Experiences in processing MPI applications in Condor environments

Paula Martínez[1,2], Jorge Ruben Santos[2,3], Emmanuel Millán Kujtiuk[1,2], Carlos Catania[2], Javier Díaz[5] y Carlos García Garino[2,4]

1- Instituto Tecnológico Universitario, Universidad Nacional de Cuyo, Mendoza, Argentina, pmart@uncu.edu.ar
2- ITIC, Instituto Universitario para las Tecnologías de la Información y las Comunicaciones, Universidad Nacional de Cuyo, Mendoza, Argentina, {emillan, ccatania, cgarcia}@itu.uncu.edu.ar
3- ICB, Instituto de Ciencias Básicas, Universidad Nacional de Cuyo, Mendoza, Argentina, jrsantos@itu.uncu.edu.ar
4- Facultad de Ingeniería, Universidad Nacional de Cuyo, Mendoza, Argentina.
5- Facultad de Informática, Universidad Nacional de La Plata, La Plata, Buenos Aires, Argentina, jdiaz@unlp.edu.ar

Abstract. Condor is a middleware specially design to manage job queues in dedicated and non-dedicated infrastructure. Condor seems to be a suitable tool to be used when the lack of available dedicated resources represent an impediment to run large application problems.
This paper discusses the use of Condor in processing MPI applications problems in homogeneous and heterogeneous pools of nodes.
In particular several experiments with different computing and network usage have been performed. From these tests it is concluded that Condor introduces a negligible overhead and also allows to conform scalable pools of nodes with a good cost-benefit ratio.

1 Introduction

Distributed Computing is a rather mature technology in USA, Europe and Japan. Different degree of development can be found in South America: Brazil and Venezuela have pioneered this area. Different conferences and workshops about High Perfomance Computing have been carried out in those countries since many years ago. More recentely, Grid Computing has become an intensive area of research in our countries. In particular, the Argentine Ministry for Science and Technology has recently launched the National Grid Initiative and High Performance Computing National System (SiNCAD) among other actions. In Argentina different scientific meetings take place such as The Parallel and Distributed Processing Workshop (WPPD), of the Argentinian Congress on Computer Sciences (CACIC) and more recently the High Performance Computing Symposia that is organized in context of the Argentinian Conference of Informatics (JAIIO).

There are many models and applications available for Distributed Computing which are discussed in section 2 of this paper. In this context the authors have

been working with the Condor middleware [1,2]. The first author Master Thesis [3] deals with this middleware and its applications has been reported in previous works [4,5,6,7,8]

Condor middleware is very well known due to its capacity to collect idle resources in order to configure a distributed computing infrastructure. Condor can manage and process a large quantity of batch jobs as well. However Condor has another attractive features such as Matchmaking and Scheduling modules, ClassAds submit files, interaction with Globus, possibility to process different jobs (old style vanilla applications, java codes, condor compiled applications, parallel programs and so on), etc.

This paper discusses the MPI application processing over partially dedicated resources managed and collected with Condor (for instance overnight idle cycles of teaching labs). MPI applications naturally would be executed on High Performance Computing installations like dedicated clusters, but it is not always possible to have this kind of resources and auxiliary installations (cooling systems, power, etc.) available. Then, partially dedicated resources can be a valuable alternative for many institutions in our country.

Different issues have to be addressed in order to process successfully MPI applications on Condor environments like administration of partially dedicated resources; MPI setup (configuration files; library installation, NFS protocol, etc.). Once these requisites are solved, Condor's overhead has to be considered.

The paper is organized in the following sections: an overview of distributed computing applications and models is included in section 2. Processing of MPI applications over Condor environments is discussed in section 3 where a short Condor overview is included as well. In section 4 the available infrastructure for testing and preliminary results are presented. The results of some applications problems are discussed in section 5. Finally, conclusions are provided in section 6.

2 Distributed Computing Models and Applications

In the context of distributed computing, the following paradigms can be distinguished:

High Performance Computing (HPC): In this paradigm prevails the execution of tasks, as efficiently as possible. Concepts such as parallelization and multithread fall within this area, and whose direct application is to make calculations that can last several weeks running on a single node, can be divided among several nodes, sharing the work to be done.

High Throughput Computing (HTC): In this paradigm prevails the execution of as many tasks as possible. Concepts such as queues and nodes managements are part of this area and its direct application goes through the completion of as much work as possible over time, sometimes a very large ones.

Grid Computing: The paradigm proposed access to different types of nodes: computing, storage, instruments, etc. This is based on the formation of virtual

organizations that share nodes. The name comes from an analogy with the Electricity Network (Power Grid) since this technology allows access to compute nodes in a similar manner that the energy is obtained by plugging in a node to the network.

Cloud Computing: is a way to share computers resources available on demand. Software requirements are provided as a service, which are located in data centers (cloud or clouds), allowing access to these services without the need to have the required infrastructure locally (power computing, storage, etc..) and usually also without requiring the user to have the knowledge or experience to use their services.

Although the HPC paradigm is well known in our country, there is no much experience in the case of HTC. The paradigm of HTC can be easily implemented when one has a multitude of medium / small sized calculations, and you want to speed up the management of all of them.[1]

Condor[2] is a distributed environment especially designed for HTC that makes use of idle nodes. The main idea behind Condor is to enable the management of high-cost calculations to be executed quickly and efficiently [4].

The evaluation and selection of the more convenient model for different applications that can be processed in Distributed Computing infrastructure requires a relationship between architecture and applications [9]. In order to do that, granularity and parallelism are the chosen variables. Granularity is defined as the ratio of Computing vs Communications time. Figure 1 illustrates the idea. For instance, embarrassingly parallel applications can be processed on a typical HTC middleware like Condor while for parallel applications based on a MPI library [10,11,12] a Cluster is suitable.

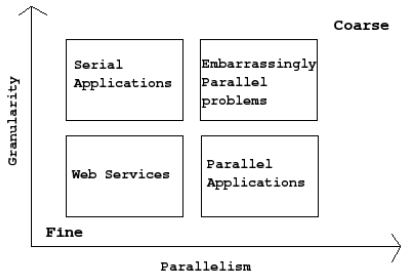

Fig. 1. Distributed computing taxonomy in terms of Granularity versus parallelism [9]

3 Processing MPI Applications Over Condor Infrastructure

This section discusses the processing of MPI applications over Condor Infrastructure. In subsection 3.1 a brief overview of Condor is provided in order to present some background material necessary for the cited discussion.

3.1 Condor Overview

The middleware Condor [1,2,13] is a versatile distributed computing tool. Among other features its main advantage is the capability to manage and execute a large quantity of batch jobs on partially dedicated resources. However Condor presents another interesting characteristics like the interaction with Globus [14] through Condor-G and G-RAM protocols, and matchmaking and scheduling possibilities to manage the submitted jobs.

Different kinds of applications can be processed with Condor: vanilla codes, java applications, Condor linked applications, parallel programs and so on. This can be accomplished by using different universes that are invoked in the ClassAds files.

Embarrassingly parallel applications are naturally managed by Condor. Applications related to Artificial Intelligence and Intrusion Detection Systems has been discussed in the work of Martínez et al. [5] and parametric studies of Solid Mechanics problems have been carried out with SOGDE-Condor application [6,7]

MPI applications can be processed in Condor environments as well, but different configuration and administrative tasks have to be done. In next subsection a detailed discussion is provided.

3.2 MPI Applications and Condor Middleware

The execution of MPI [10,11] jobs over Condor requires to do some previous configuration tasks. The next list shows the steps necessary to make Condor work with MPI:

1. Install the MPI library in the master and slaves nodes.
2. Exchange SSH keys between nodes, MPI needs to access all nodes without password.
3. Configure NFS in the master and slaves nodes, almost every MPI application needs to read input files and write output files from / to a common filesystem.
4. Tell Condor which node is the Dedicated Scheduler.
5. Parametrize the bash script *m*p2script [15] in accordance with the execution environment.
6. Edit the bash script that sets the environment variables and necessary instructions for the execution of the application (*s*cript.sh).
7. Write the ClassAds for Condor.

In the case of dedicated clusters, a Linux distribution like Rocks [16] solves the first three configuration steps. The dedicated clusters used in the experiments reported in this paper have this distribution. Consequently the installation and configuration process are easier and faster. In this case, Rocks offers an environment where is not necessary to exchange keys between nodes, since it already does the exchange during the installation process. Also includes various MPI libraries such as OpenMPI [17] and MPICH2 [11], and provides NFS shared directories between all the slaves nodes. In the case of partially dedicated resources, Rocks can not be installed and the cited facilities are not available. Then all the listed tasks have to be executed manually.

In the Condor pools available in this paper (see the infrastructure section), MPICH2 library is used and the installation was performed via NFS in the slaves nodes. MPICH is installed in a local directory of the master node. This directory is exported via NFS and mounted to each slave node.

It was necessary to exchange the SSH keys between the partial dedicated slave nodes and the master. In order to simplify this task, the home directory of the condor user was exported via NFS from the master node and mounted on the slave nodes. All slave nodes share the same SSH key and the same home directory. In this way, item 2 and 3 from the above tasks are solved.

When Condor is configured in the pool, one of the nodes has be selected as a dedicated scheduler. The MPI jobs are to submitted from this scheduler. When the dedicated scheduler claims for a node, it will try to use it immediately, and if for some reason this can not be done during a certain amount of time that node will be left, and will be available for opportunistic use.

Once MPI is invoked from Condor, MPI daemons will start on all the nodes previously assigned for the job execution. Then Condor is no longer involved in the execution of the MPI job. The middleware waits until the jobs finishes and returns the control. At that time, a Condor file reports the total execution time and, if for some reason the execution was not successful, it indicates the errors occurred. Each resource is controlled by the dedicated scheduler, and must have its name included in the local configuration file. [18]

Briefly, Condor is only involved at the matchmaking point, when the job is submitted and at the end of the process, once the MPI daemons have finished theirs jobs. A notification to the user is sent through pre-configured files. During the configuration of the Condor submit file, a script called *m*p2script must be specified as the executable script. This script contains the path from where the MPI daemons are launched on each node and the number of MPI daemons to be launched. This script receives from Condor the list of nodes which will run the parallel job, and executes the MPI daemon in each node chosen by Condor. Once the job has finished normally or abnormally, the script notifies Condor and generate the corresponding outputs. Depending on the parallel application that will be executed, another script must be configured, which is sent as part of the arguments, called *s*cript.sh. This file contains some environment variables in order to specify some tasks that are necessary preconditions for the job to run.

Then this script performs the execution of the application. Table 1 shows this script.

Table 1. Script.sh example

```
#!/bin/bash -x
EXECDIR="/home/condor/mc2tst"
export rep_from_which_model_is_launched=‘pwd‘
export AFSISIO=$EXECDIR/./mc2/data/
export F_UFMTENDIAN=big
ulimit -s unlimited
cd $EXECDIR
mkdir $EXECDIR/tmp
export TMPDIR=${EXECDIR}/tmp
$EXECDIR/mc2dm.Abs > $EXECDIR/salida_mc2.out
```

3.3 Advantages of Using Condor to Run MPI Jobs on partially dedicated resources

The advantage of launching an MPI job with Condor resides in the fact that is not necessary to configure a file with all the available resources (which may or may not be dedicated). If jobs are launched with native MPI, this file must be manually configured, which is commonly called *m*achinefile or *m*pd.hosts. This file should contain the names of machines and number of MPI processes to run per node.

Besides assembling this file with the nodes to run the job, Condor allows to specify any restrictions or requirements that the nodes have to have to run the job. For example, the node must have an especific load average, below an spcified number, a minimum amount of RAM memory or a minimum of free disk space. These requirements can not be specified when using MPI alone, without Condor or another batch schedulers like PBS [19], LFS [20], Torque [21] and so on. See for instance “mpiexec” commands for additional information [22]. However Condor can manage heterogeneous and non dedicated resources while the other schedulers are used for clusters only.

Condor will submit jobs only to those nodes, leaving the others for opportunistic use.

In Table 2 an example of a Condor submit description file (ClassAd) for a parallel job is shown. This ClassAd was configured to submit the meteorological application Mesoscale Compressible community model (MC2)[25]

Table 2. Condor ClassAd

```
Requirements= (machine == labredes01.labredes.ryt.itu.uncu.edu.ar)||
machine == labredes02.labredes.ryt.itu.uncu.edu.ar)||
machine == labredes03.labredes.ryt.itu.uncu.edu.ar)||
machine == labredes04.labredes.ryt.itu.uncu.edu.ar)||
machine == labredes05labredes.ryt.itu.uncu.edu.ar)||
machine == labredes06.labredes.ryt.itu.uncu.edu.ar)||
machine == labredes07.labredes.ryt.itu.uncu.edu.ar)||
machine == labredes08.labredes.ryt.itu.uncu.edu.ar)||
machine == labredes09.labredes.ryt.itu.uncu.edu.ar))
log = logfile
output = outfile.$(NODE)
error = errfile.$(NODE)
arguments = script.sh
machine_count = 8
should_transfer_files = yes
when_to_transfer_output = on_exit
queue
```

4 Infrastructure and Preliminary Examples

4.1 Infrastructure

In order to process the jobs for the experiments carried out for this paper, three dedicated clusters are available. These ones Twister, Storm and Opteron, respectively. There are also two part-time teaching laboratories. These pool are interconnected through a node called Tesla which acts as a router, as can be seen in figure 2.

Storm has a master workstation serving as front end, and 12 slaves nodes. All have a 3.0 GHz Intel P4 processor, 1GB of RAM memory, SATA HDD of 80 GB, and a Gigabit Ethernet network adapter. In this cluster was installed the Rocks 5.3[16] distribution, Condor 7.4.1 as middleware Condor and MPICH2 version 2[11].

Twister has a master workstation serving as front end and 8 slaves nodes. All have a Core 2 Duo 3.0 Ghz processor, 4 GB of RAM memory, 160GB of HDD and a Gigabit Ethernet network adapter. In this cluster was installed the Rocks 5.0 distribution, Condor 7.4.1 as middleware and MPICH version 2.

Opteron has 4 nodes consisting of 242 AMD Opteron processor of 1.6 GHz and 2.0 GB of RAM memory each one. The distribution Debian Lenny 2.6.18skas kernel was installed as operating system and Condor 7.4.1 as middleware. This cluster has MPICH version 2.

Two teaching laboratories, called Operating Systems and Networking, have 20 and 9 nodes respectively, with similar characteristics. These nodes have a Pentium 4 2.8 GHz processor, 80 GB HDD,1 GB of RAM memory and Fast Ethernet NIC. Both laboratories have CentOS 5.4 as operating system, Condor 7.4.1 as middleware and MPICH version 2.

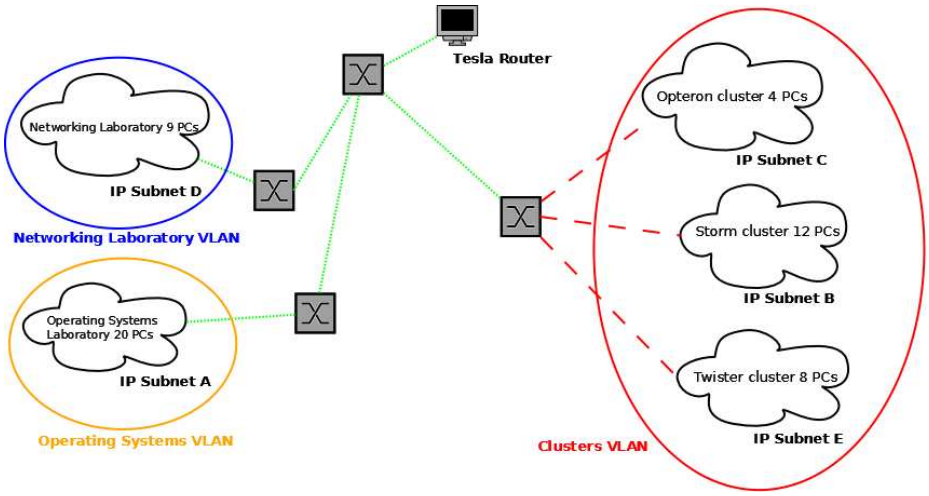

Fig. 2. Infrastructure used to process preliminary and application examples

4.2 Preliminary Examples

First, a serial program to integrate a function by the method of trapezoids [12], was run in one node of each pool. The results are shown in Table 3. This experiment was performed to obtain performance indexes of the available pool

These results allowed to calculate the performance index to measure the computing power of individual nodes of the pool in relation to the cluster front end Twister, and the performance cumulative index, which is useful for applications with scanning parameters for load balancing. The results of these experiments are shown in Table 4.

In order to obtain another measurement of the performance index of the pool, a serial algorithm of matrix multiplication was run on one node of each pool. The results can be observed in table 5. A matrix dimension of 4096 has been considered.

Table 3. Trapezoids method

Pool	Time (seconds)
Twister	18
Opteron	46
Storm	61
Laboratories	65

Table 4 shows the performance index obtained for each pool.

Table 4. Performance index

Pool	Hardware characteristics	Number of nodes	Number of cores	Processing time	Relative Performance Index	Cumulative Performance Index
Twister	Core 2 duo 3.0GHz 4 GB RAM	16	32	18	1	32
Opteron	Opteron 1.6 GHz 2GB RAM	4	8	46	2,55	3
Storm	Pentium 4, 3.0 GHz 1GB RAM	12	12	61	3,39	3,5
Networking Laboratory	Pentium 4, 2.8 GHz 1GB RAM	9	9	65	3,61	2,5
Operating Systems Laboratory	Pentium 4, 2.8 GHz 1GB RAM	20	20	65	3,61	5,5

Some conclusions drawn from table 4. Twister cluster performance is markedly superior to the rest of the nodes. Note that all nodes have a estimated combined computing power of 14 cores of the Twister cluster. This performance difference is not important if all nodes are used for parameter sweeping problems or validation statistics in Artificial Intelligence.

However, the reality is very different for simulation application problems governed by differential equations which usually divide the spatial domain of interest. In this case, the iteration "n+1" can not be processed until all the calculations of the "n" iteration have been calculated. If the domain of interest is evenly divided by all nodes (in other words assigning the same number of finite element, finite difference nodes or equivalent in other codes), then some nodes will delay up to 3.5 times the cluster Twister. Then in this case does not make sense to add a node to the cluster Twister, because the execution will be delayed.

Table 5. Serial execution time

Pool	Time (seconds)	Relative Performance Index
Twister	1248	1
Opteron	3646	2,78
Storm	3341	2,68
Laboratories	4678	3,75

5 Application Examples

5.1 Matrix Multiplication

The matrix calculation is one of the most used tools in the context of numerical methods in general and in the simulation of engineering problems in particular. One of the most commonly used operations in the matrix calculation is the product of matrices. This operation is very suitable for parallelization.

The Cartesian[23] and Fox[12] parallel algorithms implement the matrix multiplication in parallel environments through matrix decomposition into blocks or submatrices. The Cartesian algorithm is characterized by requiring large amounts of RAM memory during its execution and does not make a significant use of network communications. Fox algorithm requires small RAM memory but increases communication times. For a detailed explanation of how these algorithms work is recommended to consult the work of Costa et al. [24] and references therein.

The experiments were performed on the teaching laboratories, Opteron and Storm pools. In the case of the Twister cluster, the experiments were performed with a single core on 4 nodes. The experiments with 16 cores, were carried out by using two cores of the 8 nodes of Twister.

The matrix multiplication was performed with native MPI and MPI over Condor in order to measure the Condor's overhead that is added on the execution of parallel applications with MPI.

Table 6 presents the results of computational experiments for matrix multiplication with cartesian and Fox parallel algorithms. It is worth noting that in table 6, MPI means that the job has been executed with native MPI only, and Condor with MPI over Condor.

Table 6 shows that the implementation of the Cartesian algorithm for matrix multiplication is the one that offers the lowest execution time. Fox's algorithm for matrix multiplication is characterized by requiring intensive network communication during its execution and does not make a significant use of the RAM memory of the nodes. The Twister and Storm clusters are connected through a Gigabit Ethernet switch, so in this case where the primary bottleneck is communication, these clusters have the best performance, as shown in Table 6.

When these algorithms are run on 4 cores, it is observed that increasing the matrix dimension, implies increasing the runtime. When these algorithms are run on 16 cores, as in the case of the Twister cluster or in the 16 cores of the teaching laboratories pool, a significant decrease in execution time is achieved for matrices of dimension 2048 and 4096, in comparison with the implementation of algorithms for the same dimensions in 4 cores. Note, the execution time increases notoriously for matrices order higher than 2048, possibly by the communication time between nodes.

Briefly, if memory usage is not a limitation, the Cartesian is preferred over the the Fox algorithm, because the former offers better performance and lower execution time with respect to the later. From Table 6 is clear that the overhead that Condor adds, concerning the execution of algorithms with native MPI, is

Table 6. Parallel execution time (seconds)

	4 cores - Networking laboratory				4 cores - Storm			
	Cartesian		Fox		Cartesian		Fox	
Matrix dimension	MPI	Condor	MPI	Condor	MPI	Condor	MPI	Condor
2048	35	45	137	148	25	33	112	119
4096	242	261	1130	1141	197	208	897	928
	4 cores - Opteron's cluster				4 cores - Twister			
	Cartesian		Fox		Cartesian		Fox	
Matrix dimension	MPI	Condor	MPI	Condor	MPI	Condor	MPI	Condor
2048	52	61	310	330	15	22	70	85
4096	368	379	2709	2760	113	133	871	908
	16 cores - Operating System and Networking Laboratories				16 cores - Twister			
	Cartesian		Fox		Cartesian		Fox	
Matrix dimension	MPI	Condor	MPI	Condor	MPI	Condor	MPI	Condor
2048	21	35	49	60	6	14	8	16
4096	133	169	314	350	39	50	130	139

negligible, except in the case that the runtime of the application itself is very small.

5.2 MC2 Meteorological Model

In order to test the mixed heterogeneous infrastructure built up with partially and/or fully dedicated infrastructures a research/production code is used. In this way one of the the main objectives of this paper can be discussed. For this purpose the the Canadian Mesoscale Compressible Community Model (MC2)is chosen.

The MC2 is a fully-compressible, nonhydrostatic, limited-area model capable of one-way self-nesting. MC2 equations are formulated based on the Euler equations (EQ) for a gas on a sphere. These equations are complemented by another set which incorporates the model physics that are processes not resolved explicitly such as: cumulus parametrization, boundary layer, land-surface interactions, radiation and microphysics.

The numerical model is based on a finite-difference scheme with semi-implicit and semi-Lagrangian approach. MC2 code has been parallelized to improve its performance. This is accomplished by subdiving the computational domain in the horizontal while keeping the vertical domain dimension unchanged. Each computational subdomain is distributed among different processors with a halo region (ghost nodes) containing information of the boundary subdomains of each neighbour processor. It is worth noting that the halo region extent in the vertical since no vertical subdivision has been made in the global computational domain, so that we could identify them as ghost fronts or slabs instead of ghost nodes. The

halo is updated every timestep by exchanging information between neighbour computational processors creating heavy network traffic.

As the computational domain can be divided in many ways (topology), different domain decompositions has been explored for a given amount of available CPUs. For instance given 12 nodes, different horizontal domain decompositions are possible namely: 12x1,2x6 and 3x4. All of these possibilities were tested and no major differences in the total integration time has been found. Interested readers are referred to [25] for a complete description of the MC2 dynamics and [26] for a complete description of the MC2 physics.

Only the results from the 600 m run are analyzed to understand the storm scale features. Following the initiation of convection, two storm systems developed. One system corresponds to a right moving supercell while the other is a left mover [27].

These systems are depicted as S1 and S2 in figure 3. S1 intensifies into a supercell and will be the focus of this study, whereas S2 fails to intensify and propagates toward the left of the mean wind (Fig. 3). The mechanisms leading to the formation of S1 and S2 are explained in [27].

Briefly, the development of vertical rotation in S1 and S2 occurs through the vertical tilting of the environmental horizontal vorticity, resulting in a pair of vortices. The presence of the downdraft splits the storm system into a cyclonic and an anticyclonic rotating pair. The interaction of the updrafts with the counter-clockwise rotating winds of the environment (see Fig. 3) favors the development of S1 and leads to the weakening of S2. A tornado forms at the tip of the hook-echo shape of storm S1.

MC2 application due to its inherent characteristics usually presents important communication times in comparison with total execution time. Consequentely the perfomance of Data Network affects the results. The experiments conducted discuss this issue.

The problem presented have been processed with dedicated clusters, partially dedicated Condor pool collected at teaching labs and mixed installations conformed by dedicated and no dedicated nodes. Clusters Twister, Opteron and Storm provides the resources for dedicated nodes and Teaching Labs infrastructure was chosen for non-dedicated nodes. A mixed infrastructure was built up adding resources from Storm cluster and Teaching Labs as well.

In Figure 4 a log-log graph of the computation times required to process the problem defined in Figure 3 can be observed. All the curves shows MPI over Condor CPU times, except for the case of mixed dedicated and non-dedicated resources where both plain MPI and MPI over condor CPU times are shown.

The results obtained using dedicated results are discussed in first place. From figure 4 can be observed that CPU times reduces as the number of nodes increases. Due to the number of nodes available from clusters a minimum of CPU time has not been reached in the different processes carried out for dedicated resources. While Storm and Opteron clusters have similar CPU capacity (see table 5) Opteron CPU time required to process the simulation is almost 3 times larger than the Storm's one. This result can be explained because Opteron cluster is

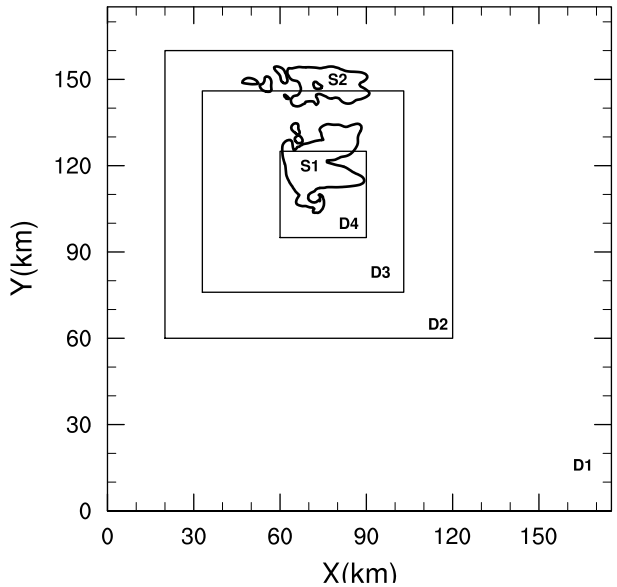

Fig. 3. The domains of the simulation. D1, D2, D3 and D4 correspond respectively to domains with grid size of 600 m, 200 m, 70 m and 30 m respectively. S1 (S2) is the right (left) moving storm.

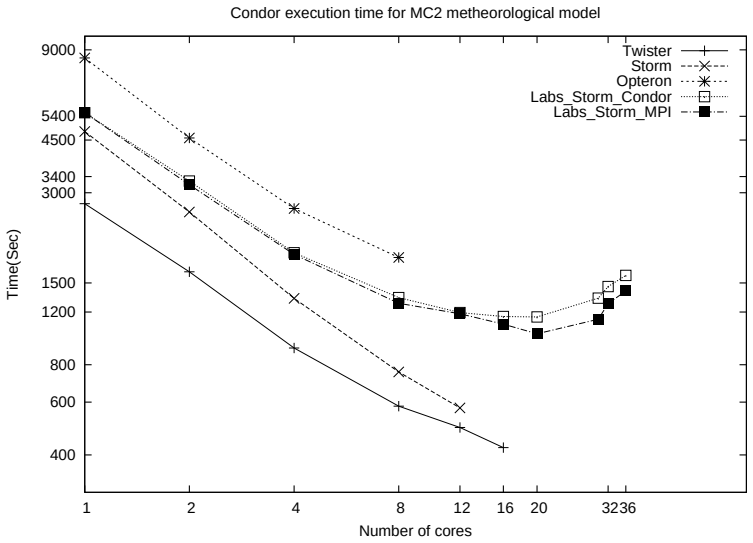

Fig. 4. Condor execution time for MC2 meteorological model

linked using a Fast Ethernet network and Storm is based on a Gigabit Ethernet one. Moreover, from the comparison of Twister and Storm results a difference of performance, according table 5, is not reached. Again in this case communication times are hiding the difference of CPU performance of these clusters. Condor's overhead has neither been shown in the graphics nor reported in tables, but no important values have been found.

The same problem has been processed using a mixed infrastructure collecting nodes from the teaching labs (non dedicated resources) and Storm cluster (dedicated ones. The obtained results are denoted like *L*abs_Storm_Condor for MPI over Condor and *L*abs_Storm_MPI for plain MPI studies respectively. The first 12 nodes were chosen from the labs pool in order to compare with storm results discussed in the paragraph above. The difference of CPU time shown in figure 4 in favor of Storm can be explained in this case in terms of network performance. It is important to note that in this case no practical difference can be observed from MPI results and MPI over Condor ones for the first 12 nodes. The remaining nodes have 12 to 36 cores chosen in a random way by Condor, so a true heterogeneous infrastructure is built up in this case. The minimum CPU time is reached for 16 nodes approximately. For a larger quantity of nodes CPU times increases because communication times are larger than processing ones. In this case overhead shows larger values than the above discussed results. A possible reason can be found in figure 2 that shows the infrastructure. Different VLAN has to be routed when resources from labs and Storm are mixed and latency is greater in this case. On the other hand resources are linked using different speed network, then it is expected that the slower network will degrade the total communication time.

6 Conclusions

In the paper the execution of MPI applications over a Condor environment has been studied.

In this sense different installation and configuration tasks required in order to execute the MPI applications with Condor over partially dedicated resources have been discussed. Some advantages of Condor in comparison with plain MPI processes at installation and configuration level have been pointed out as well.

From the results of the different examples processed both on dedicated (clusters), partially dedicated resources or mixed infrastructure can be said than Condor overhead does not represent a significant increase in execution times of parallel MPI applications with MPI. From the processed experiments can be observed that overhead is smaller for dedicated resources and remains bounded in this case. For partially dedicated resources the same conclusion can be stated. However for mixed heterogeneous infrastructure the overhead found is slightly larger. A possible explanation can be found in the different networks used (Fast Ethernet for Labs and Gigabit Ethernet for Storm). On the other hand different VLANs have to be routed and unavoidable latency is added as well.

It is important to point out that a production code like MC2 has been able to run in Condor managed infrastructure without important overhead. The larger CPU times obtained for MC2 in this case can be explained for the performance of the network for similar hardware.

The experiments carried out shown that besides other well known features of Condor MPI applications can be processed as well. In this way Distributed Computing infrastructure based on partially dedicated infrastructure like teaching labs or similar equipment can be a valuable tool in order to enhance dedicated clusters or even more important provide entry level High Performance Computing infrastructure.

7 Acknowledgments

The first author acknowledges the scholarship granted by the National University of Cuyo. The financial support provided by SECTyP projects 06/B194 and 06/M023 at National University of Cuyo and project PAE-PICT 2312 granted by the National Agency for Scientific and Technological Promotion is gratefully acknowledged.

References

[1] Thain D., Tannenbaum T., and Livny M.: Distributed Computing in Practice: The Condor Experience. Computer Sciences Department, University of Wisconsin-Madison, http://www.cs.wisc.edu/condor/publications.html, 2002.

[2] Livny M., Basney J., Raman R. and Tannenbaum T.: Mechanisms for High Throughput Computing. Computer Sciences Department, University of Wisconsin-Madison, http://www.cs.wisc.edu/condor/publications.html, 1997.

[3] Martínez P., Infraestructura para computacion de alta disponibilidad y administracion de recursos mediante Condor, Tesis de Maestría (en curso), Carrera de Maestría en Redes de Datos, Facultad de Informática, Universidad Nacional de La Plata.

[4] Martínez P., García Garino C., Catania C. and Monetti J.: Experiencias en computación de alta disponibilidad con el entorno Condor. In: II Encuentro de Investigadores y Docentes de Ingeniería. Desarrollos e Investigaciones Científico-Tecnológicos en Ingeniería. Maldonado, G., Veca, A. and Cremades, H. (eds.), Facultad Regional Mendoza, Universidad Tecnológica Nacional , Mendoza, pp. 157-163, 2007.

[5] Martínez P., Catania C., García Garino C. and Díaz J.: Reconocimiento de patrones de tráfico de red en un ambiente Condor. In: VIII Workshop de Procesamiento Distribuido y Paralelo, XIII Congreso Argentino de Ciencias de la Computación, Universidad del Nordeste, Corrientes, pp.1288-1299, 2007.

[6] C. A. Catania, C. Careglio, D. Monge, P. Martínez, A. Mirasso y C. García Garino: Estudios Paramétricos de Mecánica de Sólidos en Entornos de Computación Distribuida, Mecánica Computacional, Vol. 27, A. Cardona et al. (compiladores), AMCA, Santa Fe. Argentina, ISSN 1666-6070, pp. 1063-1084, 2008.

[7] Careglio C., Monge D., García Garino C. y Mirasso A.: Simulación numérica del ensayo de tracción simple en entornos de computación distribuida. In: V Encuentro de Investigadores y Docentes de Ingeniería. Desarrollos e Investigaciones Científico-Tecnológicos en Ingeniería. Facultad Regional Mendoza, Universidad Tecnológica Nacional, Los Reyunos, San Rafael, Mendoza, 2009.
[8] Martínez P., Catania C., Millán E., García Garino C. and Díaz J.: Procesamiento distribuido mediante el entorno Condor. Gestión de recursos total y parcialmente disponibles. In: V Encuentro de Investigadores y Docentes de Ingenieria. Desarrollos e Investigaciones Científico-Tecnológicos en Ingeniería. Facultad Regional Mendoza, Universidad Tecnológica Nacional, Los Reyunos, San Rafael, Mendoza, 2009.
[9] Lumb I., HPC Grid, Cap. 7 en Grid Computing: A practical Guide to Technology and Applications, Ahmar Abbas, Charles River Media, 2003.
[10] MPI Forum, http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf
[11] MPICH, http://www.mcs.anl.gov/research/projects/mpich2/
[12] Pacheco P. S.:Parallel Programming with MPI. Morgan Kaufmann D., Publishers, Inc. 1997
[13] Sanz A.: Condor. Manual de usuario para el cluster HERMES. Instituto de Investigación en ingeniería de Aragón, Universidad de Zaragoza, http://i3a.unizar.es/hermes/manual_hermes.pdf, 2006.
[14] The Globus Toolkit, Globus Toolkit 4.0.4, http://www.globus.org/toolkit/, 2010
[15] Mp2script, http://www.escience.cam.ac.uk/ mcal00/condor/mp2script.asc
[16] Rocks, http://www.rocksclusters.org/wordpress/
[17] OpenMPI, http://www.open-mpi.org/, 2010.
[18] Running MPICH jobs in Condor:
http://www.cs.wisc.edu/condor/manual/v6.1/2_9Running_MPICH.html
[19] PBS, Portable Batch System: http://www.pbsworks.com
[20] LSF, Load Sharing Facility :
http://www.platform.com/workload-management/high-perfomance-computing/lp
[21] Torque:
http://www.clusterresources.com/products/torque-resource-manager.php
[22] Mpiexec command, Ohio Supercomputer Center:
http://www.osc.edu/ djohnson/mpiexec/index.php
[23] Grama A., Karypis, G., Kumar V. and Gupta A., et.al.: Introduction to Parallel Computing. Addison-Wesley, 2003.
[24] Costa N., Catania C., García Garino C.,León O. and Silva M.: Cálculo de Productos Matriciales en Cluster Beowulf. II Encuentro de Investigadores y Docentes de Ingenieria. Desarrollos e Investigaciones Científico-Tecnológicos en Ingeniería. Rivera, S. and Núñez Mc Leod J. (eds.) Facultad de Ingeniería, Universidad Nacional de Cuyo, pp. 37-44, 2006.
[25] Benoit R., Desgagné M., Pellerin P., Pellerin S., Chartier Y. and Desjardins S.: The Canadian MC2: A semi-implicit semi-Lagrangian wide-band atmospheric model suited for fine-scale process studies and simulation. MWR, vol. 125, pp 2382-2415, 1997.
[26] Mailhot J., Belair S., Benoit R., Bilodeau B., Delage Y., Fillion L., Garand L., Girard C., and Tremblay A.: Scientific description of the RPN physics library - Version 3.6, Recherche en Prevision Numerique, Atmospheric Environment Service, Dorval, Quebec, pp. 188, 1998.
[27] Klemp, J. B. and Rotunno R.: Dynamics of tornadic thunderstorms. ARFM, vol. 19, pp 369-402, 1987.