Cloud Computing for Parameter Sweep Experiments[1]

Elina Pacini[1], Melisa Ribero[1,2], Cristian Mateos[3], Anibal Mirasso[2], Carlos García Garino[1,2]

[1] Instituto para las Tecnologías de la Información y las Comunicaciones (ITIC) – UNCuyo, Mendoza Argentina {epacini, cgarcia}@itu.uncu.edu.ar
[2] Facultad de Ingeniería, UNCuyo, Mendoza, Argentina, melisaribero@yahoo.com.ar, aemirasso@uncu.edu.ar
[3] ISISTAN - CONICET. Tandil, Buenos Aires, Argentina, cmateos@conicet.gov.ar

Abstract. Nowadays, scientists and engineers are more and more faced to the need of computational power to satisfy the ever-increasing resource intensive nature of their experiments. Traditionally, to cope with this, users have relied on conventional computing infrastructures such as clusters and Grids. A recent computing paradigm that is gaining momentum is Cloud Computing, which offers a simpler administration mechanism compared to those conventional infrastructures. However, there is a lack of studies in the literature about the viability of using Cloud Computing to execute scientific and engineering applications from a performance standpoint. We present an empirical study on the employment of Cloud infrastructures to run parameter sweep experiments (PSEs), particularly studies of viscoplastic solids together with simulations by using the CloudSim toolkit. In general, we obtained very good speedups, which suggest that disciplinary users could benefit from Cloud Computing for executing resource intensive PSEs.

Keywords: Parameter Sweep, Viscoplastic Solids, Cloud Computing

Introduction

Parameter Sweep Experiments, or PSEs for short, is a very popular way of conducting simulation-based experiments among scientists and engineers through which the same application code is run several times with different input parameters resulting in different outputs [1]. Representative examples of such kind of experiments are sensitivity studies of results in terms of defined parameter changes like is the case of imperfections in the simulation of simple tension test, or the study of buckling of imperfect columns.

From a purely software perspective, most PSEs are cluster friendly since individual inputs of an experiment can be handled by independent jobs. Therefore, using a software platform such as Condor [2], which is able to exploit the distributed nature of a computer cluster, allows these jobs to be run in parallel. In this way, not

1 This work is an extension of paper "E. Pacini et al., Simulation on Cloud Computing Infrastructures of Parametric Studies of Nonlinear Solids Problems" presented in ADNTIIC'2011

only PSEs execute faster, but also more computing intensive experiments can be computed, and hence more complex simulations can be performed. The same idea has been systematically applied to execute PSEs on Grid Computing [3], which are basically infrastructures that connect clusters via wide-area connections to increase computational power. To this end, software platforms designed to exploit Grids provide the illusion of the existence of a large supercomputer, which in turn virtualizes and combines the hardware capabilities of many much less powerful, geographically-dispersed machines to run resource intensive applications [4].

On the downside, for users not proficient in distributed technologies, manually configuring PSEs is tedious, time-consuming and error-prone. As a consequence, users typically waste precious time that could be instead invested into analyzing results. The availability of elaborated GUIs –specially for Grids– that help in automating an experimentation process has in part mitigated this problem. However, the highly complex nature of today's experiments and thus their associated computational cost greatly surpasses the time savings that can be delivered by this automation. Consequently, performance and particularly job scheduling becomes crucial. Broadly, job scheduling involves the recurrent problem of efficiently mapping a number of parallel jobs to available computing nodes in a distributed environment.

A recent distributed computing paradigm that is rapidly gaining momentum is Cloud Computing [4,5,6], which bases on the idea of providing an on demand computing infrastructure to end users. Typically, users exploit Clouds by requesting from them one or more *machine images*, which are virtual machines running a desired operating system on top of several physical machines (e.g. a datacenter). Interaction with a Cloud is performed by using Cloud services, which define the functional capabilities of a Cloud, i.e. machine image management, access to software/data, security, and so on. Among the benefits of Cloud Computing is precisely a simplified configuration and deployment model compared to clusters and Grids, which is extremely desirable for disciplinary users. However, even when Cloud infrastructures intuitively have the capabilities to deliver good performance, very few detailed studies about the speedups achieved for PSEs have been reported in the literature.

In this work, we will show the benefits of Cloud Computing for executing PSEs through a case study. The application domain under study involves PSEs of viscoplastic solids [7], which explore the sensitivity of viscoplastic solid behavior in terms of changes of certain model parameters (viscosity parameter η, sensitivity coefficient, and so on). In this sense parametric studies previously discussed for imperfections [8] are extended for material parameters case, which were computed on Clouds by using the CloudSim simulation toolkit [9]. Overall, although they cannot be generalized, results show that by executing our experiments in our simulated Clouds, depending on the configured computational capabilities and the scheduling policy being used, near-to-ideal speedups can be obtained.

The rest of the chapter is organized as follows. The next Section provides more details on Cloud Computing and the motivation behind considering this distributed computing paradigm for executing PSEs. The Section also explains CloudSim, the simulation toolkit used during the experiments. Section 3 describes our case study, which involves a parametric study of nonlinear solids problems. Later, Section 4

presents the results obtained from processing these problems on Cloud Computing. Finally, Section 5 concludes the chapter and describes prospective future works.

Background

Running Parameter Sweep Experiments (PSE) [1,10] involves many independent jobs, since the experiments are executed under multiple initial configurations (input parameter values) several times, to locate a particular point in the parameter space that satisfies certain criteria. In addition, different PSEs have different number of parameters. PSEs provide scientific value to a high throughput computing process along with relative ease. Interestingly, PSEs find their application in diverse scientific areas like Bioinformatics [11], Earth Sciences [12], High-Energy Physics [13], Molecular Science [14] and even Social Sciences [15].

When designing PSEs there are several issues to tackle. On one hand, it is necessary to generate all possible combinations of input parameters, which is a time-consuming task and should be automated. Besides, it is not straightforward to provide a general solution, since each problem has a different number of parameters and each of them has its own variation interval. Another issue, which is in part a consequence of the first issue, relates to scheduling PSEs on distributed environments, which is a complex activity. For this reason, it is necessary to develop efficient scheduling strategies to appropriately allocate the workload and reduce the computation time.

In recent years Grid Computing [3] and Cloud Computing technologies [5,6] have been increasingly used for running such applications. PSEs are well suited for these environments since they are inherently parallel problems with no or little data transfer between nodes during computations. Since many applications require a great need for calculation, these applications have been initially addressed to dedicated High-Throughput Computing (HTC) infrastructures such as clusters or pools of networked machines, managed by some software such as Condor [2]. Then, with the advent of Grid Computing [3] new opportunities were available to scientists, since Grids offered the computational power required to perform large experiments. Grid Computing introduced new facilities such as dynamic service discovery, the ability of relying on a large number of resources belonging to different administrative domains, and finding the best set of machines that meet an application's requirements. The use of Grid Computing in scientific applications [16] has been successful in many international projects and has led to the establishment of world-wide infrastructures available for computational science [17,18,19].

Despite the widespread use of Grid technologies in scientific computing, as demonstrated by the large amount of projects served by Grid Computing, some issues still make the access to this technology not easy for disciplinary or domain users. For example, operationally some Computational Grids are bureaucratic, since research groups have to submit a proposal describing the type of research they want to carry out prior to executing their experiments. This approach leads to a competitive use of scientific Grids, and minor research projects cannot get access to them.

Other usage-related issues involve technical hurdles. In most cases scientific Grids feature a prepackaged environment in which applications will be executed. Then, specific tools/APIs have to be used, and there could be limitations on the hosting operating systems or the services offered by the runtime environment. On the other hand, although Grid Computing favors dynamic resource discovery and provision of a wide variety of runtime environments for applications, in practice, a limited set of options are available for scientists, which are not in addition elastic enough to cover their needs. A practical example, involves the use of specific software that could not be available in the runtime environment were applications are executed. In general, applications that run on scientific Grids are implemented as bag of job applications, workflows, and MPI (Message Passing Interface) [20] parallel processes. Some scientific experiments could not fit into these models and therefore have to be redesigned to exploit a particular scientific Grid.

Cloud Computing: Overview

All in all, while the aforementioned bureaucratic issues can be a minor problem, the technical ones could constitute a fundamental obstacle for next generation scientific computing. Cloud Computing [5,6], the current emerging trend in delivering IT services, has been recently proposed to address the aforementioned problems. By means of virtualization technologies, Cloud Computing offers to end users a variety of services covering the entire computing stack, from the hardware to the application level, by charging them on a pay per use basis. This makes the spectrum of options available to scientists, and particularly PSEs users, wide enough to cover any specific need from their research. Another important feature, from which scientists can benefit, is the ability to scale up and down the computing infrastructure according to the application requirements and the budget of users. By using Cloud-based technologies scientists can have easy access to large distributed infrastructures and are allowed to completely customize their execution environment, thus deploying the most appropriate setup for their experiments. Moreover, by renting the infrastructure on a pay per use basis, they can have immediate access to required resources without any capacity planning and they are free to release resources when these latter are no longer needed.

As suggested, central to Cloud computing is the concept of virtualization, i.e. the capability of a software system of emulating various operating systems. By means of this support, scientists can exploit Clouds by requesting from them *machine images*, or virtual machines that emulate any operating system on top of several physical machines, which in turn run a host operating system. Usually, Clouds are established using the machines of a datacenter for executing user applications while they are idle.

Interaction with a Cloud environment is performed via Cloud services [5], which define the functional capabilities of a Cloud, i.e. machine image management, access to software/data, security, and so forth. Cloud services are commonly exposed to the outer world via Web Services [21], i.e. software components that can be remotely invoked by any application. By using these services, a user application can allocate machine images, upload input data, execute, and download output (result) data for

further analysis. Finally, to offer on demand, shared access to their underlying physical resources, Clouds dynamically allocate and deallocate machines images. Besides, and also important, Clouds can coallocate N machines images on M physical machines, with N ≥ M, thus concurrent user-wide resource sharing is ensured. These relationships are depicted in Fig. 1.

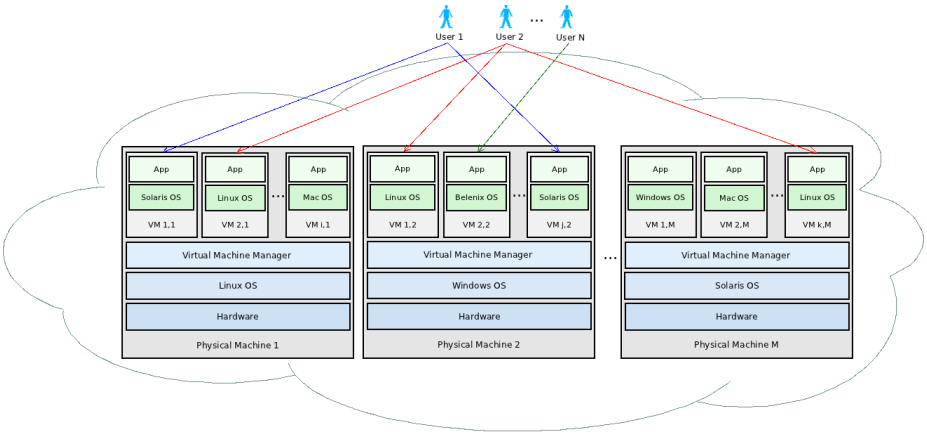


Fig. 1. Cloud Computing: High-level view

In summary, a Cloud gives users the illusion of a single, powerful computer in which complex applications can be run. Besides, the software stack of the infrastructure can be fully adapted and configured according to user's needs. This provides excellent opportunities for scientists and engineers to run applications that demand by nature a huge amount of computational resources –i.e. CPU cycles, memory and storage– and rely on specific software libraries.

With everything mentioned so far, we can say that from the perspective of domain scientists, the complexity of traditional distributed and parallel computing environments such as clusters and Grids should be hidden so that domain scientists can focus on their main concern, which is performing their experiments. As a result, the use of Cloud Computing infrastructures is a good choice for running scientific applications. Precisely, for parametric studies such as the one presented in this work, or scientific applications in general, the value of Cloud Computing as a tool to execute complex applications has been already recognized within the scientific community [22,23].

The CloudSim Toolkit: Simulation of Cloud Computing environments

CloudSim [9] is an extensible simulation toolkit that enables modeling, simulation and experimentation of Cloud Computing infrastructures and application provisioning environments. CloudSim supports both system and behavior modeling of Cloud system components such as data centers, virtual machines (VMs) and resource provisioning policies. A virtual machine (VM) is a software implementation of a

machine (i.e. a computer) that executes programs like a physical machine. By using CloudSim, researchers and developers can focus on specific system design issues that they want to investigate, without getting concerned about the low level details related to Cloud-based infrastructures and services. This is desirable as intuitively putting a real Cloud to work demands much administration effort.

CloudSim offers support for modeling and simulation of large scale Cloud Computing infrastructures, including data centers on a single physical computing node. Besides, CloudSim provides a self-contained platform for modeling data centers, service brokers, scheduling, and allocations policies. In addition, CloudSim lets users to easily switch between space-shared and time-shared allocation of both processing elements (PEs) and jobs to virtualized services.

The core hardware infrastructure services related to Clouds are modeled by a Datacenter component for handling service requests. A Datacenter is composed by a set of hosts that are responsible for managing VMs during their life cycle. Host is a component that represents a physical computing node in a Cloud, and as such is assigned a pre-configured processing capability, memory, storage, and scheduling policy for allocating processing elements (PEs) to VMs.

CloudSim supports scheduling policies at the host level and at the VM level. At the host level it is possible to specify how much of the overall processing power of each PE in a host will be assigned to each VM. At the VM level, the VMs assign a specific amount of the available processing power to individual jobs units -called cloudlet by CloudSim- that are hosted within its execution engine. At each level, CloudSim implements the *time-shared* and *space-shared* allocation policies.

When employing the *space-shared* policy only one VM can be running at a given instance of time, this policy takes into account how many processing cores will be delegated to each VM, and how much of the processing core's capacity will effectively be attributed for a given VM. So, it is possible to assign specific CPU cores to specific VMs. The same happens for provisioning cloudlets within a VM, since each cloudlet demands only one PE. If there are other cloudlets ready to run at the same time, they have to wait in the run queue. The estimated start time depends on the position of the cloudlet in the execution queue, because the processing unit is used exclusively by one cloudlet. With the space-shared policy CloudSim processes jobs in first come first serve basis. This is the sequence in which cloudlets are sent to the VMs by the *broker*. The broker models a high-level software component that controls which cloudlet should be sent to which VM and in what sequence. Last but not least, with the *time-shared* policy, the processing power of hosts is concurrently shared by the VMs. Therefore, multiple cloudlets can simultaneously multi-task within the same VM. With this policy, there are no queuing delays associated with cloudlets.

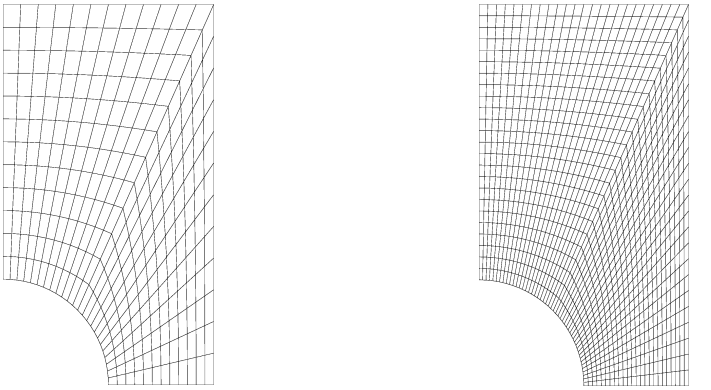
Case Study: A PSE for nonlinear solids problems

In order to assess the effectiveness of Cloud Computing environments for executing PSEs, we have processed a real experiment by using different Cloud infrastructures simulated via CloudSim toolkit. The case study chosen is the problem proposed in

[24], in which a plane strain plate with a central circular hole is studied. The dimensions of the plate are 18 x 10 m, R = 5 m. Material constants considered are E = 2.1 10^5 Mpa; ν = 0:3; σ_y = 240 Mpa; H = 0. A Perzyna viscoplastic model with m = 1 and n = ∞ is considered. The large strain elasto/viscoplastic Finite Element code SOGDE is used in this study. References to SOGDE can be seen in the works of [25,26] and application problems simulated with the code can be found in [27]. A detailed presentation of viscoplastic theory, numerical implementation and examples can be found in the works [7,28].

We have previously studied parametric problems where a geometry parameter of imperfection was chosen [8]. In this case a constitutive material coefficient is selected as a parameter. In order to do that different viscosity values of η parameter are considered: 1.10^4, 2.10^4, 3.10^4, 4.10^4, 5.10^4, 7.10^4, 1.10^5, 2.10^5, 3.10^5, 4.10^5, 5.10^5, 7.10^5, 1.10^6, 2.10^6, 3.10^6, 4.10^6, 5.10^6, 7.10^6, 1.10^7, 2.10^7, 3.10^7, 4.10^7, 5.10^7, $7.10^7$and 1.10^8 Mpas.

The two finite element meshes displayed in Fig. 2 were tested. The first one has 288 elements and the second mesh has 1,152 elements. In both cases Q1/P0 elements are chosen. Imposed displacements (at y=18m) are applied until a final displacement of 2000 mm is reached in 400 equals time steps of 0.05 mm each one. For all the time steps Δt = 1 has been set. Large strain effects are considered in all experiments simulated.



(a) 288 elements (b) 1,152 elements

Fig. 2. Finite element meshes using during our study

Experimental Results

This section presents the results obtained from our experimental study. The experiments aim to demonstrate and evaluate the viability of using Cloud Computing to perform parameter sweep experiments such as the one explained in the previous section. As mentioned earlier, to carry out these experiments, we have used CloudSim toolkit.

The methodology followed to carry out the experiments was involved two steps. First, in a single machine we run the PSE of the previous section by varying the viscosity parameter η and measuring the execution time for 25 different experiments (resulting in 25 input files with different input configurations) and two different meshes (one of 288 elements and other comprising 1,152 elements). The PSE were solved using the SOGDE solver. The characteristics of the machine on which the experiments were carried out are shown in Table 1. The machine model is AMD Athlon(tm) 64 X2 Dual Core Processor 3600+, equipped with the Linux operating system (specifically an Ubuntu 11.04 distribution) running the generic kernel version 2.6.38-8.

The obtained real information (execution times, input/output file sizes) was then used to feed CloudSim. The information regarding processing power was obtained from the benchmarking support of Linux and as such is expressed in bogomips [29]. Bogomips (from bogus and MIPS), is a metric used by Linux operating systems that indicates how fast a computer processor runs. Since the real tests were performed on a machine running the Linux operating system, we have considered using the bogomips measure which is as we mentioned the one used by this operating system to approximate CPU power.

Table 1**.** Machine used to execute the PSE

Feature	Characteristic
CPU power	4,008.64 bogoMIPS
Number of CPUs	2
RAM memory	2 Gbytes
Storage size	400 Gbytes
Bandwidth	100 Mbps

Furthermore, in a second step, we performed a number of simulations involving executing the PSE on Cloud infrastructures by using CloudSim. The simulations have been carried out by taking into account the bogomips metric. This is, once the execution times have been obtained from the real machine, we calculated for each experiment the number of executed instructions by the following formula:

$$NI_i = bogomipsCPU * T_i \quad (1)$$

where,

- NI_i is the number of million instructions to be executed by or associated to a job i
- *bogomipsCPU* is the processing power of our real machine measured in bogomips
- T_i is the time that took to run a job *i* on the real machine

Here is an example of how to calculate the number of instructions of a job that took 117 seconds to be executed. The machine where the experiment was executed has a processing power of 4,008.64 bogomips. Then, the resulting number of instructions for this experiment was 469,011 MI (Million Instructions). CloudSim was

configured as a data center composed of a single machine –or "host" in CloudSim terminology– with the same characteristics as the real machine where the experiments were performed. The characteristics of the configured host are shown in Table 2.

Processing power is expressed in MIPS (Million Instructions Per Second), RAM memory and Storage capacity are in MBytes, bandwidth in Mbps, and finally, PE is the number of processing elements (CPUs/cores) of a host. Each PE had the same processing power.

Table 2**.** Host characteristics

Host Parameters	Value
Processing Power	4,008
RAM	4,096
Storage	409,600
Bandwidth	100
PE	2

Once configured, we checked that the execution times obtained by the simulation coincided or were close to real times for each independent job performed on the real machine. The results were successful in the sense that one experiment (i.e. a variation in the value of η) took 117 seconds to be solved in the real machine, while in the simulated machine the elapsed time was 117.02 seconds. Once the execution times have been validated for a single machine on CloudSim, a new simulation scenario was set, which consisted of one datacenter with 10 hosts, each with the same hardware capabilities as the real single machine, and 40 VMs, each with the characteristics specified in Table 3. A summary of this simulation scenario is shown in Table 4.

Table 3. Virtual Machine characteristics

VM Parameters	Value
MIPS	4,008
RAM	1,024
Image Size	102,400
Bandwidth	25
PE	1
Vmm	Xen

Table 4. CloudSim simulator configuration used in the experiments

Parameter	Value
Number of Hosts	10
Number of VMs	40
Number of Cloudlets	from 25 to 250

With this new scenario, we performed several experiments to evaluate the performance of our PSE in a simulated Cloud Computing environment as we increase the number of jobs to be performed, i.e. 25 * i jobs with i = 1, 2, …, 10. This is, a base subset comprising 25 jobs was obtained by varying the value of η, while the extra jobs were obtained not by further varying this value but cloning the base subset. The reason of this was to stress the various experimental Cloud scenarios.

Each job, called cloudlet by CloudSim, had the characteristics shown in Table 5, where Length parameter is the number of instructions to be executed by a cloudlet in MI (Million Instructions), which varied between 52,112 and 104,225 MI for the mesh of 288 elements and between 244,127 and 469,011 for the mesh of 1,152 elements. Moreover, PE is the number of processing elements required to perform a job (CPUs). FileSize and OutputSize are the input file size and output file size in bytes, respectively. As it is shown in Table 5 the experiments corresponding to the mesh of 288 elements had input files of 40,038 bytes, and the experiments corresponding to the mesh of 1,152 elements of 93,082 bytes. A similar distinction applies to the sizes of output files.

Table 5. Cloudlet configuration used in the experiments

Cloudlet parameters	Value
Length	Between 52,112 and 104,225 (mesh of 288 elements) Between 244,127 and 469,011 (mesh of 1,152 elements)
PEs	1
FileSize	40,038 bytes (mesh of 288 elements) 93,082 bytes (mesh of 1,152 elements)
OutputSize	722,432 bytes (mesh of 288 elements) 2,202,010 bytes (mesh of 1,152 elements)

To perform the simulation we have considered, on one hand, that PSE cloudlets have similar processing times. The processing times are similar because both input and output files have the same size. Here, a job corresponds to execute an instance of the parametric study of viscoplastic solid. On the other hand, the goal is to assign jobs to Cloud hosts so that the total completion time, also known as makespan, is minimized. Finally, the order in which cloudlets are processed on a particular host is not relevant, since we assume they are completely independent, or in other words, cloudlets do not cooperate to perform the same computation and do not share data.

In CloudSim, the amount of available hardware resources to each VM is constrained by the total processing power and system bandwidth available within the associated host. Therefore, scheduling policies must be applied in order to assign the VMs to the host and get a maximum use of resources. On the other hand, cloudlets must also be scheduled with some scheduling policy for a maximum resource performance and minimize execution times.

In the next subsections we report on the obtained results when executing the 25 experiments of our PSE by combining the scheduling policies described in subsection 2.2. In addition, we have considered two different scenarios to evaluate the performance of our experiments and consequently we have used two types of environments, i.e. homogeneous and heterogeneous, which are explained below.

Homogeneous resources experiments

In this subsection we analyze how each scheduling policy responds when Cloud hosts and VMs follow the specifications described in Table 2 and 3.

1.1.1 Space-shared provisioning for VMs and jobs

The provisioning scenario where the space-shared policy is applied for both VMs and jobs (cloudlets) can be seen in Fig. 3. Here, the makespan of the whole cloudlets is shown.

The makespan has shown a linear growth with respect to the increasing number of cloudlets. After the creation of VMs, cloudlets were incrementally sent to VMs in groups of 25 to measure the makespan as we increase the workload on the VMs. The makespan rose from 38.21 seconds to 263.54 seconds when the number of cloudlets increased from 25 to 250 and using the mesh of 288 elements (curve in red). The makespan rose from 160.25 seconds to 1,000.86 seconds when the mesh used has 1,152 elements (curve in blue).

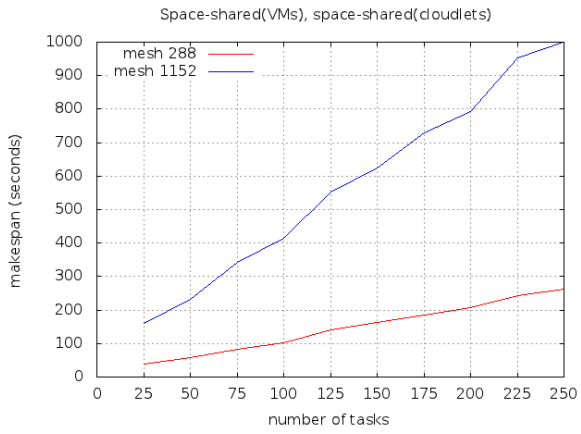


Fig. 3. Space-shared provisioning for VMs and jobs: Results

As each VM requires one PE for processing (see Table 3), with the space-shared policy only two VMs can actually run in a host at a given instant of time, because each host has two PEs as it is shown in Table 2. Therefore, given a scenario consisting of a total of 10 hosts and 40 VMs, at a given instant of time may be assigned 20 VMs to the hosts, i.e. one VM by each PE using space-shared policy, and the rest of the VMs can be assigned once the former set complete their execution.

As the number of PSEs and hence cloudlets in regard to the available amount of resources increases, the estimated start time of each cloudlet depends on the position of the cloudlet in the execution queue, since each PE is used exclusively by one cloudlet under the space-shared policy. Remaining cloudlets are queued when there are not free processing elements that can be used for execution.

The progress of execution times corresponding to the mesh of 288 elements when we sent to execute a group of 150 cloudlets, i.e. a group of 150 parameter sweep experiments as described in the previous section can be seen in Fig. 4 (curve in red). Since, under this policy, each cloudlet had its own dedicated processing element, the queue size (cloudlets waiting to be run) did not affect execution time of individual cloudlets. As can be seen in the curve, the execution times were increasing linearly approximately every 20 jobs. This is because as mentioned above, only 20 VMs were created with the space-shared policy, so the cloudlets are sent to the VMs to run in groups of 20 until they finish their execution. When the first submitted group of cloudlets finishes their execution, 20 more are sent, and so on until all cloudlets are executed.

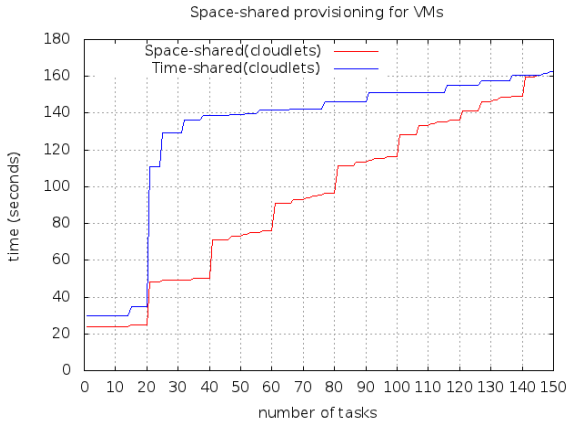


Fig. 4. Space-shared provisioning for VMs, space-shared and time-shared provisioning for jobs (mesh of 288 elements): Results

1.1.2 Space-shared provisioning for VMs and time-shared provisioning for jobs

This scenario is shown in Fig. 5. Here, a space-shared policy is applied for allocating VMs to hosts and time-shared policy for allocating job units to processing core within a VM. When time-shared policy is used by VMs, during a VM lifetime all the cloudlets assigned to it are dynamically context switched during their lifecycle.

The number of cloudlets to be performed again ranged from 25 to 250. The makespan for the mesh of 288 elements (curve in red) was 38.21 seconds and 263.54 seconds when the number of cloudlets was equal to 25 and 250, respectively. The makespan for the mesh of 1,152 elements was 160.25 seconds and 1,000.86 seconds when the number of cloudlets increased from 25 to 250.

When employing the space-shared policy to allocate the VMs to hosts, only two VMs can be run simultaneously (one VM per PE). Thus, only 20 VMs were assigned to the 10 hosts. The same case was explained in the previous subsection where the same policy was used to assign the VMs to host. In this scenario, when we used time-shared policy to assign the cloudlets to VMs, the execution time of each cloudlet varied as the number of submitted cloudlets to run increased. By using this combination of policies, execution time was significantly and negatively affected because the processing elements were concurrently context switched among the list of scheduled jobs.

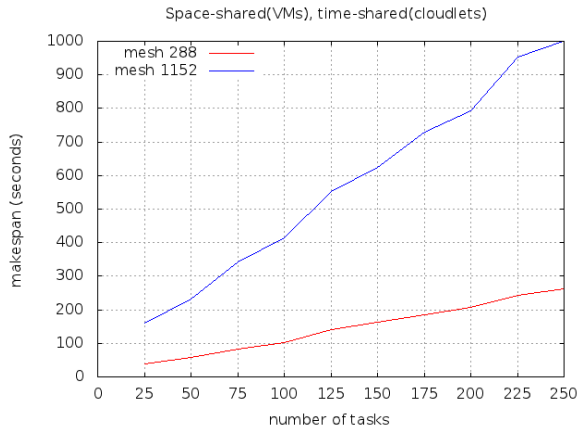


Fig. 5. Space-shared provisioning for VMs and time-shared provisioning for jobs: Results

The progress of execution times corresponding to the mesh of 288 elements when we sent to execute a group of 150 cloudlets can be seen in Fig. 4 (curve in blue). The first group of 20 cloudlets was able to complete earlier than the other ones because in this case the hosts were not overloaded at the beginning of execution. The other

cloudlets took more time to completion because were exchanged among the remaining processing elements to finish their execution. In the end, as more cloudlets reached completion, comparatively more hosts became available for allocation.

1.1.3 Time-shared provisioning for VMs and space-shared provisioning for jobs

The provisioning scenario where the time-shared policy is applied for allocating VMs to hosts and space-shared policy is applied for allocating jobs (cloudlets) can be seen in Fig. 6.

The number of cloudlets to be performed was again in the range of 25-250. In this scenario, the makespan for the mesh of 288 elements (curve in red) was 62.51 seconds and 339.17 seconds when the number of cloudlets was equal to 25 and 250, respectively. When we carried out the experiments with the mesh of 1,152 elements (curve in blue), the makespan was 280.94 seconds and 1,328.13 seconds when the number of cloudlets was equal to 25 and 250.

In this scenario, each VM receives a time slice on each processing element, and then distributes the slices among cloudlets on a space-shared policy. As the PEs are shared, the amount of processing power available to an individual VM is not constant. Given that, cloudlets are assigned based on a space-shared policy, which means that at any given instance of time only one cloudlet can be actively using a processing element.

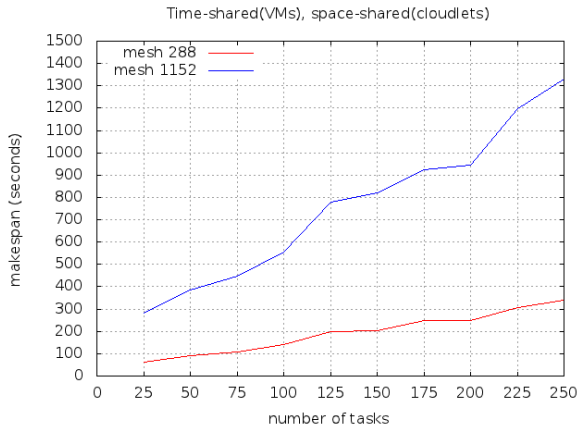


Fig. 6. Time-shared provisioning for VMs and space-shared provisioning for jobs: Results

The progress (curve in red) of execution times when we sent to execute a group of 150 cloudlets, or a group of 150 parameter sweep experiments as described in section 3 can be seen in Fig. 7. Since using the time-shared policy in the hosts the processing power available is concurrently shared by VMs, here 40 VMs have been created in the 10 available hosts. As can be seen in the curve, the execution times were increasing gradually over the first 50 cloudlets. The 100 remaining jobs took

considerably longer than the 50 first jobs. This latter effect occurs because the VMs available for processing within hosts begun to be switched between their PEs, which consumes time.

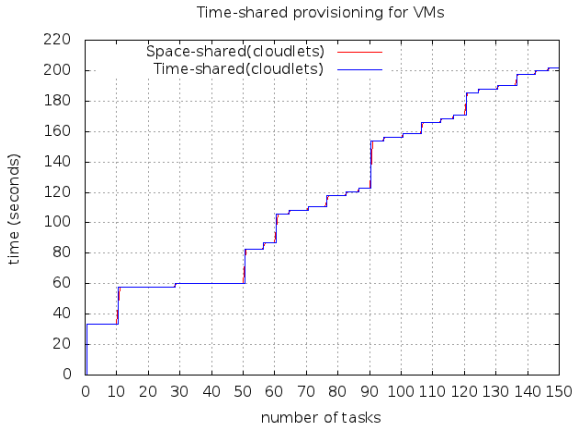


Fig. 7. Time-shared provisioning for VMs, space-shared and time-shared provisioning for jobs (mesh of 288 elements): Results

1.1.4 Time-shared provisioning for VMs and jobs

In this scenario a time-shared allocation is applied for both VMs and job units. Fig. 8 shows the makespan from this scenario as the number of cloudlets increases from 25 to 250. When the time-shared policy is used, the processing power within a host is concurrently shared by its associated VMs and the PEs of each VM are simultaneously divided among its cloudlets. As a consequence, in this scenario, there are no queuing delays associated with job units. CloudSim assumes that all the computing power of PEs is available for VMs and cloudlets, and it is divided equally among them.

In this scenario, the makespan for the mesh of 288 elements was 62.51, 91.42, 108.22, 139.53, 199.64, 202.05, 247.76, 247.75, 307.89 and 339.17 seconds when the number of cloudlets was increased from 25 to 250 by groups of 25 cloudlets (see Fig.8, curve in red). On the other hand, the makespan for the mesh of 1,152 elements (curve in blue) was 280.94, 384.3, 449.11, 557.2, 780.55, 821.43, 922.3, 946.3, 1,198.52 and 1,328.13 seconds.

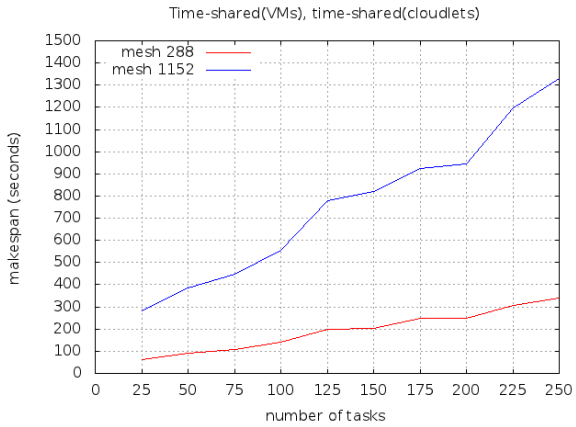


Fig. 8. Time-shared provisioning for VMs and jobs: Results

The progress (curve in blue) of execution times when we sent to execute a group of 150 cloudlets is illustrated in Fig. 7. As the reader can see, the execution times are consistent with that of the scenario of the previous subsection. Since there is a greater number of VMs actually running at any time, it was not necessary to interchange the jobs between the VMs. Whenever a cloudlet is ready to be executed, there is a machine available to use.

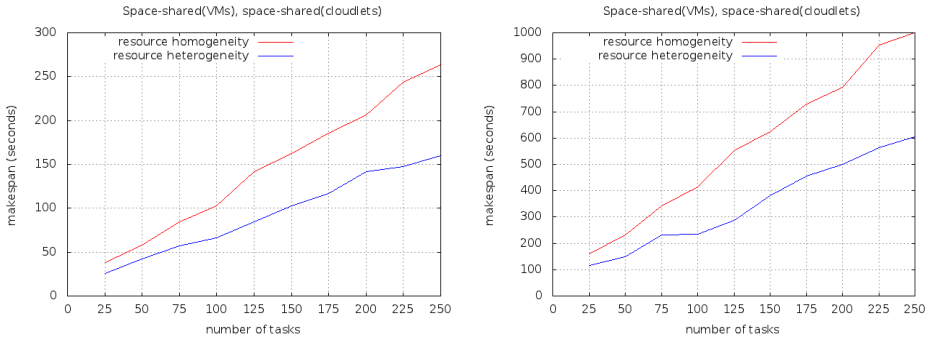
Heterogeneous resources experiments

In this subsection we analyze how each scheduling policy responds when using a Cloud with heterogeneous hosts. To analyze the performance of the scheduling algorithms, one characteristic that is of importance in real world scenarios is how the algorithms perform in the presence of resource heterogeneity. In this analysis, we have considered hosts with a random number of PEs between 1 and 6, while the other specifications are the same shown in Tables 2 and 3. Until now, each VM had only one PE. Next, we discuss the same scenarios of the previous section, and perform a comparison of job assignment with respect to homogeneous and heterogeneous infrastructures.

1.1.5 Space-shared provisioning for VMs and jobs

The provisioning scenario where the space-shared policy is applied for both VMs and cloudlets can be seen in Fig. 9. After the creation of VMs with a random number of PEs, cloudlets were incrementally sent to VMs in groups of 25 to measure the makespan as the workload on the VMs increased. The number of cloudlets to be performed ranges from 25 to 250 as in the previous subsection. The allocations of cloudlets to heterogeneous resources are illustrated in Fig. 9a and Fig. 9b by the curve

in blue. The red curve shows the same scenario that was discussed in subsection 4.1.1 for the case of homogeneous resources.



(a) Mesh of 288 elements (b) Mesh of 1,152 elements

Fig. 9. Space-shared provisioning for VMs and jobs using resource heterogeneity: Results

Due to the fact that in this scenario the entire Cloud had more number of PEs available to run the experiments (between 1 and 6 per resource), runtimes were reduced significantly with respect to the homogeneous scenario. The makespan of the first group of 25 cloudlets -corresponding to the mesh of 288 elements- was very close to the makespan of the homogeneous scenario (see Fig. 9a). This makespan was 38.21 seconds for the homogeneous scenario and 26.1 seconds when using heterogeneous resources. Here, makespan is close because in the worst case (homogeneous scenario) the number of PEs available to execute the cloudlets is nearby to the number of executed cloudlets (20 VMs to execute 25 cloudlets). Then, each cloudlet is executed in one PEs until the former finishes. For the following groups of cloudlets –between 50 and 250- the makespan was always lower when using heterogeneous resources. When we sent 250 cloudlets the makespan was 159.89 seconds.

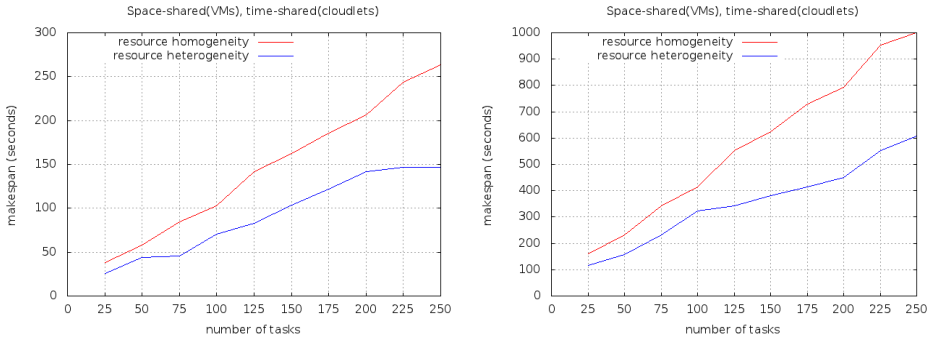
The same heterogeneous scenario when we using the mesh of 1,152 elements is illustrated in Fig. 9b (curve in blue). Here the makespan was 117.12, 150.24, 231.38, 235.43, 288.54, 380.74, 454.94, 499, 563.07, and 604.22 seconds when the number of cloudlets was increased from 25 to 250 by groups of 25 cloudlets.

1.1.6 Space-shared provisioning for VMs and time-shared provisioning for jobs

This subsection presents an heterogeneous scenario where the space-shared policy is applied for allocating VMs to hosts and the time-shared policy is used for allocating jobs to processing elements within a VM. Fig. 10 illustrates the makespan when the number of cloudlets is increased from 25 to 250 and compares the results with the homogeneous scenario. Here, the blue curve represents the makespan of cloudlets when heterogeneous resources are used.

The curve in red illustrates the makespan for homogeneous resources. When we compared the makespan, we obtained that the execution times for each group of

cloudlets submitted to run was greatly improved when relying on heterogeneity. This improvement was because there was more PEs available to execute each group of jobs.



(a) Mesh of 288 elements (b) Mesh of 1,152 elements

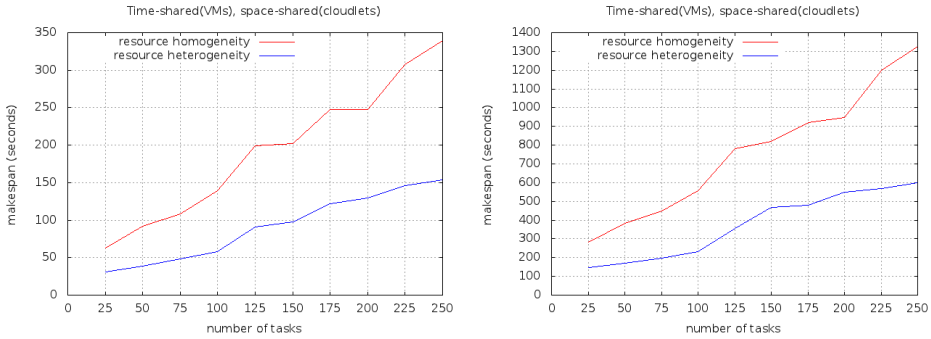
Fig. 10. Space-shared provisioning for VMs and time-shared provisioning for jobs using resource heterogeneity: Results

In this heterogeneous scenario, for the mesh of 288 elements (see Fig. 10a) the makespan was 26.1 seconds and 146.82 seconds when the number of cloudlets was equal to 25 and 250, respectively. The experiments corresponding to the mesh of 1,152 elements is illustrated in Fig. 10b. The makespan was 117.12, 157.21, 231.38, 324.52, 342.74, 381.91, 414.94, 451.04, 551.88 and 606.57 seconds when the number of cloudlets was increased from 25 to 250 by groups of 25 cloudlets.

1.1.7 Time-shared provisioning for VMs and space-shared provisioning for jobs

An heterogeneous experimental scenario where the time-shared policy is applied for allocating VMs to hosts and the space-shared policy is used for allocating jobs (cloudlets) to PEs within a VM can be seen in Fig. 11.

The number of cloudlets to be computed was again in the range 25-250. The makespan for the case of heterogeneous resources is illustrated in blue and the makespan when we used homogeneous resources was colored in red.



(a) Mesh of 288 elements (b) Mesh of 1,152 elements

Fig. 11. Time-shared provisioning for VMs and space-shared provisioning for jobs using resource heterogeneity: Results

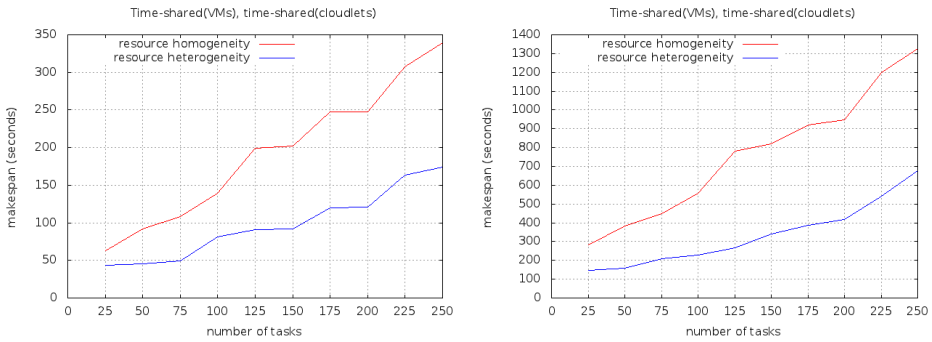
As it is shown in the Fig. 11a and Fig. 11b, both curves show similar behavior. This is because we used the same allocation policies. The way the schedulers decided to assign the VMs to hosts and the cloudlets to VMs took into account the same considerations. This means that each VM receives a time slice on each processing element in the host, and then distributes the slices among cloudlets on a space-shared policy. The makespan for the mesh of 288 elements was 31.35 seconds and 154.15 seconds when the number of cloudlets was equal to 25 and 250. When we carried out the experiments with the mesh of 1,152 elements, the makespan was 146.37 seconds and 599.05 seconds when the number of cloudlets was equal to 25 and 250.

1.1.8 Time-shared provisioning for VMs and jobs

In this subsection we present the results for a heterogeneous scenario where the time-shared policy is applied. The progress (curve in blue) of execution times as the number of cloudlets increase from 25 to 250 is illustrated in Fig 12.

In this heterogeneous scenario, the makespan was 146.37 seconds and 677.64 seconds when the number of cloudlets was equal to 25 and 250, respectively. In the figure, we have performed the comparison with the scenario composed of homogeneous resources (curve in red). Overall, we obtained that, with this heterogeneous scenario, the makespan for the entire set of cloudlets was significantly reduced.

In this heterogeneous scenario, the makespan for the mesh of 288 elements illustrated in Fig. 12a was 43.88, 45.17, 49.05, 81.22, 90.6, 91.55, 119.93, 121.07, 162.94 and 174.3 seconds when the number of cloudlets was increased from 25 to 250 by groups of 25 cloudlets. In the figure, we have performed the comparison with the scenario composed of homogeneous VM (curve in red). Overall, we obtained that, with this heterogeneous scenario, the makespan for the entire set of cloudlets was significantly reduced. The same scenario when using the mesh of 1,152 elements is presented in Fig. 12b. Here, the makespan was 146.37 seconds and 677.64 seconds when the number of cloudlets was equal to 25 and 250, respectively.



(a) Mesh of 288 elements (b) Mesh of 1,152 elements

Fig. 12. Time-shared provisioning for VMs and jobs using resource heterogeneity: Results

For the sake of summarization, Table 6 shows a comparison of the different obtained makespans for each group of jobs (from 25 to 250) when the number of processing elements within hosts varies as described previously. For space reasons, we have summarized the experiments corresponding to the mesh of 1,152 elements only.

Summary and discussion

In the previous sections we have reported the results obtained from executing our PSE on a simulated Cloud under two different scenarios and also taking into account whether the resources are homogeneous or heterogeneous. According to the experimental results the makespan of cloudlets depends on the policy used to assign the VMs to hosts. When the space-shared policy is used, the scheduler assigns as many VMs as PEs have available on the hosts (20 VMs in the proposed scenario). Instead, when time-shared policy is used, the PEs share their time slots among all VMs to be created (40 VMs).

On the other hand, the cloudlets are sent to be executed something similar happens. When using the space-shared policy each cloudlet is assigned to a VM until it completes its execution. With the time-sharing policy processing power must be shared among several cloudlets, generating a lot of exchanges for completing their execution, which makes each cloudlet to take longer to finish.

The better performance obtained by the space-shared policy is mainly because each VM can allocate and get all the processing power that needs to execute assigned cloudlets from the host where the VM executes. Instead, with the time-shared policy, each VM receives a time slice on each processing element, and then distributes the slices among the PSEs to be executed. Due to the fact that the VMs have less processing power (time slices) the experiments took longer to complete. As a result, the space-shared policy was more appropriate to this type of PSEs.

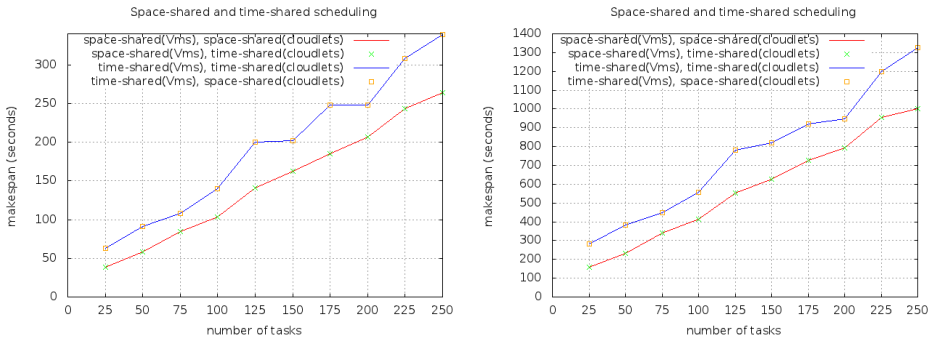
Table 6. Summary of the makespan for the mesh of 1,152 elements (SS=Space-shared, TS=Time-shared)

Cloudlets	Homogeneous				Heterogeneous			
	SS(VMs), SS(cloudlets)	SS(VMs), TS(cloudlets)	TS(VMs), SS(cloudlets)	TS(VMs), TS(cloudlets)	SS(VMs), SS(cloudlets)	SS(VMs), TS(cloudlets)	TS(VMs), SS(cloudlets)	TS(VMs), TS(cloudlets)
25	160.25 sec.	160.25 sec.	280.94 sec.	280.94 sec.	117.12 sec.	117.12 sec.	146.37 sec.	146.37 sec.
50	232.36 sec.	232.36 sec.	384.30 sec.	384.30 sec.	150.24 sec.	157.21 sec.	168.24 sec.	160.01 sec.
75	342.26 sec.	342.26 sec.	449.11 sec.	449.11 sec.	231.38 sec.	231.38 sec.	198.92 sec.	210.52 sec.
100	414.27 sec.	414.27 sec.	557.20 sec.	557.20 sec.	235.43 sec.	324.52 sec.	230.85 sec.	229.86 sec.
125	553.39 sec.	553.39 sec.	780.55 sec.	780.55 sec.	288.54 sec.	342.74 sec.	355.53 sec.	267.00 sec.
150	625.41 sec.	625.41 sec.	821.43 sec.	821.43 sec.	380.74 sec.	381.91 sec.	468.62 sec.	340.74 sec.
175	727.95 sec.	727.95 sec.	922.30 sec.	922.30 sec.	454.94 sec.	414.94 sec.	480.27 sec.	385.60 sec.
200	794.04 sec.	794.04 sec.	946.30 sec.	946.30 sec.	499.00 sec.	451.04 sec.	547.26 sec.	416.02 sec.
225	954.40 sec.	954.40 sec.	1,198.52 sec.	1,198.52 sec.	563.07 sec.	551.88 sec.	569.09 sec.	541.54 sec.

250	1,000.86 sec.	1,000.86 sec.	1,328.13 sec.	1,328.13 sec.	604.22 sec.	606.57 sec.	599.05 sec.	677.64 sec.

Subsequently, we performed the analysis of the same scenarios, but this time in a heterogeneous environment. The behavior of the different combinations of scheduling techniques was exactly the same as the case of homogeneous resources. Although the scheduling criteria to assign VMs to hosts and cloudlets to VMs were the same, the makespan was significantly reduced for all scenarios. This improvement in the makespan was because in the heterogeneous environment were available as many PEs as needed to run experiments.

A summary of the four scenarios presented above for the homogeneous environment and for the meshes of 288 and 1,152 elements are illustrated in Fig. 13a and 13b. As the reader can see, the makespan coincide whenever the scheduling policy used to allocate the VM is the same (blue and orange curves for space-shared policies, red and green curves for time-shared policies).



(a) Mesh of 288 elements (b) Mesh of 1,152 elements

Fig. 13. Space-shared and time-shared policies for VMs and cloudlets: Results

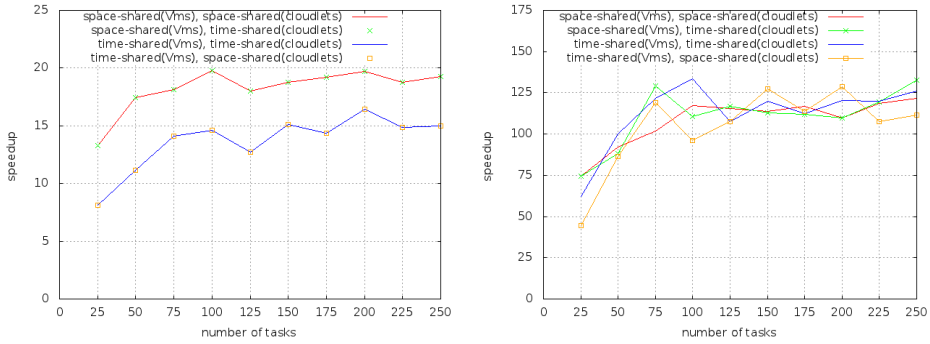
Finally, we performed a speedup analysis to measure the performance of each technique (space-shared and time-shared) to execute the PSEs on the Cloud with respect to the sequential execution on a single machine. The speedup for the mesh of 288 elements is shown in Fig. 14, and the speedup for the mesh of 1,152 elements in Fig. 15.

The speedup is calculated as:

$$S_p = T_1 / T_p \qquad (2)$$

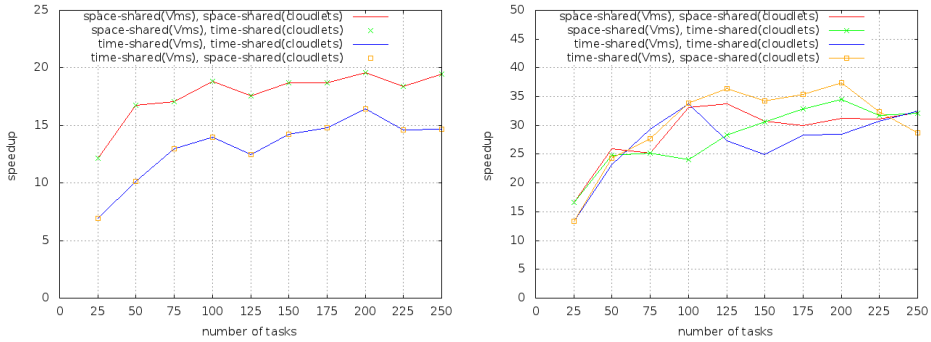
were,

- p is the number of processing elements
- T_1 is the completion time of the sequential execution in a single machine.
- T_p is the completion time of the parallel execution with p processing elements



(a) Without resource heterogeneity (b) With resource heterogeneity

Fig. 14. Speedup achieved by space-shared and time-shared policies (mesh of 288 elements): Results



(a) Without resource heterogeneity (b) With resource heterogeneity

Fig. 15. Speedup respect to space-shared and time-shared policies (mesh of 1,152 elements): Results

We conclude that a scenario in which the space-sharing policy is used for the VMs allocation to hosts enables a better speedup than the time-sharing policy in both scenarios (homogeneous and heterogeneous). While for the experiments we have conducted in this work a space-share policy for the allocation of VMs to hosts yields better results, due to the fact that the employed cloudlets are sequential –i.e. they have no inner parallelism to exploit–, a time-share policy to assign the VMs to hosts would be more appropriate for other types of applications (not batch or sequential) and also could be good to improve not only the makespan but also the perceptible response time to the user, since incoming jobs could be periodically scheduled and then executed in small groups, thus giving sign of progress.

Conclusions

Cloud Computing is a new paradigm that provides the means for building the next generation distributed and parallel computing infrastructures. Although the use of Clouds finds its roots in IT environments, the idea is gradually entering scientific and academic ones. Even when the positive effects of Cloud Computing regarding simplified administration is a well-known fact, little research has been done with respect to evaluating the benefits of the paradigm for scheduling and executing resource intensive scientific applications. In this sense, through a real case study and simulations, we have reported on the speedups obtained when running parameter sweep experiments on Clouds. Results are quite encouraging and actually support the idea of using Clouds in the academia

At present, we are extending this work in several directions. First, we are conducting studies with other kind of PSEs, such as tension tests in metals [27], to further support our claims. Second, one of the key points to achieve good performance when using Clouds concerns job scheduling. In particular, there is an important amount of work in this respect in the area of Cloud Computing and distributed systems in general that aim at building schedulers by borrowing notions from Swarm Intelligence (SI), a branch of Artificial Intelligence that comprise models that resemble the collective behavior of decentralized, self-organized systems like ants, bees or birds. Moreover, a recent survey of our own [30] shows that there is little work regarding SI-based schedulers for Cloud Computing. Therefore, we aim at designing a new SI-based scheduler that is capable of efficiently run PSEs. We are also planning to embed the resulting scheduler into CloudSim in order to provide empirical evidence of its effectiveness. Eventually, we could implement the scheduler on top of a real (not simulated) Cloud middleware or support, such as Eucalyptus (http://www.eucalyptus.com).

References

1. Youn C. and Kaiser T. Management of a parameter sweep for scientific applications on cluster environments. Concurrency and Computation: Practice and Experience, vol. 22, pp. 2381–2400, (2010)
2. Thain D., Tannenbaum T., and Livny M. Distributed computing in practice: The Condor experience. Concurrency and Computation: Practice and Experience, vol. 17, pp. 323–356, (2005)
3. Foster I. and Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Inc., San Francisco, CA, USA, (2003)
4. Foster I., Zhao Y., Raicu I. and Lu S. Cloud Computing and Grid Computing 360-degree compared. In Grid Computing Environments Workshop (GCE '08), IEEE Computer Society, pp. 1-10, (November 2008)
5. Buyya R., Yeo C.S., Venugopal S., Broberg J., and Brandic I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, vol. 25, pp. 599–616, (2009)

6. Dikaiakos M.D., Katsaros D., Mehra P., Pallis G. and Vakali, A. Cloud Computing: Distributed Internet Computing for IT and Scientific Research, IEEE Internet Computing, vol.13, No.5, pp.10-13, (September 2009)
7. García Garino C., Ribero Vairo M., Andía Fagés S., Mirasso A. and Ponthot J.-P.: Numerical Simulation of Finite Strain Viscoplastic Problems. Invited paper, Numerical methods for large deformation analysis minisymposium, Boman R. & Ponthot J.-P. (Organisers) in Proceedings of Fifth International Conference on Advanced COmputational Methods in ENgineering (ACOMEN 2011), M. Hogge et al.(Eds.), Liège, Belgium, 14-17 November 2011, University of Liège, (2011). ISBN: 978-2-9601143-1-7
8. Careglio C., Monge D., Pacini C., Mateos C., Mirasso A. and García Garino C.: Sensibilidad de resultados del ensayo de tracción simple frente a diferentes tamaños y tipos de imperfecciones. In Dvorkin E., Goldschmit M., and M. Storti, editors, Proceedings of II South American Congress on Computational Mechanics (MECOM 2010), Mecánica Computacional, XXIX, pages 4181-4197, Buenos Aires, Argentina, (2010), AMCA. ISSN 1666-6070.
9. Calheiros R.N., Ranjan R., Beloglazov A., De Rose C., and Buyya R. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software - Practice and Experience, vol. 41, pp. 23–50, (2011)
10. Samples M.E., Daida J.M., Byom M. and Pizzimenti M. Parameter sweeps for exploring GP parameters. In Conference on Genetic and Evolutionary Computation, GECCO '05, pp. 212-219, (2005)
11. Sun C., Kim B., Yi G. and Park H. A model of problem solving environment for integrated bioinformatics solution on grid by using condor. In Grid and Cooperative Computing, pp. 935-938, (2004)
12. Gulamali M., Mcgough A., Newhouse S. and Darlington J. Using ICENI to run parameter sweep applications across multiple grid resources. In Global Grid Forum 10, Case Studies on Grid Applications Workshop, (2004)
13. Basney J., Livny M. and Mazzanti P. Harnessing the capacity of computational grids for high energy physics. In Conference on Computing in High Energy and Nuclear Physics, (2000)
14. Wozniak J., Striegel A., Salyers D. and Izaguirre J. GIPSE: Streamlining the management of simulation on the Grid. In 38th Annual Simulation Symposium, pp. 130-37, (2005)
15. Axelrod R. The Dissemination of Culture: A Model with Local Convergence and Global Polarization. Journal of Conflict Resolution, vol.41, No.2, pp. 203-226, (1997)
16. Coveney P., Chin J., Harvey M. and Jha S. Scientific grid computing: The first generation. Computing in Science and Engineering, vol. 7, pp. 24-32, (September 2005)
17. Pordes R., Petravick D., Kramer B., Olson D., Livny M., Roy A., Avery P., Black-burn K., Wenaus T., Wurthwein F., Foster I., Gardner R., Wilde M., Blatecky A., McGee J. and Quick R. The open science grid. Journal of Physics: Conference Series, vol.78, No.1, pp.012-057, (2007)
18. Gagliardi F. and Begin M.E.. Egee - providing a production quality grid for e-science. In Proceedings of the 2005 IEEE International Symposium on Mass Storage Systems and Technology, IEEE Computer Society, pp. 88-92 (2005)
19. Catlett C TeraGrid: A Foundation for US Cyber infrastructure. In NPC'05, Berlin Heidelberg, Springer, (2005)
20. Gropp W., Lusk E., and Skjellum A. Using MPI: Portable Parallel Programming with the Message Passing Interface. MIT Press, (1994)

21. Erickson J. and Siau K. Web Service, Service-Oriented Computing, and Service-Oriented Architecture: Separating hype from reality. Journal of Database Management, vol. 19, pp. 42–54, (2008)
22. Wang L., Tao J., Kunze M., Castellanos A.C., Kramer D., and Karl W. Scientific cloud computing: Early definition and experience. In: 10th IEEE International Conference on High Performance Computing and Communications, pp. 825–830, (2008)
23. Pacini E., Ribero M., Mateos C., Mirasso A. and García Garino A. Simulation on Cloud Computing Infrastructures of Parametric Studies of Nonlinear Solids Problems. Advances in New Technologies, Interactive Interfaces and Communicability (ADNTIIC 2011), pp. 56-68, F.V. Cipolla-Ficarra, A. Kratky, K. Veltman, M. Brie, J. Huang, E. Nicol, G. Mori, and M. Cipolla-Ficarra (Eds.), Blue Herons, (2011). ISBN 978-88-96471-03-6.
24. Alfano G., Angelis F.D., and Rosati L. General Solution procedures in elasto-viscoplasticity. Computer Methods in Applied Mechanics and Engineering, vol. 190, pp. 5123–5147, (2001)
25. García Garino C. and Oliver J. Un modelo constitutivo para el análisis de sólidos elastoplásticos sometidos a grandes deformaciones: Parte i formulación teórica y aplicación a metales. Revista internacional de métodos numéricos para cálculo y diseño en ingeniería, vol.11, pp. 105–122, (1995)
26. Garcia Garino C. and Oliver J. Un modelo constitutivo para el análisis de sólidos elastoplásticos sometidos a grandes deformaciones: Parte ii implementación numérica y ejemplos de aplicación. Revista internacional de métodos numéricos para cálculo y diseño en ingeniería, vol.12, pp. 147–169, (1996)
27. García Garino C., Gabaldón F., and Goicolea J.M. Finite element simulation of the simple tension test in metals. Finite Elements in Analysis and Design, vol. 42 (13), pp. 1187–1197, (2006)
28. Ribero Vairo M., van Hooijdonk J., Andía Fagés S., Mirasso A. and García Garino C.: Análisis de un modelo elasto-viscoplástico no-lineal, Actas del ENIEF 2011, XIX Congreso sobre Métodos Numéricos y sus Aplicaciones. Mecánica Computacional Vol XXX, pp. 787-803, O. Möller, J. W. Signorelli, M. A. Storti (Eds.), AMCA, (2011). ISSN 1666-6070.
29. Van Dorst W. Bogomips mini-howto. http://www.clifton.nl/bogomips.html, (March 2006)
30. Pacini E., Mateos C., and García Garino C. Planificadores basados en inteligencia colectiva para experimentos de simulación numérica en entornos distribuidos. In: Sexta Edición del Encuentro de Investigadores y Docentes de Ingeniería ENIDI'11. (2011)