a sandbox and the resource sniffer that discovers new nodes. Between the two sides, the Cloud deals with resources and topology management.

In this section we detail each one of these services.

3.3.1. Job submission

Once the user needs a given computing service to be executed, it will use the job submission engine to provide the system with the detail of the job's requirements: Number of machines to be reserved, individual characteristics (power, memory size, capabilities), topology of the machines, command to be run, entry parameters, location for the output data. This detailed data will be used by the Match Finder described hereafter to find the proper set of machines that will satisfy the requirements for the requested service.

3.3.2. Match finder

Once each node is able to get its own view of the Cloud, it then knows a list of available machines of the network that it can use for submitting jobs. The user selects the computing service to invoke, and sets some additional information such as the entry parameters for the job, data file locations and the location where the output data should be exported (NFS, FTP, X display). This information is handled to a component called the Match Finder, which will be responsible for finding a group of nodes which has the best capabilities for the execution of the job. The I-Cluster Framework makes use of the available network resources and topology. This makes it possible for a given service to require a set of resources with critical network requirements, for instance machines that share the same subnet or interconnect. It is then possible to execute a very finely grained supercomputing job on the allocated resources.

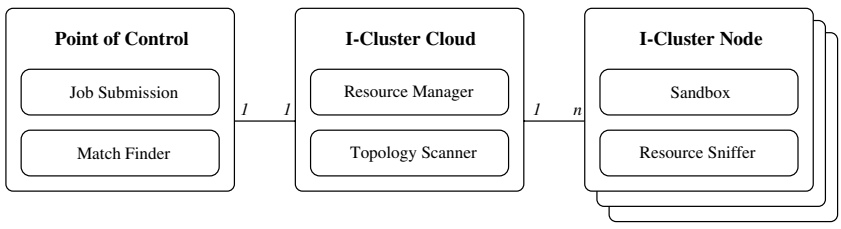

Fig. 3. I-Cluster architecture.

Each I-Cluster computing service is mapped to a description of how the service can be composed (instantiated) onto a cluster. The service composition takes into account the complexity of the computational job, expressed in computational power required by the job, number of machines, type and speed of processor required, minimal memory size, bandwidth and latency per node, as well as the required interconnect capacity between allocated nodes expressed in terms of network topology, maximum network latency between nodes and network bisection bandwidth.

The Match Finder will check the local database for a set of computers that match the service requirements, and attempt to allocate them to the job. This is done using a 2-phase commit algorithm, which first attempts the allocation of each required computer, and commits each of them if all the computers are available for the computation. This very simplistic allocation scheme may be very inefficient for a common cluster configuration, but in our case we assume that lots of idle machines will be available at the time of service invocation, hence the chances that two different attempts occur at the same time on a single machine are low. A double commit sequence is used to prevent problems when this occurs. I-Cluster shares the same allocation paradigm as MPI: The machines allocated to a given service are fully allocated for the duration of the service, and the number of machines cannot change during the execution.

3.3.3. Resource management

The shared information about each machine is characterized by a triplet (t, *id*, s), where t is a timestamp (last update time from a logical Lamport clock [11]), *id* is an unique identifier (generated by a random function) of the machine, and s is the state of the machine (online, off-line, at disposal, running, hibernating). The triplet is updated by the machine itself, but may also be modified by other nodes of the Cloud, for instance if they reach a consensus that the machine is not responding anymore to a ping and must be therefore considered as off-line.

When a node 'A' wants to enter the Cloud, it has to know in advance the *id* of at least one peer 'B' already connected to the Cloud (this is a pre-requisite of the algorithm). 'A' will send a message to 'B', that will answer sending the information that it has about neighbors in the Cloud. Thus 'A' will get some references about some other nodes of the Cloud and start interacting with them on its own.

The handling of the disconnection is made by consensus, following a 3 parts protocol:

- when a node A detects a timeout in a tentative to connect to one of his peers B, it adds it to a ''suspects list'', along with the timestamp at which the failure was detected and with its own identification id_A.
- If another node C is later unsuccessful in trying to connect to B, looking at the suspects list it will find out that B has already been suspected of being disconnected by A.
- C will then change B's status to *disconnected* and B will be removed from the suspects list.

The global set of triplets and the suspects list are stored in a distributed database, as in MOSIX. Each node of the Cloud has its own partial view of the global database, which is only a part of the whole set of nodes. The fraction of the total Cloud that it owns depends directly on its capacities, such as for instance a PDA may store a few hundreds triplets whereas a standard desktop PC would store a few thousands ones. The update of the (partial) databases is done through gossiping: at regular time-steps the gossiping agent of one node will choose a random neighbor in its local database, connect to it and they will exchange the content of their respective databases. If the information they exchange conflicts about some nodes, the t timestamp allows it to identify the most recent one.

In order to bound the information stored on a node, a filter mechanism has to be used at the time when it receives the information of one of his peers. The filter eliminates nodes from the database based on the information it has on their "interest": power, availability, network proximity. Nodes less interesting are not memorized. Using this scheme, one could construct a highly clustered graph of nodes, yet the mean-length path could be high. In order to lower it, a small fraction (typically 5%) of the least interesting nodes are kept in the database, thus favorizing the creation of a "Small-World" graph [12].

3.3.4. Topology scanner

Our system needs to be able to efficiently solve fine-grained problems. As we will allocate idle machines from the network to such distributed computations, we have to ensure that an allocated set of machines is as homogeneous as possible—in terms of CPU speed for instance—and that network latency is minimal. These conditions are basic requirements for efficient fine-grained computation. Concerning the network latency, it is not obvious how a given set of machines should be allocated so that the inter-node latency is as small as possible. This requires some knowledge of the network characteristics, and more precisely of its topology. However, full knowledge of the topology of an intranet is something which is not convenient to gather nor to maintain. For this reason we have been limiting the problem to the fact that a set of machines allocated to a single job should always be located on a single subnet. This ensures that inter-node latency on such an allocated set of machines will always be limited to a single cross-switch latency. Experience [13] has shown that this condition raised very acceptable results in practice, as we have been able to reach TOP500 performance levels using standard machines interconnected through Ethernet 100.

3.3.5. The I-Cluster sandbox

The term sandbox is used in this paper to characterize a safe and isolated area for the execution of untrusted code on a machine, allowing access to certain system features (network access, for example), while protecting the target machine and the data stored on it against security attacks from malicious or erroneous codes loaded by the sandbox [10,14,15]. Hawblitzel et al. [16] suggested classification of protection mechanisms in classes:

- The first class includes mechanisms based on physical virtual memory, which enables isolation of execution codes using hardware. Our aim was to offer an architecture based on a standard PC network, so hardware modifications were not feasible for I-Cluster.
- The second class of solutions is for *capacity systems*, used in microkernel operating systems such as Amoeba [17] and Chorus [18]. These systems, based on examination of data types on execution, require instrumentation of source codes which leaves the necessary traces for the mechanism controlling the interest lists. The Java Runtime Environment described by Arnold et al. [19] has a sandbox that is automatically activated when applets belonging to this class are executed. Internet C++ [20] is another example enabling execution of POSIX code associated with a virtual machine giving access to a network interface (Berkeley sockets model) and a graphical interface (OpenGL). A capacity systems class sandbox could have enabled us to natively run Windows applications, and therefore actively share the resources of each machine by running I-Cluster applications in the background with software protection from erroneous code. However, this class of mechanisms requires recompilation of existing applications, so it has been dropped.
- The last class contains SFI, Software-based Fault Isolation mechanisms. I-Cluster sandbox is in this class, and shares the resources of each machine on a time basis. At a given moment, a machine will therefore be fully available to its user, or fully allocated to I-Cluster tasks. We could have used less strict resource sharing, such as that carried out by the VMWare[3] virtual machine, which enables transit between two operating systems such as Windows and Linux using a software emulation layer. The complexity of such an environment makes it very difficult to stabilize, so we preferred working with simple mechanisms.

3.3.6. Resource Sniffer

In order to instrument hardware capabilities of each node we have built a software sensor that instruments a given machine. This sensor digs into the machine's BIOS and extract very precise information about the processor(s) number, family, type and speed, the level 2 and level 3 memory cache size, mass storage, and precise connectivity in terms of network technology available (Ethernet, Myrinet, Wireless, Modem) and IP parameters. This information is made available through the Cloud, so that precise characteristics of machines on the network are available, and allocated sets of machines can be as homogeneous as possible.

4. Experimental results

In order to validate the key ideas of the I-Cluster project, two approaches have been followed: experimentation with a prototype and simulation. This section

[3] http://www.vmware.com/products/virtualplatform.html

describes some experiments done with a prototype we developed [21] on which the main services from the I-Cluster specification have been implemented and tested. Section 5 will analyze our simulation results.

4.1. Prototype architecture and testbed

The experimental implementation of the I-Cluster Cloud is a key point for its validation. We implemented a prototype with a simplified version of the infrastructure. Because one of the key issues of the I-Cluster is the dynamic behavior of the resources—joining and leaving the Cloud—we chose a bottom-up approach, implementing the lower layers of the stack. Therefore the prototype architecture is built around the Sandbox layer, including several resource management services. Fig. 4 shows a high-level structure of the prototype architecture and its relation to the original I-Cluster stack.

To conduct some performance and availability tests (presented in Section 4.4) we installed the prototype in 25 workstations from PUCRS's computer science college's laboratory. One node is fixed in the prototype serving as root of the Cloud and does not calculate, so only 24 nodes are used in the following experiments.

Since students may freely use these PCs, the availability of the nodes in the Cloud never can be guaranteed, which turns it ideal for our tests. Each node is an HP E-PC with an Intel Pentium-4 1.6 GHz processor and 128 MB of RAM memory. The operating system used in the lab is Windows 2000 and when a machine joins the Cloud it changes to Mandrake GNU/Linux.

Because the testbed is limited to a small number of nodes, we did not implement the whole I-Cluster resource management algorithms (Section 3). We did the following simplifications for the prototype:

Sandbox. A module developed by HP Brazil called Mode Switch is responsible for the sandbox mechanism, alternating between the user operating system (user-mode) and the cluster operating system (cluster-mode).

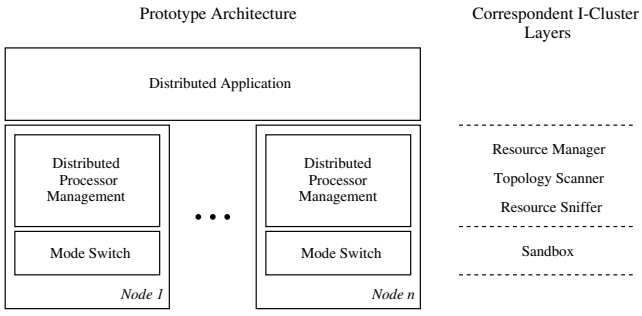

Fig. 4. Prototype architecture and its layers.

Resource Manager. We implemented a simplified version of the gossiping algorithm to allocate the available resources called Distributed Processor Management (dpm) [22].
Topology Scanner. This functionality was included in dpm as an additional service responsible to connect resources that join the Cloud in a logical topology. When a request operation is issued this logical topology is traversed by the resource manager to find free resources.
Resource Sniffer. This functionality was also included in dpm as an additional service responsible to track the configuration and availability of the resources.

The following sections describe the implemented prototype layers in detail.

4.2. Implementation of the sandbox

The Mode Switch module, developed by HP Brazil, is the module responsible for the sandbox mechanism, alternating between the user operating system (user-mode), and the cluster operating system (cluster-mode) as the I-Cluster Cloud specifies [23].

The main Mode Switch role is to keep track of the usage of the machines where it is installed and identify in which periods these machines are idle and could be exploited in the Cloud. Based on this knowledge the Mode Switch initiates the transition to the cluster-mode without user intervention. Its design has the following characteristics:

Efficient mode switching. Switching between modes is implemented in an efficient way resulting in switching times lower than 1 min.
Local use has higher priority. Immediate switch when the user comes back (no context is saved to reduce switching times).
Zero local interference. The Mode Switch installation does not have any impact on the target system performance when in user-mode.
No special hardware requirements. The Mode Switch is implemented completely in software. No specific hardware is needed other than a network connection.

At implementation level, the Mode Switch relies on a fixed-size partition (1 Gbyte) allocated on the node's native disk. Among other things are included in this partition traditional middleware and libraries for parallel computing (MPI, Blas, etc.), and tools to transfer the jobs and their data. These tools are part of the Linux distribution for cluster called Clic [24].

The I-Cluster partition is created in a highly transparent way when the I-Cluster environment is deployed. We designed a tool that enables the creation of a contiguous and permanent file in the user file management system (Microsoft Windows). We then automatically copy the raw image of the I-Cluster partition to this file. We then have an I-Cluster partition included in the user's native partition. This avoids all low-level operations by the user (such as partitioning or formatting the hard disk) and makes the installation automatic.

This partition is created in the user's native partition, such as no repartitioning of the disk is necessary at the installation of the Mode Switch. In order to make this mode-switch partition bootable, the MBR of the disk is also modified to install a customized boot-manager. This boot-manager is used to determine in which state the node is and to boot in the right configuration (user- or cluster-mode).

The determination of the state of the node is made by the Mode Switch agent, that runs on each node, detects idle time and analyzes if this matches a predefined pattern (e.g. it is longer than a specified delay, or occurs after 23h00). If yes, the agent updates the state in which the node should be, and triggers the hibernation of the node: a physical dump of the current node's state is made on disk. Then it causes the reboot of the node and the customized boot-manager will read the state and select the right mode to boot.

A state diagram of the Mode Switch is presented in Fig. 5.

Both context saving and restoring operations are done using system calls. So, since these context switching tasks are provided by the operating system API, they must be theoretically trustworthy and we assume that they occur without any data loss. The switching from user-mode to cluster-mode is presented in Fig. 5 as the transition labeled as ''Idle time detected''. After the context switching, the machine will boot the cluster operating system and then the node may enter the I-Cluster Cloud. Two events will force the node to return to user-mode: the detection of a local user presence or the end of the profiled idle time. In the first case, any mouse motion or keyboard hit is used to detect the presence of the local user. The second case will not require any user intervention because the Mode Switch will switch back to the user-mode, based on its knowledge about the workstation usage. In both cases the user-mode will be restored to the state it had before switching.

In the user-mode the Mode Switch presents itself as a small icon in the user's task bar to indicate it is running. As said before, it does not interfere with the normal operation of the machine and the user does not lose the control over the machine because the cluster-mode is only activated after a defined idle period.

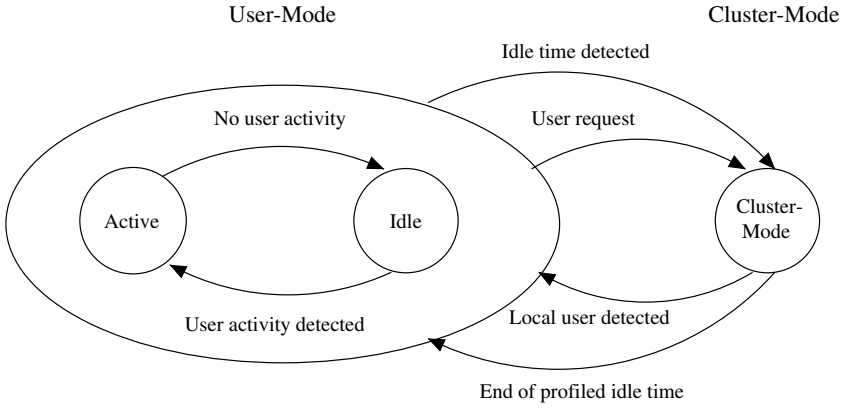

Fig. 5. State diagram of the Mode Switch.

Table 1
Switching times in a HP E-PC node

Mode-switching	Time (s)
User-mode to cluster-mode (Windows to Linux)	49.04
Cluster-mode to user-mode (Linux to Windows)	34.88

In Table 1 we present the switching times achieved by the Mode Switch in our test node (mean time of five measurements for each case). The node is a HP E-PC with an Intel Pentium 4 1.6 GHz processor and 128 MB of RAM memory. The operating system used in user-mode was Windows 2000. In cluster-mode the node runs Mandrake Linux with 2.4.3 kernel. These measures show that if a student tries to use a workstation that is in cluster-mode it has to wait only 35 s for the machine to become available.

4.3. Implementation of the resource management layers

Because of the small number of nodes available in our testbed we integrated the three I-Cluster resource manager layers in a prototype layer called Distributed Processor Management (dpm). In the following sections we will describe the core functionalities of dpm and the included services to implement the Topology Scanner and the Resource Sniffer.

4.3.1. Distributed processor manager (dpm)

dpm is a distributed processor manager that allows dynamic addition and removal of the nodes being managed [21]. It is totally distributed, this means that there is not any central module or manager that is responsible for the inclusion or exclusion of nodes. A daemon is running on each node of the cluster to control local information about the node. Once the nodes have booted in cluster-mode, they run the dpm daemon to organize themselves within the Cloud.

Users can allocate their jobs, and use an API provided by dpm to communicate with the system. So, user's applications can allocate more nodes at execution time and spawn their jobs among nodes that became free after the initial allocation.

To minimize network interference, dpm uses a hierarchical logical topology when communicating with the nodes. In the I-Cluster simplified prototype, the Cloud is structured as a tree. Thus the three possible states of a I-Cluster node are the following (see Fig. 6):

Node in user-mode. Represented in Fig. 6 by the white circles. These are nodes that are not idle and therefore are not part of the Cloud.

Free node in cluster-mode. Represented in Fig. 6 by the light gray circles. Because these nodes where idle, the Mode Switch booted the cluster operating system and connected them into a logical topology (tree based). This logical topology will be used to find free nodes when a request for the Cloud arrives. No distributed processing is running in these nodes because they where not allocated yet.

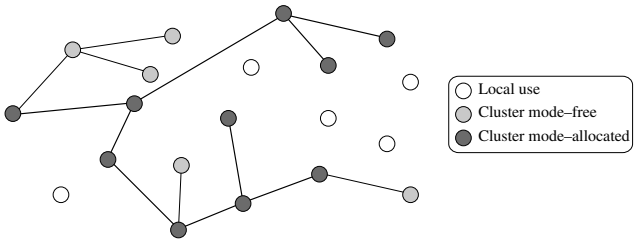

Fig. 6. Possible states of a node and the l-Cluster Cloud and the logical topology that connects them (tree).

Allocated node in cluster-mode. Represented in Fig. 6 by the dark gray circles. These are nodes in cluster-mode that are already allocated and executing a distributed application. The thicker connection lines represent the links to the other allocated nodes running the same application.

4.3.2. Topology scanner

After a node changes to the cluster-mode and boots the cluster operating system, it runs the Distributed Processor Management (dpm) daemon. dpm connects the node to the Cloud logical topology and implements a dynamic processor management ensuring that a distributed application may request additional nodes or partially release the nodes it is using.

Because dpm was originally not able to cope with resources joining and leaving the Cloud we implemented the Topology Scanner in the form of an additional service. In the prototype, the logical topology is a tree so that the Topology Scanner service has two main responsibilities:

- connect nodes that join the Cloud to the tree;
- disconnect nodes that leave the Cloud from the tree reorganizing the logical topology.

When dpm needs to connect a node to the Cloud it looks in the same network to find at least one peer already in cluster-mode to connect to him. If no node in the same network is in the Cloud it will be connected to the root of the tree.

4.3.3. Resource Sniffer

The Resource Sniffer is also implemented as a service of dpm. Each time a node joins the Cloud dpm sends information to a repository about its hardware configuration and arrival time. When it leaves the Cloud the departure time is registered. It is important to have detailed information about the hardware configuration of the nodes in the cloud (processor, memory, disks, etc.). This will be used later to match available nodes to an application request (Match Finder).

The repository was updated in the prototype with the standard E-Mail service. One message is send when a node enters the Cloud and another when it leaves.

The message header contains the node identification and a time reference and the message body the information about hardware configuration extracted from a special Linux system folder called `/proc/`.

This allows us to analyze the behavior of each individual resource and also about the available computing power of the whole Cloud in different time periods. Some statistical results are presented in Section 4.4.2.

4.4. Experimental results with the prototype

In this section we show some experimental results obtained with the I-Cluster prototype. We analyzed two aspects of the prototype: the performance executing a distributed application and the mean availability of the nodes.

4.4.1. Performance results

In order to validate the performance of the prototype we implemented a test application. Since communication costs may be very high among idle workstations spread in a university campus or different floors of a company building, the application should be coarse grained to allow gaining performance in such an environment. It is also important that partial results may be easily rescheduled to other nodes in the case that some machine abruptly leave the I-Cluster. Based on these characteristics we have chosen a distributed Ray-Tracer application for the first prototype.

POV-Ray (Persistence Of Vision Raytracer[4]) is a three-dimensional rendering engine. The program derives information from a file containing the description of a scene (objects, textures, lights and point of view) simulating the way the light interacts with the objects in the scene to obtain a three-dimensional realistic image (procedure known as ray tracing). Each pixel of the resulting image may be calculated from the scene description without knowledge of neighbor points. MPIPovray 3.01 is a parallel implementation of the above application that divides the image to be calculated in horizontal slices, mapping each slice to one slave process. A master process is dedicated to the image partitioning, slices distribution and screen drawing. Slave processes do not wait for the calculation of the whole slice and send ready screen lines to the master to allow real time drawing of the already calculated points.

We have changed the distributed implementation of the MPIPovray so that the dynamic I-Cluster nodes have the initiative to request work, as in a client/server relation. The clients (running on the I-Cluster nodes) connect to a server to request a section of the image, render the received section and then send the rendered lines back to the server. Thus, each node renders a section of the image and the complete image is drawn only on the server.

The use of the client-server architecture was chosen because all the initiative must start from the client side. As such, if a node is not in the cluster-mode, it is simply ignored. Also, the server does not store any information about the active clients. This is particularly important because the nodes of the I-Cluster may change to user-mode

[4] http://www.povray.org

at any time, and this characteristic allows the clients to abort their execution at any point of the code, without the need to do any extra work after the user activity detection.

Therefore, the server must guarantee that the entire image will be rendered independently if a section was delegated to a node that changed back to user-mode during the execution or no. This is done by re-sending the section that was lost when all other sections were received. So, when a client has to change back to the user-mode, it does not have to report this to the server, resulting in a more efficient switching operation.

The performance evaluation of the distributed version of POV-Ray was done rendering an image with 640×480 pixels (chess2.pov) in the testbed described in Section 4.1. The image has been partitioned in slices of 15 lines. This means that when a client requests a work to the server, it will receive a slice with 15 lines to render. The client will request a new section only after its previous 15 lines have been rendered.

The time that the application needs to render the image decreases as new clients are included in the calculation (Fig. 7). The time that the sequential version needs to render the image is 23 min and 52 s. Using the distributed version the best Speed-Up obtained was with 16 clients (a factor of 12.49) being the same image rendered in 1 min and 54 s. This was the performance cut-off point for this application with this load.

Thus the POV-Ray example shows that our prototype implementation of the I-Cluster environment can yield significant parallel gains for master/slave based applications.

4.4.2. Resource availability

With the Resource Sniffer (Section 4.3.3) we got the information about the availability of each of the 24 nodes. This gives an idea about the wasted processing power even in such a small test environment.

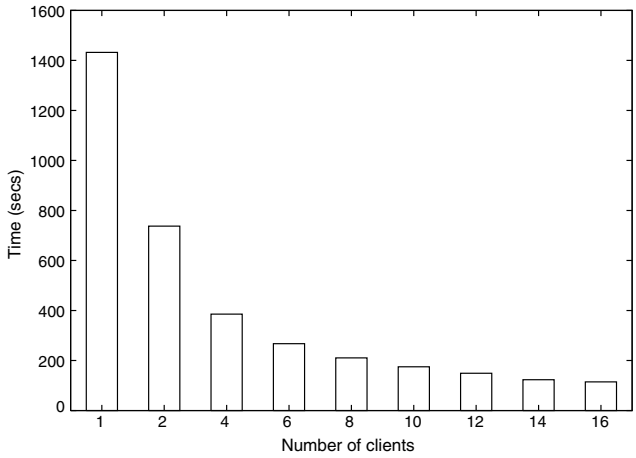

Fig. 7. Performance of the distributed pov-ray application over the I-Cluster prototype.

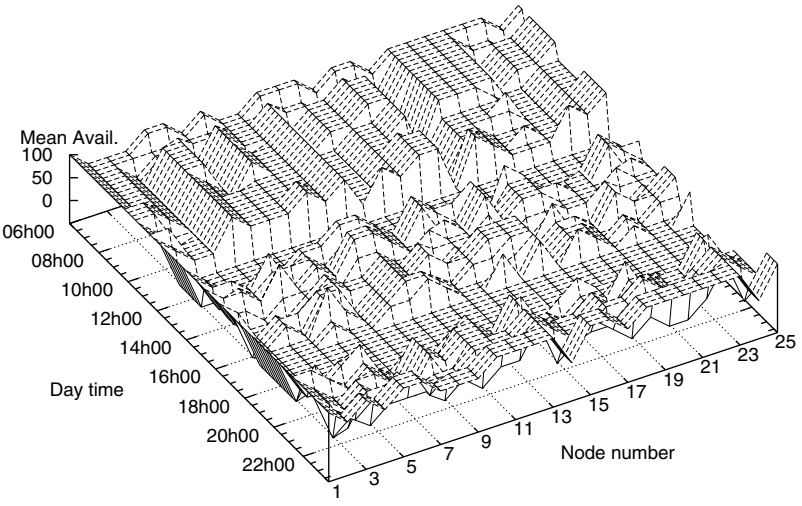

Fig. 8. Mean number of available nodes vs. time.

We got a three dimensional graph of a typical day of the week (see Fig. 8), whose z axis gives the probability that a node be idle and available for computation.

The prototype architecture is in production mode for more than one year in our lab. We use several applications that run for several hours along the day, losing and receiving extra nodes during execution.

5. Simulation results

The Cloud defines a set of distributed algorithms that offer a good resistance to changing conditions such as network partitions or disconnections; tolerance to machine volatility, as users may reclaim their machines at any time; self-configuration and automatic maintenance; and horizontal scalability up to a large number of machines (10,000 was our goal). The experimental platform of previous section allowed some practical validations. This section validates the scalability of I-Cluster's management algorithms.

The algorithms are derived from MOSIX's load distribution. MOSIX offers a very good convergence concerning information distribution time, as shown in Section 2.4, which let us hope that the Cloud could scale nicely. In order to validate this part of the project, we have used simulation: a simple simulation has shown (Section 2.4) how our gossiping algorithm allows the diffusion of the information in the Cloud. We used here a generic simulator, that has also been developed to validate more complex behaviors of the Cloud. First, this simulator is rapidly presented; the next one explains the kind of scenario that has been used to simulate the Cloud; at least, Section 5.3 presents the obtained simulations.

5.1. A generic simulator

In order to validate the way the nodes exchange their knowledge about their neighborhoods, we developed a simulator in Java, called GESSI.

The user defines his Events as being specializations of the generic and virtual `Event` class, such as for instance the `EventEnter` or `EventLeave` classes. Each one must provide an `exec(Node n)` method that calls n's right method to treat the Event.

A generic class `Node` is also proposed to the user, that implements some common characteristics of a node, and most of all provides the `exec()` methods that answer to the possible `Events`: when being notified by the Scheduler of a certain kind of `Event` (e.g. the reception of a message), the adapted `exec()` method will be called by the Node. The user has to derive a specific class (e.g. NodeIcluster) from `Node`, in which he or she will override or define the `exec()` methods that correspond to the Events that he defined. Some usual services are pre-defined, such as sending or receiving a message for instance. A Node is also defined to have some `Neighbors`, which are a fixed-size collection of references on other Nodes. The collection may be updated, for instance one Node may fusion its neighbors with those of another Node through the appropriate method.

All the possible `Nodes` are grouped into the `Universe` class, that provides a lookup service. A special package, the `Scheduler`, allows to design scenarios that are used to simulate the logical behavior of the nodes: it builds an "Agenda", which is a time-ordered list of lists of couples (`Node`, `Event`). The `Scheduler` then provides an `exec()` method that runs across the Agenda and notifies each Node of the Event that it has to deal with. Thus, GESSI's main simply initializes an `Universe` and a `Scheduler`, and runs the Universe's `exec()` method.

GESSI provides a user interface in order to easily specify scenarios (i.e. which time-step ordered events are notified to which nodes). We defined a grammar that is able to define events notified to nodes at different time-steps with a simple single line command. Wildcards and interval definitions are allowed to specify time-steps or nodes. This syntax is very much like the one used by the Unix cron instruction and has the advantage of being very simple to schedule periodic events to a high number of nodes. Since some algorithms may need special parameters, these ones have to be defined in a different file and loaded before the scenario. This way, GeSSi may use the same scenario file with different algorithms, making it easier their comparison.

5.2. Scenarios for the simulation

In order to fully validate the I-Cluster gossiping algorithm, we need to define the characteristics of the Cloud that we aim at simulating.

We will simply consider the following population of nodes: the *servers* will be considered as reliable nodes, that will never fail and never stop. They model for instance the servers of big firms or universities. The *workstations* will have an entry/exit rate

of medium importance: basically they will be available for computing at night, and used by their owner during days, except during a one-hour pause at noon. At last, the *PC* will represent the domestic, personal computers. They will be available only during daytime and have a much higher rate of failure.

This preliminary study will be based on the use of three time-zones, roughly corresponding to computers distributed among Europe and America, northern (USA) and southern (Brazil, Argentina and Mexico). Thus, some nodes will enter during nighttime in Europe while others, in America, are still used because it is daytime. The proportion of each type of node is also difficult to model. For this simple model we have used the data compiled by the http://www.journaldunet.com web-site. Summing-up all the available information we will consider the repartition of nodes given in Table 2.

Taking into account the number of servers and workstations sold and comparing it to the number of PCs sold in these same areas, we will consider that we have a ratio $\rho = 0.28\%$ of the number of PCs that are servers or workstations. At last, we will consider that each server hosts in average some 200 workstations.

Summing it up, given a number n of nodes that we want to simulate with GESSI, we will have the proportion of nodes of each type and geographic zone following the proportions given in Table 3.

5.3. Results of the simulations: scalability of I-Cluster

At the start of the simulation, the scheduler will then instantiate n nodes, following the required proportions for each category. The simulation will start at time

Table 2
Number of PCs in our population

	Total	Europe	USA	S. America
PC (millions)	193.5	96.5	85	12

Table 3
Proportion of nodes in the simulation

	Total	Europe	USA	S. America
PC	$\frac{n}{1+\rho}$	$0.5\frac{n}{1+\rho}$	$0.44\frac{n}{1+\rho}$	$0.06\frac{n}{1+\rho}$
WS	$\frac{0.995n\rho}{(1+\rho)}$	$0.5\frac{0.995n\rho}{(1+\rho)}$	$0.44\frac{0.995n\rho}{(1+\rho)}$	$0.06\frac{0.995n\rho}{(1+\rho)}$
Server	$\frac{n\rho}{201(1+\rho)}$	$0.5\frac{n\rho}{201(1+\rho)}$	$0.44\frac{n\rho}{201(1+\rho)}$	$0.06\frac{n\rho}{201(1+\rho)}$

n is the total number of nodes and $\rho = 0.28$.

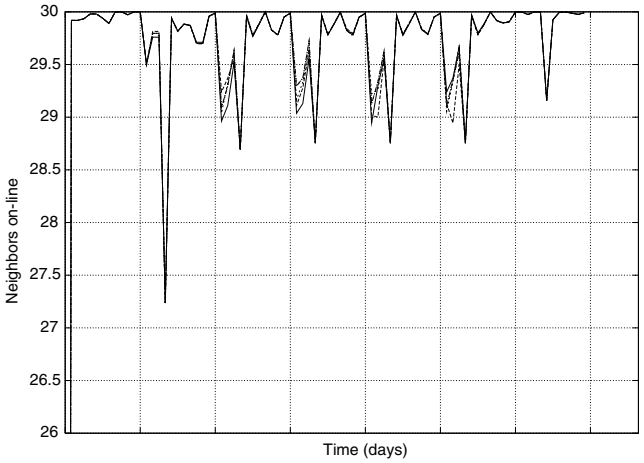

Fig. 9. Mean number of known, on-line, neighbors vs. time. 20,000 nodes have been simulated during 7 days (starting on Sunday). Each node has a buffer for a maximum of 30 neighbors. We superposed five simulations.

00h00 GMT on sunday. Fig. 9 shows the behavior of the nodes managed by the I-Cluster algorithm, as revealed by the mean number of neighbors known by a node that are on-line.

In this experiment up to 20,000 nodes were simulated during 7 days, starting on a Sunday, the chosen time-step being of 20 min. Each node has a buffer of up to 30 neighbors with whom it may exchange messages following the I-Cluster algorithm. The simulation took 10 min on a Pentium-4 CPU of frequency 1.70 GHz and with 400 Mb of RAM, using Sun's J2SDK and J2RE 1.4.1. We superposed five simulations to show that these results may be considered as statistically significant: very few differences are observed between each run.

As can be seen, a daily phenomenon appears, following the daily rhythm of the node's availability: at the start of the day (in Europe) all nodes have neighbors that are staying "on" (albeit some slight variations may be seen because of the noon pause). By the middle of the day, some of these neighbors that were workstations of the American zone disappear, because of the starting day there. Yet, the I-Cluster algorithm reacts remarkably well since within a few time-steps (and messages exchanges) all the nodes soon come back to filling their buffers with stable neighbors, until the next daily change. Finally, the influence of the end of the week-end may be noticed as a bigger failure in the mean number of known stable neighbors, on start of the second day (Monday) of the simulation. This failure, too, is quickly compensated by the algorithm.

6. Conclusions

We have presented the I-Cluster project, which aims at federating computing resources during their idle time in order to dispose of dynamic high-throughput

virtual clusters. We detailed some of the original contributions of the I-Cluster Cloud:

- a sandbox mechanism granted by an operational system switch, which turns the I-Cluster mode compatible with usual Operating Systems, as well as provides security for the normal user of the I-Cluster nodes;
- a management algorithm based on gossiping that provides an efficient and scalable way to disseminate the information about the nodes in the Cloud;
- a module that allows to profile the characteristics of the nodes.

Thus the Cloud is able to manage and use thousands of nodes, for instance distributed on an intranet as studied in [25], in order to run arbitrary jobs on the virtual clusters, without any interference with the normal users of the nodes.

We validated the I-Cluster original specification with a prototype based on 25 non-dedicated machines, equipped with the Mode Switch module to implement the sandbox mechanism, on which we tested a distributed ray-tracing application. We used an e-mail based tool to monitor the available nodes of this realistic implementation. As to the validation of the management algorithms, we presented different simulations that show the good scalability of the Peer-to-Peer Cloud.

The results, theoretical and experimental, obtained in the I-Cluster project are the following:

- an implemented sandbox mechanism that has a good switching time (more or less 1 min);
- a platform for idle cycles use that presented a good throughput on tasks-farm applications;
- a study of the availability of resources in an academic environment that revealed a good potential;
- an algorithm to manage the nodes with simulated high scalability.

An outcome of this work is also a generic simulator that may be used for the study of other algorithms for the management of distributed resources.

Current studies on the I-Cluster project include a checkpointing mechanism in the ID-IMAG laboratories, in order to avoid the loss of an interrupted task when a user comes back and his nodes are leaving the Cloud. The preempted I-Cluster jobs could thus be migrated to other available nodes of the Cloud. Another approach is the use of a network-booted middleware to enter the Cloud, instead of the disk-based Mode Switch. Augerat's ICATIS project[5] explores this possibility, which allows do avoid the local storage of a Linux partition on the nodes. As far as management algorithms are concerned, the CPAD – PUCRS/HP keeps on studying the evaluation and simulation of various solutions that could be incorporated to the dpm resource manager.

[5] http://www.icatis.com/

References

[1] A. Goscinski et al., Cluster operating system support for parallel autonomic computing, in: Proceedings of the First International Workshops on Operating Systems, Programming Environments and Management Tools for High Performance Computing on Clusters, 2004.

[2] J. Basney, M. Livny, Deploying a High Throughput Computing Cluster, vol. 1, Prentice-Hall PTR, 1999 (Chapter 5).

[3] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, J. Pruyne, A worldwide flock of condors: load sharing among workstation clusters, Journal on Future Generations of Computer Systems 12 (1996).

[4] I. Foster, The anatomy of the Grid: enabling scalable virtual organizations, Lecture Notes in Computer Science 2150 (2001) 1–457.

[5] P. Combes, F. Lombard, M. Quinson, F. Suter, A scalable approach to network enabled servers, in: Proceedings of the 8th International EuroPar Conference, 2002.

[6] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, S. Vadhiyar, Users' Guide to NetSolve V1.4.1, Innovative Computing Department Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[7] G. Fedak, Xtremweb: a peer-to-peer global computing experimental platform, in: Proceedings of FOSDEM Free Software and Open Source Developers Meeting, Brussels, 2002.

[8] D. Anderson et al., SETI@home: the search for extraterrestrial intelligence, Technical Report, Space Sciences Laboratory, University of California at Berkeley, 1999. Available from: <http://setiathome.ssl.berkeley.edu/>.

[9] A. Barak, S. Guday, R.G. Wheeler, The MOSIX Distributed Operating System, Springer, Berlin, 1993.

[10] B. Richard, I-cluster: the execution sandbox, in: Proceedings of the IEEE International Conference on Cluster Computing, 2002.

[11] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (7) (1978) 558–565.

[12] D.J. Watts, Small Worlds. The Dynamics of Networks between Order and Randomness, Princeton University Press, 1999.

[13] B. Richard, P. Augerat, N. Maillard, S. Derr, S. Martin, C. Robert, I-cluster: reaching top500 performance using mainstream hardware, Technical Report HPL-2001-206 20010831, HP Laboratories Grenoble, August 2001.

[14] E. Dodonov, J. Sousa, H. Guardia, Gridbox: securing hosts from malicious and greedy applications, in: Proceedings of the 2nd Workshop on Middleware for Grid Computing, 2004.

[15] L. Gong, M. Mueller, H. Prafullchandra, R. Schemers, Going beyond the sandbox: an overview of the new security architecture in the java development kit 1.2, in: USENIX Symposium on Internet Technologies and Systems, 2004, pp. 103–112.

[16] C. Hawblitzel, T. Von Eicken, A case for language-based protection, Technical Report 98-1670, Cornell University, Department of Computer Science, 1998.

[17] A. Tannenbaum, R. Van Renesse, H. Van Staveren, G. Sharp, S. Mullender, J. Jansen, G. Van Rossum, Experiences with the amoeba distributed operating system, CACM 33 (12) (1990) 46–63.

[18] V. Abrossimov, M. Rozier, M. Gien, Virtual memory management in chorus, in: Proceedings of the Progress in Distributed Operating Systems and Distributed Systems arrangement, 1989.

[19] K. Arnold, J. Gosling, D. Holmes, The Java Programming Language, Addison-Wesley, 1994.

[20] M. Abis, B. Dayley, An open alternative to java and C-sharp, http://ivm.sourceforge.net, 2000.

[21] C. De Rose, F. Blanco, N. Maillard, K. Saikoski, R. Novaes, O. Richard, B. Richard, The virtual cluster: a dynamic environment for exploitation of idle network resources, in: Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002), 2002.

[22] C. De Rose, H.-U. Heiss, Dynamic processor allocation in large mesh-connected multicomputers, in: Proceedings of EURO-PAR 2001, Manchester, England, Lecture Notes in Computer Science (LNCS), 2001.

[23] B. Richard, P. Augerat, I-cluster: intense computing with untapped resources, MPCS'02, Ischia, Italy, April 2002.

[24] O. Richard, Cyril Martin, Parallel launcher for clusters of PCs, in: Proceedings of Parallel Computing (ParCo), Italy, 2001, pp. 473–480.
[25] B. Richard, I-cluster: agrégation des ressources inexploitées d'un intranet et exploitation pour l'instanciation de services de calcul intensif, Ph.D. thesis, INPG, December 2003.