Neural, Parallel, and Scientific Computations 17 (2009) 257 – 280

A LOW-COST PLATFORM FOR CLUSTER AND VOLUNTEER COMPUTING

Dimitris Kehagias and Michael Grivas
Department of Informatics,
Technological Educational Institute (TEI) of Athens,
12210 Athens, Greece

Abstract
This paper presents the development of a low-cost multi-platform computing environment, consisting of a simple framework for Volunteer computing using Java, and the DYNER – a platform which combines NoWs and Clusters that can support high performance computing in a dynamic way. This platform can be used for educational and scientific purposes within the bounds of a university campus.

Keywords clusters**,** java, volunteer computing, education

1. INTRODUCTION

The last 15 years, the world of computing and programming has faced one of the greatest turns to its technology. Primarily, due to the progress of communication infrastructures, the basis of computing has moved from big, monolithic systems to distributed computing. In parallel, multi-core processors made their appearance in desktop computers. With these advances, the landscape of computing has been dominated by parallel and distributed variations. This progress enables the realization of cost effective powerful distributed computing systems. A large number of scientists have proposed and realized such systems, and have developed a lot of initiatives, frameworks and architectures to permit distributing processing across a network of connected systems.

Most of the computer science curricula include parallel and distributed computing in courses, such as parallel algorithms, parallel architectures, distributed systems, clusters and high performance computing (HPC). Educators are continually looking for an appropriate low-budget infrastructure that can be easily deployed and used to explore the distributed computing paradigm. Sharing resources on a computing platform can help achieve both of these goals.

We, in the department of Informatics of the Technological Educational Institution (T.E.I.) of Athens, offer several courses related to parallel and distributed computing such as “parallel computation” and “distributed computing”. For the purposes of these courses,

Received June 13, 2009

as well as for more computing power for faculty members' research, we decided to develop a low-cost infrastructure which easily can be deployed and used by both students and faculty members.

Another approach that stems from distributed computing is volunteer computing. Since breaking RSA encryption algorithm (Atkins et al., 1995), computers over the Internet have been used for massively parallel computations in both generic models like grids and project-driven uses such as SETI@home (Anderson et al., 2002) etc. Grids are usually built over well-defined infrastructures, while ad hoc approaches take advantage of idle computers. However, those approaches have mostly been used for well-specified concrete projects, where the type of processing and data structures has previously been analysed and well defined.

In this paper we present the development and implementation of a multi-platform environment that spans several modern distributed computing models of work, including clusters, volunteer and grid computing. The cluster part of the platform is a previous work of the authors on clusters and NoWs (Kehagias et al., 2003), which concluded to a dynamic, high-performance, versatile, multi-computer complex called DYNER, while the volunteer part is a new proposal. Our volunteer computing platform will serve as a massively parallel computations system, utilizing the idle time of computers all over the campus. It intends to overcome the drawbacks of the task-oriented solutions, without the limitations and preparation needed for grid computing. It is designed in such a way that can work under virtually any circumstances, both on-line and off-line, over telephone lines or Gigabit Ethernet, on multi-processor computers or older single-core 32-bit PCs.

The cluster and the volunteer parts are synthesized in a sophisticated platform that supports virtually any kind of distributed computing, as it is today, under a single control.

Next section surveys the existing works on volunteer computing, briefly presents the evolution of clusters and mentions the previous work of the authors on clusters and NoWs. Section 3 explains the objectives that drove this research. Section 4 presents the available computing resources. Section 5 explains the platform, its functionality and its implementation. Section 6 presents the qualitative and quantitative evaluation of the platform. Section 7 discusses the benefit that the platform gives to the department. The following section concludes with a synoptical view of this work.

2. PREVIOUS WORK

The proposed platform unifies under a single controlling mechanism, a previous work of the authors on clusters, with a new implementation for volunteer computing. Following,

we briefly present the clustering part of the platform and then, in detail the volunteer computing.

2.1 Cluster computing

Clusters brought parallel computing to the masses, providing a cost-effective alternative for computation-hungry applications. Beowulf class clusters (Sterling et al., 1995; Kaplan and Nelson, 1994) are dedicated, high performance homogeneous clusters that are deployed when performance is the main priority. The basic feature of a Beowulf class cluster is homogeneity, i.e., all of its nodes are identical and dedicated, except of the "front-end" computer, which acts as monitor and control. Processes are issued by the front-end. Usually, nodes are not directly accessible by the users, having no keyboards, mice, or monitors.

NoWs (Aderson et al., 1995) are heterogeneous clusters that take advantage of otherwise "wasted" computing cycles on unused computers. A master system takes job requests from authorized users, and then submits them for execution on any available workstation. The key differences from Beowulf clusters are heterogeneity i.e. stand alone end-user workstations instead of dedicated nodes. Such platforms have also enabled small organizations build or participate in clusters or multinational grids (Foster and Kesselman, 1999). Clusters share conventional technology, they are largely expandable, easier to understand and administer and they offer unrivalled availability (Baker et al., 2002; Baird, 2002).

Our previous work on clusters and NoWs concluded to a dynamic, high-performance, versatile, multi-computer complex called DYNER (Figure 1) that exhibited dynamically adapted performance, ensured cluster-level minimal performance and availability, as well as an interesting grid resemblance (Kehagias et al., 2003). The complex has been used primarily as an educational platform. Nevertheless, as a dynamic platform, it provides high-performance that can be used for research purposes, too.

2.2 Volunteer computing

Volunteer computing uses independent end-user computational resources that would otherwise be unused (Anderson et al., 2002). It has received much attention in recent years. It focuses on utilizing the available resources in volunteered computers. Volunteer computing systems communicate with a single point.

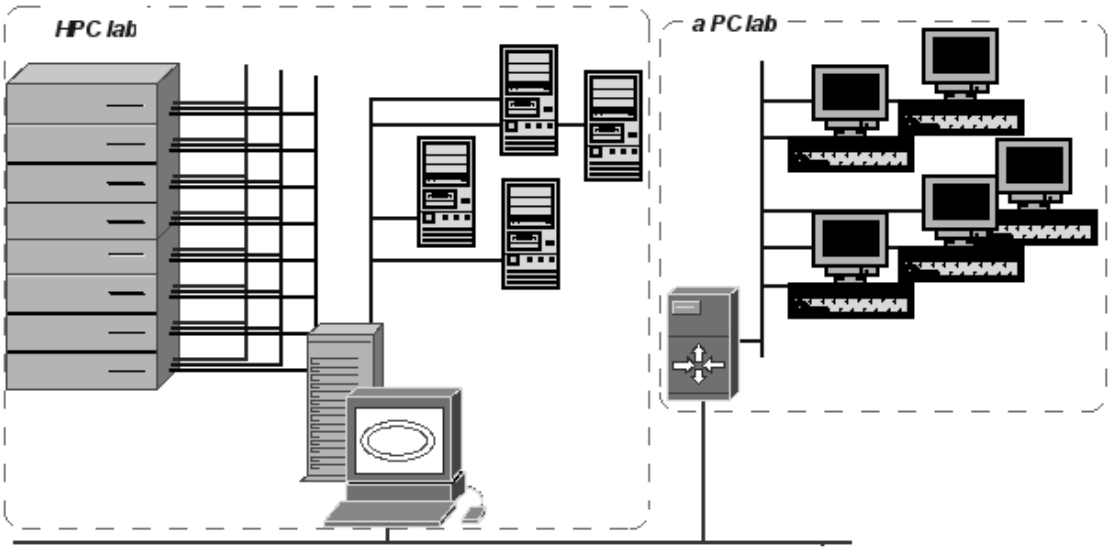

Figure 1: Initial multiple clusters & NoW platform

Historically, the first large attempt to use others' computing equipment for HPC was the RSA-129 code breaking project (Atkins et al., 1995). Then, others followed, such as the "Search for Extraterrestrial Intelligence" project or SETI@home (Anderson et al., 2002). For these projects, specific computational units have been built, usually in C/C++, which most of the PC of the era could run. The computational units had to be downloaded and deployed locally, by the node's user. Then, each unit downloaded automatically pieces of data, did the calculations and returned the results.

The term volunteer computing coined in the pioneering works for the Bayanihan framework (Sarmenta, 2001; Sarmenta and Hirano, 1999). It offers a simple framework, where programming is based on a quite simple API and it is easy to install. On client side it can run in any equipment that can have java on it. Bayanihan works on a strict farmer-worker model, i.e. each node works independently to each other, receives jobs from the farmer and returns results to it. There is no direct inter-node communication. It also provides solutions for security, fault-tolerance and scalability. On the cons side, Bayanihan does not cooperate or include any notion of grid computing or clusters.

Today, the possibly most known and respected framework is BOINC (Anderson, 2004), which is practically the generalization of SETI@home. In this framework, a client program must have been prepared, downloaded and installed in each client. Both server and client have been written in C++ and have been optimized for better performance and less needs in system resources. However, BOINC is quite difficult to initially install. It has quite some requirements on the server side (Apache web server, mySQL database system) and needs installation on the client side, too. As mentioned, the client part is prebuilt for each system, meaning that if a system is not supported, it cannot participate. Nodes must be under some professional control.

The newer SLINC project (Baldassari et al., 2006), compares to BOINC and intends to simplify development of the scientific jobs. SLINC, too, employees various techniques for result validation, such as majority voting, spot checking and migration against a possible threat. The architectures resemble, too. The basis is a client-server model, where each part is sub-divided into several components. Communication is XML-based Remote Procedure Call (RPC). That imposes, as the author admits, significant extra complexity and communication load. Programming for this framework needs also quite some knowledge and several additions to ones code, although it is considered simpler than BOINC. Last but not least, the server has no notion of cooperation or collaboration with same or other systems.

All those projects were based on specific, well-defined computations and a huge set of data, based on the SIMD model (Single Instruction Multiple Data). There are two drawbacks in such solutions: First, parallel processing means same instructions applied on different data and second, each solution is designed and dedicated to a specific project each time. On the other hand, generic solutions that reside in the grid computing area are based on grid infrastructures (Foster and Kesselman, 1999; Laszewski et al., 2002; Liu et al., 2002) on systems that have been designed or configured for that kind of work and which are available to the grid according to mutually known rules. Unknown computing equipment, with vastly fluctuating availability, which is not controlled by some cooperating authority, is usually out of question.

3. DESIGN GOALS

As explained, our prime objective was an infrastructure for the needs of both the parallel and distributed computing courses, and the research on high-performance computing.

The requirements that drove our design, where mainly *simplicity in use, expandability and simplicity in maintenance*.

a. A major requirement was *ease of use*. The platform implemented should be "attractive" to both students and researchers. That was accomplished by:
    i. A simple and obvious single user interface, which covers both underlying infrastructures and models of work. The user interface exposes most of the information needed for submitting, compiling, executing and controlling a job, while it keeps the visual parts quite simple and the flow of work minimal and obvious.
    ii. A simple way of submitting and controlling tasks, for both underlying platforms, either as single entities or as whole projects that consist of several tasks and files.

iii. Clear distinction of the two underlying platforms, i.e. the MPI-based clusters and the volunteer computing platform. Jobs for each platform should be distinguished and controlled independently, although from the same user interface and in a similar way.

iv. A simple way of building programs and projects for the platform, without extended modifications to source code or difficult preparatory tasks or deep knowledge of infrastructure. The clusters use MPI that is the most common way of programming in such platforms. The volunteer computing part requires solely java implementing our interfaces that is a minimal adaptation. It does not practically affect programming and certainly does not require any specific knowledge; neither does it use any additional libraries.

b. Apart from the general ease of use that applies to any kind of user, there were also needs for *flexibility for the researchers' use*. That stems for the diversity in tasks and requirements for the projects that will be hosted, the plurality in programming languages and libraries that are used in those projects, and last but not least the availability of resources and operating systems in each case.

c. The infrastructure should be designed in such away that *expandability* should not be an issue, in most cases. Research projects and other beneficial actions affect the platforms' morphology and breadth, as well as the infrastructures used. Hence, expanding it should be a trivial task, at least to the extended that it can be accomplished and to the breadth that can be controlled. The availability of equipment in any clustering or NoW form is in continuous reassessment, throughout the department, the campus and beyond. More laboratories are expected to enter as NoW, either interconnected or independent to each other. On the volunteer size, more personal computers are expected to enter the arena, as long as the platform proves its functionality and merits. Complexity in structure or simplicity in use should not impede expandability.

d. *Maintaining* such a polymorphic and complex structure may become impossible, if it has not been designed properly. The task of maintaining the whole platform should be possible to most final-year students.

4. AVAILABLE RESOURCES PARTICIPATING IN THE VOLUNTEER PLATFORM

Volunteer computing, as stated in its name, is based on volunteering resources. Such resources may become unavailable at any time. Before building our platform, in order to see its feasibility and the possible benefits, we first investigated the availability of personal computers within our campus. That investigation justified the implementation of

our platform. In addition, it provides a set of information that may guide modification for more efficient use of resources in the future.

Our investigation distinguished three categories of computers that can be used, based on their availability for participating in a volunteer computing scheme.

- Laboratory equipment within our department.
- Laboratory equipment of other departments
- Personal Computers of administrative services.

In our department there are 7 laboratories with 15 PC each, bearing 32-bit hyper-threading or 64-bit dual-core or quad-core CPUs, 3 laboratories with 15 low-budget PC (32 bit single core) each, 35 PC (32-bit hyper-threading or multi-core) for administrative or personal use, as well as 5 high-end servers (64 bit, multi-core). The laboratory equipment is used for the workshops averagely 6.5 months a year in a daily basis, from 8am to 8pm. The rest of their time they can be used either as parts of a NoW or within the volunteer computing platform. We should notice that even within workshop sessions, computers may have long idle periods. The high-end servers work 24h a day, 7 days a week, though normally lightly loaded. The rest of the aforementioned computing equipment is normally used during working days and hours.

The laboratory equipment of other departments includes around 400 computers, working roughly with a similar time-schedule. Their usage within our platform is a matter of discussion and investigation with the other departments. Colleagues are generally positive, since several research works would benefit from such and increase and availability of computing power.

The office computers of administrative personnel are about 300 contemporary PCs and are considered busy within all work hours. Their participation with our platform in their idle time is difficult, due to considerations from their users mainly about security. We believe that discussions and informative presentations will likely settle their objections. The computers that do not belong to the department constitute a very large pool of computing power that can be used on a volunteering basis. However, it cannot be properly assessed until we conclude with the discussions about their availability and participation. Within the field of control of our department, all computers participate or will participate in our platform, in either the cluster or volunteer computing part. The 150 laboratory computers will be available for at least the 12 non-working hours, another 2-3 hours within work hours and all weekends and holidays. The rest of the 35 computers from the administrative services can participate as part of the volunteer computing pool for 17 hours on work day and the whole weekends and holidays. The high-end servers are available when not busy. Apart from the obvious power of such a pool of computers, another important issue is availability when there is a need for more computing power.

5. PROPOSED PLATFORM

The combined platform consists of three major distinct parts: the central control entity, the cluster platform and the volunteer platform. Cluster and volunteer platform are further divided into server or farmer and nodes. In this section we present the above parts. At the end of the section, there is a description of the volunteer computing jobs.

5.1 Central control and monitoring

The central control entity plays the role of the single point of job submission and includes any procedure regarding the job and system control. It does also include a monitoring mechanism, which in turn is divided into farmer and clusters monitoring, although the monitoring could be separated.

The main tasks of the central control are (Figure 2):

- User administration: It authorizes the user's access to the projects and the jobs. It also presents the status of his/her jobs and their output. Finally, it may present statistics regarding the usage, time of execution etc.
- Accept, maintain and present projects and jobs: The central control takes care of each submitted work. It also takes care of projects. Each user can create, administer, use or simply see a project. The jobs will be assigned to the responsible underlying platform, i.e. cluster or volunteer computing, regarding the project's and job's attributes, as given by the user. It presents also information regarding submitted jobs and projects, either from its queue for unassigned jobs or from the responsible system. Finally, it shows the jobs' output, when finished or while executing, if they produce any.
- Present status of infrastructure: Either for administrative or for informative and educational purposes, the controller produces system status and information, for either underlying platform. The information presented includes statistics and current usage, for systems and jobs.
- Administer both platforms: The controller acts as the front-end to the administrator of the whole platform. Any administrative task is passed to the responsible mechanism for each platform.
- Those tasks are done in a simple, easy to use, manner, with a web-based application and over web-pages.

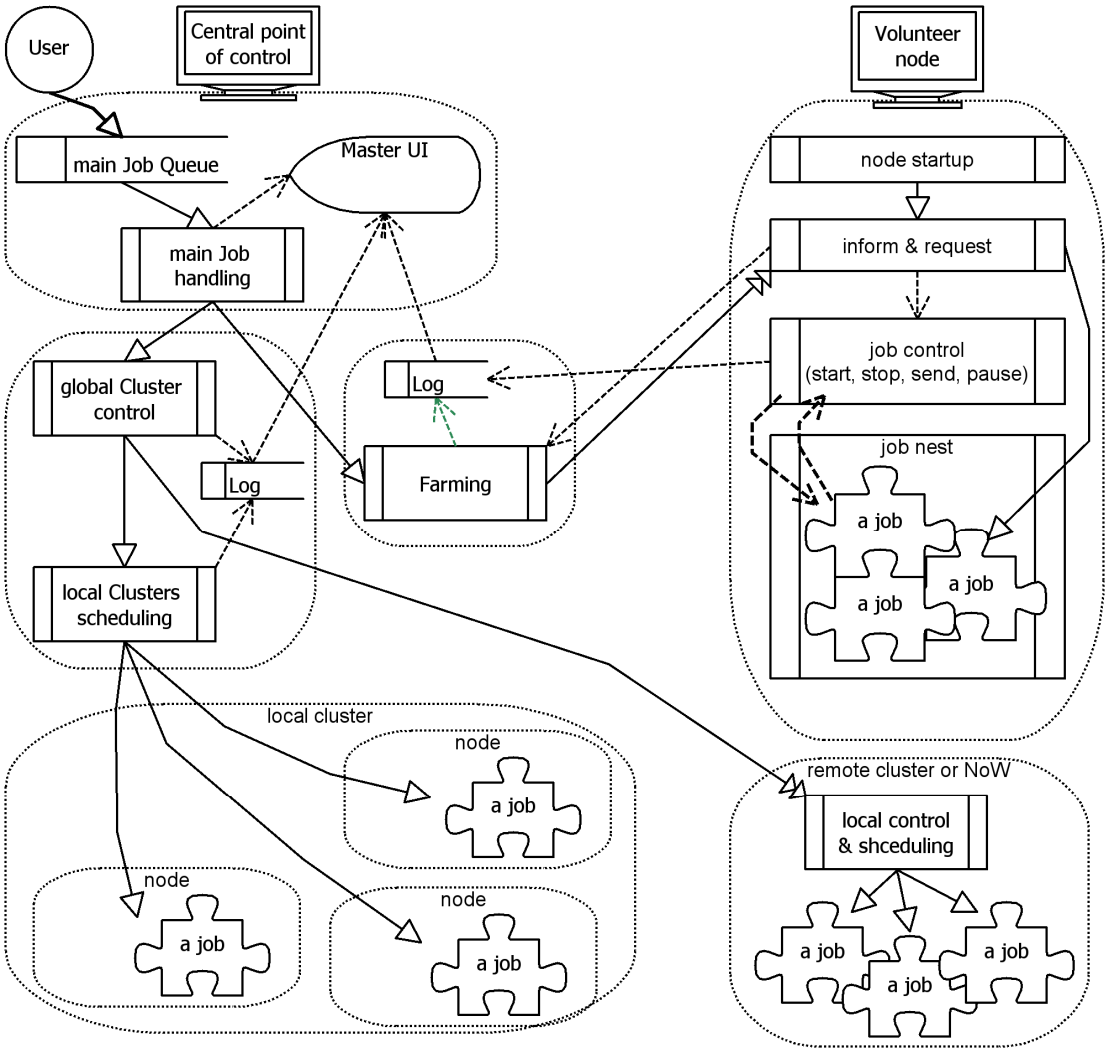

Figure 2: The internal structure

The central entity consists of the following parts:

- The User Interface: A set of dynamic web-pages that produce information from the Main Job Queue and the subsystems.
- The Main Job Queue: Solely keeps the incoming tasks (jobs), until they are taken. Tasks have an attribute that characterises them as for which platform they will use.
- The Job Handler: Its role, in brief, is to forward a task to the proper subsystem, i.e. the proper queue or scheduler.

The central entity keeps information for all given tasks that can be presented together with other statistics and the output. Information is extracted from its own structures (the queue and the job handler) and from the subsystems, e.g. the *mpd* tools for the MPICH.

In our implementation, the central controller resides in the same computer with the other two servers, although they may well be separated. The most important is that it cooperates with the rest of the services. Hence, the server as a whole provides a web-

ready, unified way to submit work, compile programs, schedule or run a task, control availability and performance, receive the results, check log files or start and stop services and devices. In general, it provides through web-pages most of the information and the tools that one may need for educational purposes or scientific, heavy-load computation. The system is based on common standards and well-known facilities and services, which makes it easier to communicate or cooperate with other systems, allowing it to participate in larger formations of systems, such as grids. It also allows virtually any expansion or new module one may need.

5.2 The Cluster & NoW platform

Server

The cluster server drives two clusters and a NoW directly, it supports scheduling and control for more clusters and NoWs, while it can communicate or control other schedulers based on the same system or even on different ones, it supports MPI directly or over redirected sockets to any system (Figure 3).

Scheduling happens in two levels: a global control takes the necessary actions to drive a job properly. This part is pluggable, that is it can be changed or adapted as required, even on the fly. Currently, it is deactivated, since a single MPI controls the existing clusters. In the future, as clusters of other departments may come in, proprietary schemes or existing solutions, such as condor, may take place. The second level is the common scheduling of a cluster. Currently, a single MPI-ring runs on our major cluster, which is connected to the server over a Gigabit Ethernet (Kehagias et al., 2003). We should notice that the server does not participate into any MPI-ring. It merely executes remotely the job to the first of the cluster nodes, which in turn initiates the parallel execution. The only exception is the NoW, where the scheduler has been informed for the availability of nodes and initiates the work in the first available node. This way we take advantage of each cluster's characteristics, like the dual Gigabit Ethernet and Myrinet in one of them, or the local dynamic assignment in the NoW.

The server is a heavy-duty tower computer designed for server use. It employs two Gigabit Ethernet NIC, one dedicated to the communication with the nodes and one exposed to the Internet. Its processor is a 3 GHz Hyper-threading Xeon and carries 1GB of RAM. The server does not normally participate in execution of programs, instead work as an administrative centre.

Cluster and NoW nodes

These nodes participate in a cluster and are directly controlled by the server. However, as parts of a cluster, they can communicate, according to the MPI manner. They all have

both MPICH and OpenMPI. In any one cluster, MPI may vary in version or communication characteristics. There is not much to say, since each node is simply an MPI-ready computer.

Our major cluster consists of 8 rackable computers, employing a 3GHz Xeon processor, two Gigabit Ethernet NIC and a fiber-optic 2 Gbps Myrinet NIC. One of the Ethernet NIC serves mostly common communication tasks that happen mainly between a node and the server, such as telnet sessions and boot sequence, while the other NIC is dedicated to the MPI-ring. They are all connected to a single Gigabit switch configured into 2 distinct Ethernet branches. The Myrinet NIC is again dedicated to the MPI-ring for the MPICH or used directly from the OpenMPI, when required. Myrinet optical fibers end up to a Myrinet switch.

For the NoW nodes, we have to notice that the way we have installed our systems allow better usage, regardless of what the student may do. Each node has an underlying Linux installed and each student, when logs in, works on a virtual machine, with his/her own copy of either Linux or Microsoft Windows. Hence, our MPI nodes that has been installed and initiated in the underlying system, does not get affected. It may merely inform the server about its work-load and gets jobs accordingly.

Computers that participate in NoW consist of common PCs that reside in laboratories. Nowadays, we have removed from the NoW configuration any 32-bit machines. The ones used are all Pentium 4, 64-bit PCs, with 1, 2 or 4 GB RAM, connected to 100 Mbps local Ethernet networks in each laboratory and to the TEI backbone over a gateway.

5.3 The Volunteer computing platform

As explained earlier, among the major objectives for our volunteer computing platform are simplicity in use and programming, as well as compatibility and interoperability; it should be able to cooperate with our existing platform.

Bayanihan (Sarmenta, 2001) is similar to our volunteer computing framework. The simplicity of its API is comparable but not as straight-forward as the proposed one. We do not study security, since we focus on forced *volunteer computing* (Sarmenta L.F.G. and Hirano, 1999), i.e. the computing equipment that is available within an academic institution. The complexity of keeping multiple copies for the majority-vote scheme, as well as the existing abstract classes that inhibit the programmer from following his own class schema, are two programmer-orientated differences. An important task-oriented difference is that our platform accepts and schedules multiple tasks, regardless of whether they execute on the same group of nodes. Furthermore, our platform has been built to coexist and cooperate with other systems. Most volunteer computing projects do not mix well or do not cooperate with other models of work, as seen in a previous section.

Our model intends to be a simple way to take advantage of the different virtues of each distinct model of work, such as a guaranteed high level of performance and availability that clusters provide, easy and controllable performance enhancements utilizing laboratory and other computing equipment, as well as vast augmentation in performance from the volunteer based large computers base.

Volunteer computing central, i.e. Farmer

The farmer mainly schedules the tasks from the central queue to available nodes (workers). It is also responsible to collect the results of the nodes or reassign any job that did not finish for any reason; it is also responsible to ensure that workers do function properly, either through communication or by time-out.

On the other direction, it is responsible of informing the controller and providing all information that a user may request (Figure 3).

Volunteer computing node

The other component of the system is the volunteer node. Volunteer node is any computing device that has a -permanent or not- connection to the Internet, can run java and has downloaded our Java program and uses it as screensaver. Its ability to run a job starts with a java program that runs as a daemon and acts as a virtual machine, a nest, for the given tasks. That program can start in different ways, either in boot time and remaining frozen while there is workload on the computer, or starting automatically when computer stays idle (e.g. as a screensaver). It can finally be started manually, for example as an applet when requesting a web page. The use of the screensaver simplifies the detection of workload or idle time, since it starts from the OS or the graphical subsystem, whenever a predefined amount of time passes without activity. Upon initiation, the nest program connects to the central farming system and sends a message declaring availability and informing about its “abilities”. The “abilities” refer to the node's characteristics that may be system information such as CPU type, clock frequency, amount of memory etc, or may be results of some small tests, such as short measuring computation, free memory etc. (Hayes, 1994). Something similar happens when coming back from a pause, where it merely informs about the recovering state.

The response of the *farmer* is a job. This job starts executing in the node's *Java Virtual Machine* (JVM), as a thread of the nest program. Communication is connectionless. This way we ensure that a loss of communication will not affect the system's performance; neither will it cause any serious problem to the project. The state of the communication is preserved in logs, both in farmer and node.

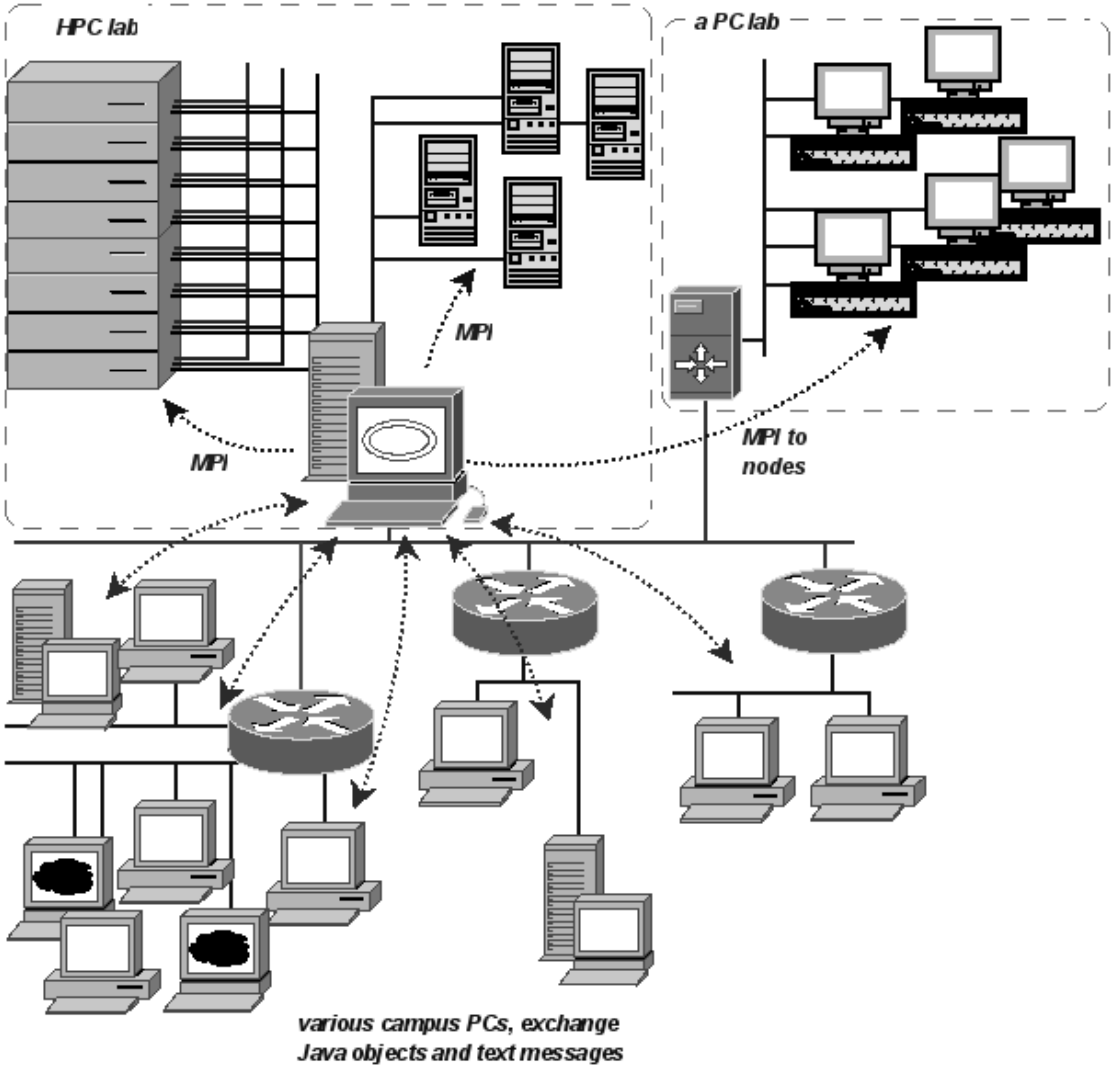

Figure 3: Synthesis - The current cluster, NoW & volunteer-computing platform

It should be noted that JVM offers extensive access control and protection against malfunctioning code (Anderson, 2004). Although JVM cannot ensure absolute protection against problems, it is quite safe to use it and quite protective against malicious code, limiting the access to the node's resources.

Our nest has been built to act in several ways. A screensaver automatically asks the nest program to stop, while the same can happen with the *unload* event in a web page. In any such case the nest program pauses all current jobs, saves them and its state for later use and releases as much memory as possible. Then, when it starts again, it can start from where it paused.

Job description

In order to benefit most of the system, any project has to be well split into parts that are totally independent to each other. Process intercommunication is out of question and any other intermediate communication would cause severe degradation of performance. Best case is when each job has a single starting point and a single end point.

Having said that, there are no technical limitations that hinder the system from supporting any kind of communication. Simple techniques of inter-process communication or even message-passing (MPI) can be supported. They would however multiply the delays caused by big latencies. Even Ethernet introduces latencies to the communication, to such an extent that the shape and relation of processing and communication must follow some patterns in order to benefit from parallelism (Chervenak et al., 2001).

In the proposed system, there are no procedures saved on a common site, neither happens any pure data transfer. Everything is done in objects and they are sent from the farming computer. There is no need for repository, network disks or any other data or process providing entity. Hence, data carry their way of process and there is no need for specific protocols on code or data manipulation. Object mobility is nothing new. However, our system uses java's ability to send over any communication channel its objects (*Serializable*). This way, we transfer over the internet and save to files the jobs to be executed, in the form of objects.

Our system's form and requirements for the jobs that will be submitted can be summarized in the following:

- Written in Java.
- Implement Serializable and Runnable.
- Implement our interface and its functions start(), pause(), kill(), init(), init(stream), serialized() and result().

Also, as discussed earlier, it should avoid communication, because that would introduce significant delays.

These functions serve all the necessary procedures for the system. They have to be implemented by the programmer that submits the project, but apart from that, he/she does not need to care about any other task. The jobs are sent automatically, they are initiated, served, stopped, re-initiated and get back their results. Following is a brief description of what these functions should do:

init() - Prepares the object/job to be executed. The programmer can put here anything that should be done before the object runs.

init(stream) - It is the same as init(), but the job is taken from a stream and initialised. That allows for saving the objects or transferring it to another site.

start() - It starts the execution of the object.

pause() - When the node's user comes back, the screensaver stops. Then it pauses the execution of the job, which will cease for the time. In order to be able to continue, it should save its state here.

kill() - For the unpleasant situation of a badly written program or other reasons that forces the node user or the central operator to cease a job.

serialized() - It returns a transferable form of the object. Most of it is ready by Java. However, it does not maintain its status, like variables, point of execution etc.

result() - After the job finishes, this function returns the results that are to be sent back to the farming entity.

No execution of job is blocking. Each object received by the node starts immediately as a thread (*Runnable*). A node can get more than one objects to execute, although it has not been studied as a plausible use of the system.

The execution of the objects may pause for some reasons. The most important one is user's request to work on the node. On the other hand, when the object finishes executing, it returns with its results. The results, too, are *serializable* objects.

6. APPLICATIONS AND PERFORMANCE

6.1 Efficient VSM-based parallel text retrieval using MPI

We used the cluster in order to experimentally evaluate an efficient Vector Space Model (VSM)-based parallel text retrieval algorithm, oriented to cluster environments (Mamalis et al., 2008). Our approach is based on direct VSM-based indexing, distribution and retrieval methods and efficiently handles the two main problems met in this case: (a) even distribution of the computational load (with regard to local scoring and local ranking subtasks) over the multiple processors and (b) efficient parallelization of the necessary global ranking sub-procedure, in order to gain both low response times and high speed-up values. For clarity and completeness regarding parallelism, the Parallel Text Retrieval (PTR) algorithm follows:

Steps 2–7 of the 'working processors' represents the 'clustering-based search mechanism' of the algorithm (searching first in the super-clusters and the clusters of the collection and rejecting the non-relevant ones by comparing the similarity of the corresponding centroid and super-centroid vectors to the user-query to some user-specified threshold value).

Root processor (processor with ID=0)	**Working Processors (processors with ID<>0)**
1. Read the terms of the user-query	1. Receive (via **broadcast**) the query-vector Q_i
2. Preprocess the query (stopping, stemming, phrases, thesaurus classes) in the same way as done for doc-vectors	**2.** Compute the similarity of Q_i with each one of your local Supercentroid Vectors SV_i using the well-known 'cosine-similarity' function **(local scoring)**
3. Formulate the query-vector Q_i	3. Extract the qualifying SuperCentroid Vectors (the ones with similarity value $SIM_{ij}(Q_i,SV_j) >= th1$)
4. Send the query-vector Q_i to all the other processors **(broadcast operation)**	**4.** Send the *IDs* of the qualifying Super Clusters SC_{ij} to all the other processors **(broadcast operation)**
…… …… ……	**5.** Compute the similarity of Q_i with each one of your local Centroid Vectors CV_i (the ones that correspond to clusters of the qualifying super-clusters) using the 'cosine-similarity' function **(local scoring)**
5. In the middle, act as a normal 'working processor'	*6.* Extract the qualifying Centroid Vectors (the ones with similarity value $SIM_{ij}(Q_i,CV_j) >= th2$
…… ……	**7.** Send the *IDs* of the qualifying Clusters C_{ij} to all the other processors **(broadcast operation)**
6. Receive (**gather operation**) the local ranked results ('R_i' document vectors' scores) from all the 'P' processors	**8.** Compute the similarity of Q_i with each one of your local Document Vectors DV_j (the ones that correspond to documents of the qualifying clusters) using the 'cosine-similarity' function **(local scoring)**
7. Sort the document-vectors' scores gathered during the above step and extract the final (of size R') ranked relevant documents list **(global ranking)**	9. Extract the final qualifying 'R_i' document vectors (the ones with similarity value $SIM_{ij}(Q_i,DV_j) >= th3$)
8. Present the final ranked relevant documents list to the user	**10.** Sort the above 'R_i' document vectors' scores locally, according to their similarity (score) – **(local ranking)**
	11. Send (**gather operation**) the *IDs* of the above R_i top-rank document-vectors to the root processor (i.e. processor with *ID=0*)

Steps 6-7 of the 'root processor' and steps 10-11 of the 'working processors' represents the 'local and global ranking/sorting task' of the algorithm with regard to the final document-vectors' scores (similarities to the user-query).

The corresponding experiments have been performed over the well known and extensively used TIPSTER/TREC WSJ standard document collection (almost 500 Mbytes data in full extent) as well as simulated multiples (up to 8 times WSJ – by reproducing the WSJ appropriately – which result up to 4 GB data) of this collection in order to take simulated results for larger collection sizes. All the measurements were taken based on the average running performance of 50 user-queries (the ad-hoc TREC topics 251-300). Specifically, we've run detailed experiments with respect to the following cases:

- **With and without the 'clustering-based mechanism'.** The best performance of our system was observed while using both the data sharing and balancing scheme and the three-based merge-sort algorithm. In the case of the simple operating mode of the system (without the use of 'clustering-based search' – Table 1) quite high values of 'speed-up' were achieved by running experiments with 2, 4 and 8 nodes over the WSJ collection. The maximum value of speed-up was almost equal to 6.05 when 8 processors were used which leads to approximately 75% 'efficiency' (6.05/8). The efficiency slightly decreases as the number of processor increase (i.e. with 2 processors it's equal to 82% – speed-up=1.64 – and with 4 processors becomes approximately 78% – speed-up=3.12) – thus, the maximum efficiency is achieved with 2 processors. This decrease is quite normal since with more number of processors (a) the communication overhead becomes more crucial than the computation times with regard to the total performance of the system, whereas (b) the merging overhead becomes more significant too. The above results are very satisfactory and even better than the ones achieved till now in related works (i.e. in (Rungsawang et al., 2001; Chung et al., 2001)).

  Slightly worse speed-up and efficiency values (however quite satisfactory too – Table 2) were also obtained when the built-in clustering-based search mechanism of our algorithm was used (the efficiency values for 2, 4 and 8 processors were 75%, 69% and 60% respectively), which is due to the extra communication operations required in this case for broadcasting the IDs of the qualifying clusters and superclusters throughout the network.

  Finally, the best response times achieved over the WSJ collection was 1.18sec over 8 processors (Table 1) without the use of clustering-based search and approximately 0.71sec with the use of the clustering-based search mechanism (over 8 processors – Table 2).

- **With and without the 'data sharing and balancing scheme'.** The results given in the previous paragraph would not be achieved if our specially developed 'data sharing and balancing scheme' was not used. Specifically, comparing the performance of our algorithm with and without the use of the data sharing and balancing scheme (in the basic operation mode – without the clustering-based mechanism – Table 3), the improvements given by running experiments over our cluster were very important. Specifically an up to 28.9% improvement was observed in total response times in the best case (2 processors, WSJ). This difference slightly decreases as the number of processors increases (i.e. with 4 processors it's equal to 26,7% and with 8 processors becomes 23,4%) which is quite normal since with more number of processors the

communication overhead becomes more crucial than the computation times with regard to the total performance of the system.

# of Processors	Speed-up ($S=T/T_P$)	Efficiency ($E=S/P$)	Response Time (sec)
1	-	-	7.14
2	1.64	82%	4.35
4	3.12	78%	2.29
8	6.05	75%	1.18

Table 1: Basic Experiments without clustering

# of Processors	Speed-up ($S=T/T_P$)	Efficiency ($E=S/P$)	Response Time (sec)
1	-	-	3.42
2	1.51	75%	2.26
4	2.76	69%	1.24
8	4.82	60%	0.71

Table 2: Basic Experiments with clustering

# of Processors	Response Time with DSB	Response Time without DSB	% Improvement
1	7.14	7.14	0.0%
2	4.35	6.12	28.9%
4	2.29	3.12	26.7%
8	1.18	1.54	23.4%

Table 3: With and without 'data sharing and balancing' – DSB (without clustering, WSJ)

6.2 Performance results

When developing a high-performance computing platform, it is crucial to measure its actual performance, as well as the performance scaling in relation to the number of nodes that participate. In this section we present the results of benchmarks on our cluster.

Many researchers have proposed simple, user-level tasks as measuring facilities that shows the overall system performance in specific kinds of tasks. From the differentiation among different kinds of tasks and several parameters, one can better locate the influencing factors. Such metric programs include simple tasks like array multiplication, or more complex sets of measuring tasks that extract immediately sets of results for several values, such as NAS (Bailey et al., 1995), and HINT (Gustafson and Snell, 1995) benchmarks.

Our work has been guided by the research interests of influencing colleagues. Specifically, our current research emphasises the use of cluster for load balancing studies, image analysis and information retrieval. Another important factor, due mainly to the educational perspective, was simplicity in configuration and result sets. We concluded in using the NAS parallel benchmarks (NPB) to demonstrate the performance of the cluster.

The NPB suite was developed by NASA to measure the performance of parallel systems. It consists of 8 tests, namely the Block Tridiagonial (BT), the Conjugate Gradient (CG), the Embarrasingly Parallel (EP), the Fourier Transform (FT), the Integer Sort (IS), the LU Decomposition (LU), the Multi-Grid (MG) and the Scalar Pentadiagonal (SP). Each benchmark has five classes: S, W, A, B and C. Problem-size grows from class S to class C.

For our studies, the most interesting tests are EP and IS that are compute and communication intensive, correspondingly. Our experiments were done with class B size, as moderately demanding and sufficiently accurate.

Since EP is a computation-bound program and requires almost no communication during the runs, it could effectively utilize the CPUs' resources without being concerned with the communication overhead, which makes the performance improve significantly. This condition is true regardless of the number of nodes of the cluster. Figure 4 shows the EP scalability of performance improved linearly from one to eight nodes. When an additional set of 8 NoW nodes participated, results kept nearly linearly scalable, although communication overheads are vastly different.

IS program is a large integer sort. This program performs a sorting operation that is important in “particle method” codes. It tests both integer computation speed and communication performance. Figure 4 shows the IS results, too, as they scale from one to eight nodes.

The progressive hysteresis against linearity can be explained by the nature of the test and the increasing number of messages passed between nodes that are limited by the communication latencies. Performance gain degrades to nearly flatten, when another 8 nodes from the NoW were included. That can be explained by the extreme overhead of communication over common channels between the server and the laboratory equipment, although those PC are available (idle) and the network lightly loaded.

As a measure of scalability, we selected the classically calculated parallel speed-up that is the ratio between execution on a single node and more nodes. The speed-up of EP and IS benchmarks are reported in Figure 5.

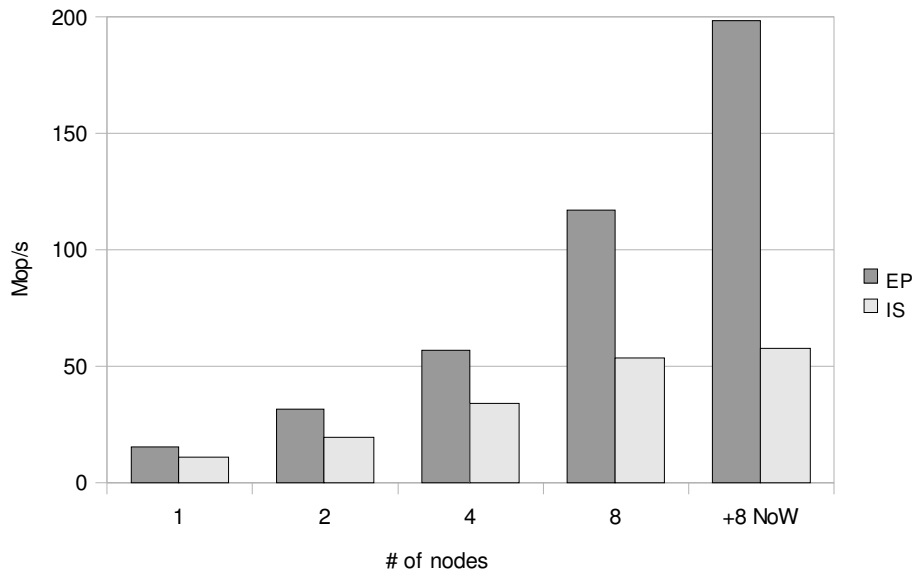

Figure 4: Performance measurements - EP & IS

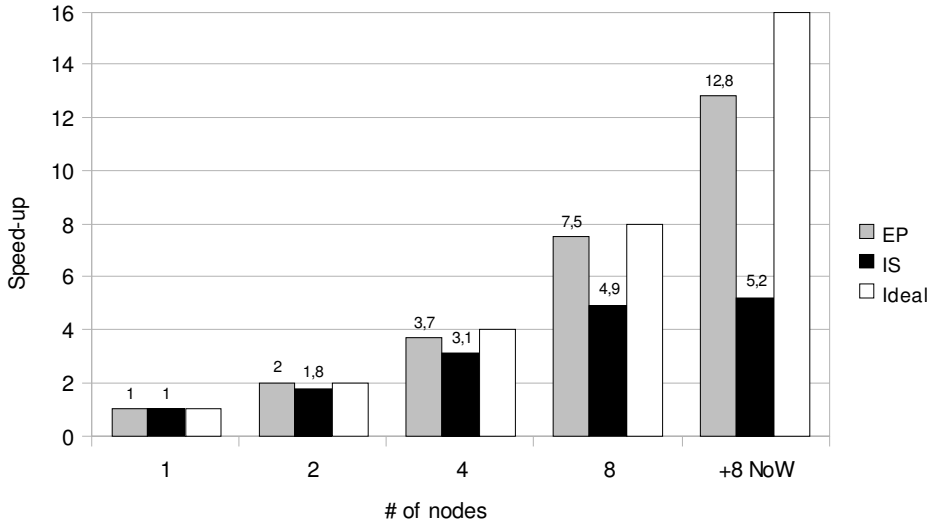

Figure 5: Speed-up of EP & IS

7. UTILIZATION OF THE MULTI-PLATFORM

Our model intends to be a simple way to take advantage of the different virtues of each distinct model of work, such as a guaranteed high level of performance and availability that clusters provide, easy and controllable performance enhancements utilizing laboratory and other computing equipment, as well as vast augmentation in performance from the volunteer-based large computers base.

The uses of such a platform practically span all areas of computing applications. In our educational institution, the intended uses are mainly research and education.

7.1 Utilization as an educational tool

The educational excellence derives from synthesizing different and incompatible areas of distributed computing, that allow students to study virtually any form of distributed computing. This allows focusing on either the technological details of administering such complexity and the programming issues that arise in each model. Currently, the students of the department of Informatics can benefit from the proposed multi-platform, by gaining hands-on experience working on and running various projects. These projects may be part of the educational process, mainly for the courses of "Parallel Computation" and "Operating Systems II".

The "Parallel Computation" course in our department covers a wide band of issues regarding parallel computations. The primary purpose of the course is to aid the undergraduates in thinking creatively about problems in a parallel fashion and designing parallel solutions using the MPI library. In the "Operating Systems II" course, the students receive an in-depth knowledge of distributed systems. The parts of the course that are related to the DYNER platform include distributed systems hardware and software, networking, communication protocols, remote procedure call (RPC), message passing, task scheduling and synchronization, atomic transactions, multi-threading, distributed file systems and security issues.

The utilization of the multi-platform as a teaching tool for the aforementioned courses will bring the following:

- Students will be able to work with parallel and distributed technologies and understand their relations and limitations
- Students will be able to apprehend the fundamental principles of parallel computation by carrying out various projects using MPI libraries
- Students will have the opportunity to experience distributing computing as well as to become aware of the computation leverage provided in a multi-platform environment
- Students will have the opportunity to appreciate the concept of system scalability and comprehend the notion of upgrade ability, to its smallest level, i.e. the improvement of a computing node
- Students will have the opportunity to learn the responsibilities and procedures of system administration
- Students will be able to carry out their thesis project

7.2 Utilization as a research tool

The proposed framework, as a research tool, provides a very powerful HPC platform that can employ several hundreds of CPU. The volunteer part, which by itself leads to a form of HPC, combined with the DYNER, results to a very valuable tool that provides a research platform for many faculty members, from a variety of disciplines, allowing them to conduct research requiring intensive computational power. The utilization of the platform as a research tool is not limited to faculty members but, following numerous conversations and exchanging of ideas with colleagues of other departments of the Institution, it appeared that the research community is interested in using HPC clusters as a tool for their research. This kind of utilization will enable the collaboration of multiple departmental labs and centres within the Institution and will bring together computational scientists, students, and researchers across the campus, over and within a common, shared computer platform.

Current use of the cluster part by faculty members takes place in three directions: reducing network overhead, scheduling algorithms and prototypes, and parallel text retrieval.

8. CONCLUSION

In this paper, we have described the main features of a multi-platform framework for cluster and volunteer computing. The proposed platform combines in a grid-like framework, clusters and volunteer computing for educational and scientific purposes within the bounds of a university campus. We also evaluated the performance of the cluster-side of the platform using benchmarking tools and a real application for Information Retrieval on clusters.

The platform's strength stems from its versatility and ambiguity: Each part serves better for different cases, while it expands infinitely, as long as more computers come into the group, dedicated, collaborative or volunteering. These create a wider area of application for the platform, in addition to more patterns of communication, management and organization to study.

The utilization of the platform as a teaching tool provided students an opportunity to experience parallel and distributed computing as well as to become aware of the computation leverage provided in a multiple computer environment.

Future directions of our research include scheduling algorithms per platform and for the whole complex, as well as interoperability with other systems and platforms.

REFERENCES

1. Aderson, T.E., Culler, D. and Patterson, D.A. (1995), "A Case for NoW (Network of Workstations)", *IEEE Micro*, Vol. 15 No. 1, pp. 54-64.
2. Anderson, D.P. (2004), "Boinc: A System for Public-Resource Computing and Storage", *Proceedings of the 5th IEEE/ACM Intlernational Workshop on Grid Computing*, November, Pittsburgh, USA, pp. 4–10.
3. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D. (2002), "Seti@home: An experiment in public-resource computing" *Communications of the ACM*, Vol. 45 No. 11, pp. 56–61.
4. Atkins, D., Graff, M., Lenstra, A.K. and Leyland, P.C. (1995), "The magic words are squeamish ossifrage" *in 4th international conference on the theory and applications of cryptology, Springer-Verlag*, Berlin, November, pp. 263–77.
5. Bailey, D.H., Harsis, T., Saphir, W., der Wijngaart, R.V., Woo, A. and Yarrow, M. (1995), "The NAS Parallel Benchmarks 2.0: Report NAS-95-020", *Nasa Ames Research Center*, Dec.
6. Baird, I. (2002), "Understanding GRID computing", *Daily News and Information for the Global Grid Communit*y, Vol. 1, July.
7. Baker, M., Buyya, R. and Laforenza, D. (2002), "Grids and grid technologies for wide-area distributed computing", *International Journal of Software: Practice and Experience (SPE*), Vol. 32 No.2, pp. 1437–1466.
8. Baldassari, J., Finkel, D. and Toth, D. (2006), "SLINC: A Framework for Volunteer Computing", *Proceedings of the 18th IASTED Intlernational Conference on Parallel and Distributed Computing and Systems - PDCS 2006*, USA.
9. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. (2001), "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", *Journal of Network and Computer Applications,* Vol. 23, pp. 187-200.
10. Chung, S., Kwon, H., Ryu, K., Chung, Y., Jang, H. and Choi, C. (2001), "Information Retrieval on an SCI-Based PC Cluster", *the Journal of Supercomputing, Kluwer Academic Publishers*, Vol.19, pp. 251-265.
11. Foster, I. and Kesselman, C. (1999), *The Grid: Blueprint for a New Computing Infrastructur*e, Morgan-Kaufman, San Mateo, CA.
12. Gustafson, J. L. and Snell, Q.O. (1995), "HINT-A New Way to Measure Computer Performance", *Proceedings of the HICSS-28 Conference,* Wailela, Maui, Hawaii, January 3-6.

13. Hayes, B.P. (1994), "The Magic Words are Squeamish Ossifrage", *American Scientist*, Vol. 82 No. 4, pp.312–316.
14. Kaplan, J.A. and Nelson, M.L. (1994), *A Comparison of Queuing, Cluster and Distributed Computing Systems*, Technical Report TM 109025 (Revision 1), NASA Langley Research Center, Hampton, VA.
15. Kehagias, D., Grivas, M., Meletiou, G., Pantziou, G., Sakellarios, B., Sterpis, D. and Ximerakis, D. (2003), "Building a low-cost high-performance dynamic clustering system", *Proceedings of the 6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services,* October 2, pp. 608-13.
16. Laszewski, G.V., Foster, I., Gawor, J., Schreiber, A. and Pena C. (2002), "InfoGram: A Grid Service that Supports Both Information Queries and Job Execution", *Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, Edinburg, Scotland.
17. Liu, C., Yang, L., Foster, I. and Angulo, D. (2002), "Design and Evaluation of a Resource Selection Framework for Grid Applications", *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, pp. 63-72.
18. Mamalis, B., Kehagias, D. and G. Pantziou, G. (2008), "Efficient Techniques for Parallel Text Retrieval on PC-Cluster Environments", *International Journal of Computers and their Applications (IJCA),* Vol.15, No.1, pp. 27-42.
19. Rungsawang, A., Laohakanniyom, A. and Lertpasretkune, M. (2001), "Low-Cost Parallel Text Retrieval Using PC-Cluster", *Euro PVM/MPI 2001, LNCS 2131, Springer-Verlag*, pp. 419-426.
20. Sarmenta, L.F.G. (2001), *Volunteer Computing*, PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.
21. Sarmenta L.F.G. and Hirano, S. (1999), "Bayanihan: Building and Studying Web-based Volunteer Computing Systems Using Java", *Future Generation Computer Systems: Special Issue on Metacomputing, Elsevier*, Vol.15 No.5–6, pp.675–86.
22. Sterling, T., Becker, D., Savarese, D., Dorband, J.E., Ranawak, U.A. and Packer, C.V. (1995), "BEOWULF: A Parallel Workstation for Scientific Computation", *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, Vol. 1, August, pp. 11-14.