University of Castilla-La Mancha

A publication of the

Computing Systems Department

Nacreous: An Adaptive Service–Aware IaaS Cloud Manager

by

Javier Conejero, Carmen Carrión and Blanca Caminero

Technical Report **#XXX** Mes, año

This work was supported by the Spanish Government under Grant TIN2012-38341-C04-04 and through a FPI scholarship associated to TIN2009-14475-C04-03 project.

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

Nacreous: An Adaptive Service–Aware IaaS Cloud Manager *

Javier Conejero Carmen Carrión
Blanca Caminero
Computing Systems Department. Escuela Politécnica Superior
Universidad de Castilla-La Mancha. 02071 - Albacete, SPAIN
{francisco.conejero, mariablanca.caminero, carmen.carrion}@uclm.es

9 de marzo de 2015

Resumen

Nowadays, Cloud Computing is a trending topic both in academia and in industry, offering a wide range of services at different levels. Thus, efficient Cloud service management poses a challenge on Cloud providers, specially when non-trivial Quality of Service (QoS) guarantees are expected by users. Also, Cloud providers seek to maximize their Return on Investment (ROI) by managing the underlying resources in the most efficient way, so that clients receive their expected level of service with the minimum possible amount of allocated resources and other associated costs (i.e., energy consumption).

In this work an extensible framework, named Nacreous, is proposed. Nacreous is able to consolidate and coordinate multiple services within an IaaS Cloud, and optimise them by allowing different adaptive and complex policies for deployment and scheduling to be implemented. From the user point of view, Nacreous will

*This work was supported by the Spanish Government under Grant TIN2012-38341-C04-04 and through a FPI scholarship associated to TIN2009-14475-C04-03 project.

enable the specification of simple or complex IaaS service requests by using a WS-Agreement based interface (or a CLI API). Some experiments aimed at providing a glance of Nacreous basic functionalities over a private Cloud testbed, with a specific attention to Hadoop applications, are also provided.

1. Introduction

Over recent years, Cloud Computing is being considered a revolutionary and hot topic technology. The amount of efforts and developments, both from academia and industry, focused on its implantation and integration across datacenters are growing quickly and seem to be unstoppable. The Cloud computing concept is very wide and offers various service models, namely: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). There are three deployment models that a Cloud environment can follow, these are: Public, Private and Hybrid (as a combination of both). The amount of services that Cloud computing can provide within each service model is huge. In particular, so-called Big-Data applications, like Hadoop [1], are experiencing an increasing demand in order to be able to analyse the growing amount of data produced every day by social media networks (e.g., Twitter [2], Facebook [3], etc.) [4]. These applications require a vast amount of resources in order to perform an analysis across a representative amount of data.

On the other hand, pay–per–use models impose a number of terms, associated requirements and costs, that are usually established by SLA negotiation between the users and the service provider. These terms condition the decisions that a Cloud service provider needs to take when negotiating an agreement with its clients. More precisely, the performance of the backend Cloud infrastructure, the types of services being demanded, the amount of users and their expectations varies over time. Consequently,

any information that can be extracted from the behaviour of both Cloud infrastructure and users, can be useful in order to optimise resource usage, i.e., defining an adequate exploitation policy (e.g., in terms of energy saving, maximum performance, etc.), while at the same time fulfilling the maximum number of established SLAs.

The purpose of this work is to design and provide an extensible framework able to consolidate multiple IaaS related services within Cloud environments and optimise them by using different adaptive and complex deployment policies, based on historical performance monitorization or on individual IaaS service provider specifications. More specifically, we focus on two main types of services: 1) providing virtual infrastructures that customers can use in order to deploy and exploit particular applications, and 2) providing applications within virtual infrastructures, adaptively, configured and controlled. Both services can require complex virtual infrastructures. Thus, an architecture able to consolidate and manage multiple (related or not) Virtual Machines (VMs) within the same infrastructure and coordinate their service offering and provisioning is proposed here. It is not specific to any particular IaaS Cloud environment, and it can be integrated into existing Cloud infrastructures. This architecture has been implemented as a ready–to–use and extendable framework, called *Nacreous*, that attends the users expectations by offering them a WS-Agreement compliant service.

This paper is structured as follows. Nacreous architecture, including some insight into its features and internal design, is presented in detail in Section 2. Details on the implantation and some proof of concept results are provided in Section 3. In Section 4 related work is presented. Finally, the conclusions derived from the work and suggested future work are presented in Section 5.

2. Nacreous

In this Section, the Nacreous[1] framework is described in detail, paying special attention to the new additions to the existing Cloud infrastructures.

Formally, Nacreous is a framework able to consolidate and manage multiple services within the same infrastructure, and coordinate the service offering and provisioning requested by users. Nacreous will smartly act on IaaS Cloud managers, but will help to hide the complexities of the services provided. Consequently, it has been developed as a modular software layer that lays on top of a whole IaaS Cloud architecture (i.e., OpenNebula, Openstack, etc.) and it is ready to attend users directly (Figure 1). Nacreous can obtain the monitoring performance from the underlying IaaS Cloud manager and also the users expectations, and act as a broker that can be configured with adaptive exploitation policies.

Nacreous is aimed at providing an adaptive IaaS service provisioning. To this end, Nacreous interacts with the underlying IaaS managers in order to provide the best service (previously negotiated through SLAs with the users). Consequently, Nacreous can handle an IaaS manager in order to deploy a complex virtual infrastructure and a specific service/application on it (e.g., Hadoop virtual cluster). But it can also handle existing virtual infrastructures, leveraging the complexities of specific services or applications. For example, Nacreous can be used to exploit Hadoop virtual clusters within Cloud environments, hiding the deployment and configuration complexities, or alternatively, it can handle and exploit an existing Hadoop virtual cluster. It must be noted that Nacreous does not interact directly with the virtualization software layer (i.e., hypervisors such as KVM). Alternatively, it delegates this tasks to the underlying

[1]Nacreous is named after the Polar Stratospheric Clouds (PSCs), which are known as nacreous clouds (from nacre) due to its iridescence. These clouds can be found at a very high altitude in the stratosphere, over most clouds.

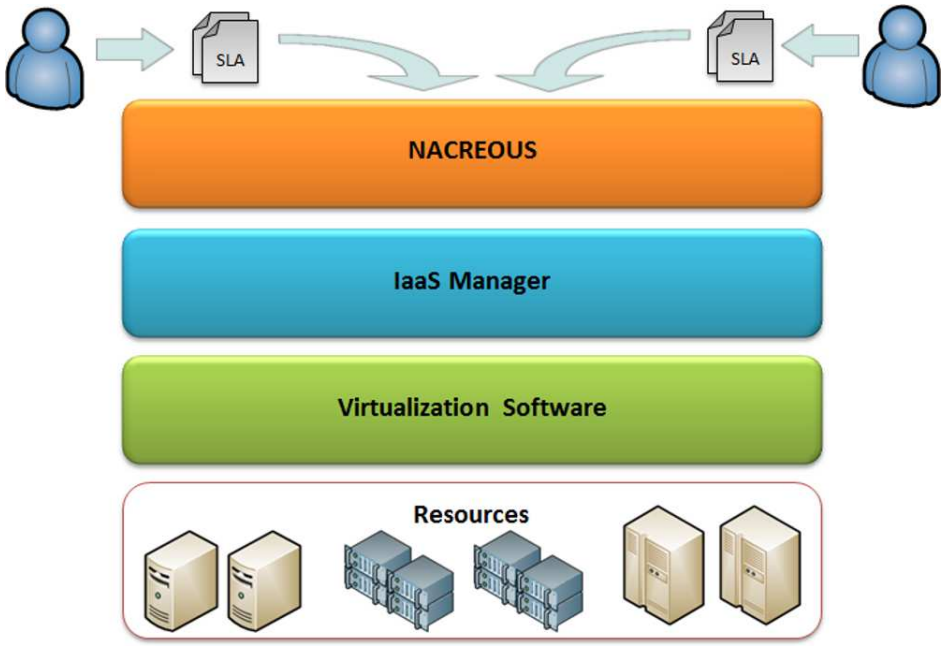


Figura 1: Proposed Cloud Architecture.

IaaS manager, avoiding the complexities associated with these tasks.

2.1. Features

The IaaS Cloud model provides users with virtual infrastructures in order to exploit them. The virtual infrastructures offered are single (or groups of individual) virtual machines (VMs) that can be deployed following different policies (e.g., maximum spread, consolidation policies, etc.). There are also services/applications that require specific deployment architectures. One example are distributed applications (such as Map/Reduce, MPI, etc.) which require a cluster architecture, where a group of related VMs, configured as a Virtual Cluster (VC) is needed. Furthermore, the virtual infrastructures that can be deployed are offered for a multi–purpose mission. This fact complicates the requirements provisioning in order to fulfil the SLAs. Moreover, it is important to take into account the inter–VM interferences that may occur when they are being exploited in order to achieve the best performance.

Nacreous provides the functionalities to handle complex virtual infrastructures depending on specific applications/services requirements explicitly defined by users, that is, sets complex virtualization contexts within single VMs. On the other hand, taking into account inter–VM interference is not an easy task since the service provider needs to have information related to the service deployed within the virtual infrastructure. To this end, Nacreous incorporates performance monitors and predictors that keep track of the evolution of the overall infrastructure and executions.

Moreover, Nacreous provides mechanisms for QoS differentiation. To this end, it allows service providers to define a multi–level policy in terms of the explicitly defined SLA terms, specific Service Level Objectives (SLOs) or even higher level parameters such as confidence on the service.

These features are explained more in detail in the next paragraphs, distinguishing the features related to clients and to the services/applications offered. The first one is focused on the interaction between Nacreous and the users, and the second one focuses on the features related to the services provided.

2.1.1. Client features

In order to attend users, Nacreous implements a WS-Agreement compliant service [5]. This service is in charge of attending the users allowing them to negotiate and establish SLAs and delegate their job/service execution. Although WS-Agreement provides the structure, workflow and negotiation protocols, it leaves the negotiation terms to be defined by the service provider. Thus, Nacreous provides an initial set of terms (Table 1) that can be easily customized and/or extended.

The *USER* and *NAME* terms are used for human identification purposes and track the users trends. The *JOB* term identifies the kind of service requested, that is,

Cuadro 1: SLA Template Terms and Example

TERM	DESCRIPTION	EXAMPLE
USER	User name (identification)	Client-5
NAME	Submission Name (identification)	MRtest
JOB	Job/Service to be executed	Hadoop
START	Start day and time	15-Jul-2014 14:00:00
END	Expected end day and time	15-Jul-2014 18:00:00
QoS	Expected QoS	High priority
REQUIREMENTS	Specific VM resources reqs.	20 Gb HDD; 1 Gb RAM; Linux
NUM_SLAVES	Amount of slaves (VC mode)	5
SLAVES_REQS.	Specific VM slaves resources reqs.	20 Gb HDD; 1 Gb RAM; Linux
COST	Service cost	TBD
PENALIZATION	Penalization conditions	TBD
VALUE	Penalization value	TBD

the availability of an infrastructure for future use or the execution of an application. The *START* and *END* terms are used to determine the period of time when the user wants the service, and the *QoS* term is used to represent the QoS level that the user expects to be provided. This level depends on the specific exploitation policy, and allows the service provider to settle the service cost. For example, a service provider can set a number of QoS levels to be offered. As a result, users may request a specific level for the service requested in terms of the specific QoS level conditions, such as: performance ensured, costs, etc. Moreover, it can also be used in order to achieve the confidence that users expect from the service provider for the agreed service. The *REQUIREMENTS*, *NUM_SLAVES* and *SLAVES_REQUIREMENTS* terms are optional. The first one, *REQUIREMENTS* is used to specify the requirements that the user imposes (these requirements can go from specific software versions, to amount of resources required and/or hardware restrictions). The *NUM_SLAVES* and *SLAVES_REQUIREMENTS* are used to extend the requirements that a user can specify by allowing to define

a virtual cluster. For single VM requests, these terms are not specified.Finally, the *COST*, *PENALIZATION*, and *VALUE* are used to agree on the economical terms. The *COST* term is set by the service provider in order to negotiate the service price, while the *PENALIZATION* term is used to specify the agreement violation conditions, and the *VALUE* term specifies the associated violation cost.

An example of use of this template is also shown in Table 1. In this case, a user named *Client-5* requests a Virtual Cluster (VC) composed of 6 VMs with Linux OS (1 master with 20 Gb of storage and 1 Gb of RAM and 5 workers with 8 Gb of storage and 1 Gb of RAM) during 3 hours starting at 14:00 on the 15th of July 2014 with high priority. In this example, the terms related to costs and penalization are omitted since they have to be determined by the provider.

Nacreous is able to use the underlying CLI API when managing a private IaaS Cloud infrastructure. Also, Nacreous could manage public IaaS though OCCI [6] compliant APIs.

2.1.2. Service features

The main aspect that the Nacreous framework provides is the ability to manage and control multiple services within multiple resource provisioning infrastructures. Consequently, Nacreous provides an extensible monitoring system and a complex policy definition system.

Nacreous supports multiple information sources and monitoring applications. All the information gathered from these sources is candidate to be used within management policies. More in detail, the monitoring information that Nacreous exploits is focused on: resource status, virtual infrastructure status, client demands and behaviour and offered service status.

Nacreous also provides mechanisms for generic VM image management (that implements different services), that can be used within IaaS infrastructures in order to provide multiple services, and configuration mechanisms for each VM image being deployed. For example, when a user requires a VM with a specific software, the system deploys the generic VM image that best suits the user requirements, and customizes that VM for that user following the requested specifications.

Finally, Nacreous provides scheduling features, since it is able to interact with multiple virtual infrastructures. To this end, it is able to submit, delegate, and control the jobs/services offered by using the underlying management mechanisms.

2.2. Internal Design

Nacreous is a modular software layer that lays on top of a whole IaaS Cloud architecture and provides an interface to the users (Figure 2).

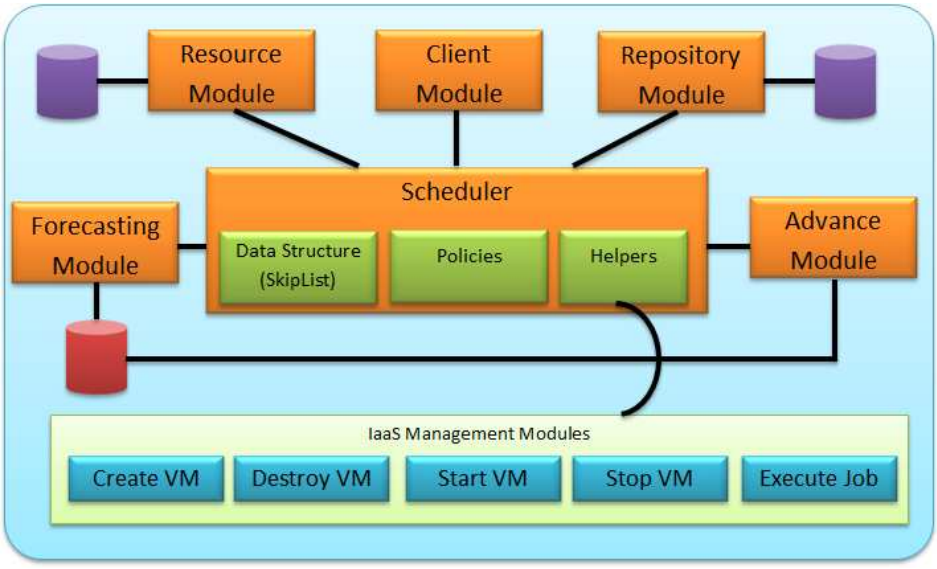


Figura 2: Nacreous internal architecture.

Besides a WS-Agreement compliant service, Nacreous provides users with a Command Line Interface (CLI) API. Both are implemented within the *Client Module*. It is

in charge of attending the users allowing them to negotiate and establish agreements (SLAs) with the Cloud service provider and delegate their job/service execution.

The Cloud infrastructure status monitoring and performance evolution is controlled by the *Resource Module*, thanks to the communication with the *Scheduler*. Thus, it also keeps track of the infrastructure, monitoring the available resources and their associated performance. These data are useful in order to estimate which resources are (and how) being used by Nacreous for the given users requests and requirements. The available VM images and offered applications are also monitored. To this end, the *Repository Module* updates periodically this information. The *Resource Service* and the *Repository Service* gather the information from the available monitoring tools through wrappers. Consequently, they are prepared to work with any underlying IaaS Cloud provider, since they provide the required information (e.g., amount of resources associated, resources use percentage, VM images offered, applications offered). If there is any parameter that can not be obtained, or another monitoring software wants to be used alternatively, Nacreous can be easily adapted.

The core module, namely *Scheduler*, applies the exploitation policies in terms of the monitoring results, users requests and requirements, and future expected performance. The service level or QoS differentiation is specified in this module. It can depend on specific service provider needs and it can be defined in high level terms that the policy must translate into low level terms for the service/infrastructure management. The *Scheduler* implements three main sub–modules. The first one is *Data Structure*, where the underlying resources are mapped to a data structure in order to keep track of the physical resources assignment and ease the decision taking process. This data structure must be efficient in terms of speed and space. The policies are explicitly defined in the *Policies* sub–module. And, in order to apply the decisions taken, the *Scheduler* implements a specific set of sub–modules, known as *Helpers*, which are in

charge of transmitting the actions taken to the underlying Cloud infrastructure.

Nacreous has been designed considering the future use of performance and users expectation forecasting (*Forecast Module*) and proactive mechanisms (*Advance Module*). These modules interact with the *Scheduler* and among them, in order to perform all the forecasting process and to perform proactive actions.

3. Implantation and Proof of Concept

The Nacreous framework proposed in this work has been successfully tested on the University of Castilla-La Mancha (UCLM) Cloud testbed. In this Section, some details on its implantation, service, operation and proof of concept are provided.

3.1. Testbed Description

The *University of Castilla-La Mancha* (UCLM) Cloud testbed (known as *Vesuvius*) is composed of 10 compute nodes, with 2 Xeon e5462 CPU (4 cores per processor), 32 GB of main memory and 4 TB of storage each. It also has a headnode, with the same configuration and a Gigabit Ethernet network across all nodes. The OS deployed across all nodes is CentOS 6.2 Linux [7].

The virtualization software deployed is the KVM (Kernel-based Virtual Machine) hypervisor [8] and the IaaS Cloud management system available in the *Vesuvius* Cloud is OpenNebula 4.2 [9]. OpenNebula manages KVM in order to deploy, monitor and control any virtual machine on each physical machine associated with the local Cloud infrastructure.

We have chosen the Hadoop [1] framework as an example of service that can

be deployed through Nacreous in a convenient way. To this end, a set of VM images containing Hadoop packages installed on them have been created.

Providing "Hadoop as a Service" requires an application and data to be consumed by Hadoop. This is usually left to clients decision. However, for this proof of concept, the Hadoop application chosen for this evaluation is the Sentiment Analysis Tool for Hadoop [10] designed by the COSMOS project [11]. In this case, we will request the analysis of 100 million tweets in order to get their positive or negative sentiment. This tool has shown to be interesting within research and marketing industries, but it imposes a heavy stress on CPU and I/O and requires a distributed environment, such as a cluster (or in this environment a virtual cluster (VC)).

Using Nacreous to deploy VCs and exploiting a service on them, the process and the complexities derived from such a complex environment deployment and setup are hidden to users. We have performed several tests by requesting multiple VCs with different amounts of *slaves*. To this end, all VMs that belong to the VC have the same specifications: $CORES = 1; RAM = 1{,}7; HDD = 8Gb; ARCH = x64; OS = Ubuntu;$

3.2. Experimental Results

In order to perform a proof of concept of the Nacreous framework, we have specified one deployment policy (maximum consolidation) and considered 3 sets of 60 requests of VCs. The maximum consolidation policy consists on concentrating as many VMs as possible within the minimum amount of physical nodes without oversubscribing the physical resources. Consequently, the amount of physical nodes used is reduced and the power consumption minimised. The three sets of requests contain 60 VC requests, composed by a random number of VMs (Master + Workers) between 2 and 8.

Each VC query requests Hadoop services (more specifically, the Sentiment Anal-

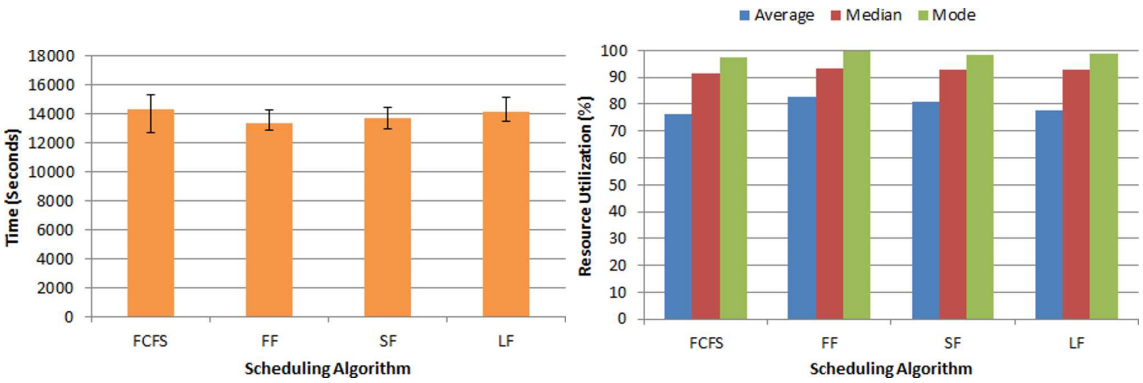
ysis for Hadoop). The time range required to perform this analysis (considering all the stages needed: deployment, boot, contextualization, Hadoop start-up, HDFS set-up, analysis and destruction) varies from $\sim$ 65 minutes to $\sim$ 45 minutes depending on the amount of VMs per VC (between 2 and 8).

We have evaluated Nacreous performance without time restrictions in order to analyse the overall performance and decisions with multiple queries. This scenario is considered as the reception of a set of queries that request the same *start-time* with unknown *end-time*.

We analyse the overall performance, waiting times and resource utilization taking into account four scheduling algorithms, namely, First Come First Served (FCFS), First Fit (FF), Small First (SF) and Large First (LF). They have been chosen due to the fact that they represent the basis of more complex scheduling algorithms, showing the ability to decide, sort and reschedule. FCFS and FF are the well-known scheduling algorithms. SF and LF sort requests in ascending or descending order, according to the size of the virtual cluster.

Nacreous is able to process the sets of requests in less than 240 minutes applying the FCFS scheduling algorithm for the three sets in average (Figure 3a). The performance is increased a 6.46 % applying FF compared to FCFS. While applying the SF and BF scheduling algorithms, the performance improvement is lower: $\sim$ 3,9 and $\sim$ 3,1 respectively.

Regarding resource utilization (Figure 3b), it is interesting to observe how the FF algorithm achieves best results compared with the other algorithms. This is due to the fact that FF tries to allocate another VC when the FCFS has decided to wait. Consequently, if another query fits into the available resources, it is allocated, exploiting more resources at the same time. Since the SF and LF scheduling algorithms rely on the FCFS algorithm, they both achieve similar resource utilization to FCFS. However,



(a) Average processing time.

(b) Resource utilization.

Figura 3: Performance

Cuadro 2: Waiting time (Seconds)

	FCFS	**FF**	**SF**	**LF**
Min.	2765.00	2831.00	2887.00	2775.00
Perc. 10	3652.47	3309.77	3876.07	3319.73
Median	6763.50	6190.33	6771.50	7062.00
Perc. 90	9883.80	9691.33	9971.93	9759.80
Max.	10683.00	11462.00	11521.00	10332.00
AVERAGE	6729.28	6128.99	6824.20	6956.21

depending on the request order, the performance is slightly improved.

Observing the requests waiting time avoiding the transitory period (period until the system gets full of VCs for the first time) (Table 2), the FF algorithm has shown to reduce the average waiting time compared to FCFS. However, due to the FF nature, some queries that arrived later where performed earlier than others that arrived before (so that they had to wait more). The SF and LF algorithms have shown to achieve worse average waiting time than FCFS. However, the waiting time range and median are similar to FCFS.

Finally, we want to remark that Nacreous has shown to handle large amounts of

VCs (and consequently, large amounts of related VMs), across a real Cloud testbed, to a set of users with different demands. All SLAs received from the users were accepted, and the QoS experienced is related to the waiting and processing time. As a result, we have achieved better results applying the FF scheduling. However, considering a maximum waiting time as an SLO, the amount of successful SLAs can be analysed (Figure 4). The FF reduces the amount of violated SLAs compared with FCFS and LF. But the SF achieves better successful ratio for lower maximum accepted time. This is due to the fact that SF starts processing the smaller VC queries, so more VCs achieve resources first, and consequently, the amount of processed VCs grows faster than in the other algorithms. But the larger VC queries experience larger waiting times. The difference between all of them reduces when the maximum accepted waiting time is higher than the 90 % of the maximum waiting time observed in these experiments.

To sum up, the FF scheduling algorithm achieves better results when providing this service, reducing the average processing time, improving slightly the resource utilization, reducing the waiting time and although the SF achieves better successful SLA ratio (in terms of the maximum waiting time SLO), the FF does not penalize so much large VC queries.

4. Related Work

There have been several relevant efforts in Cloud computing management, service offering, SLAs and standardization, service coordination, and resource orchestration. Some of the most significant are reviewed next.

In [12], a multi–level autonomic architecture for the management of virtualized application environments in Cloud platforms is proposed. This work proposes an orchestration framework which is focused on tackling the Cloud resources adaptation in

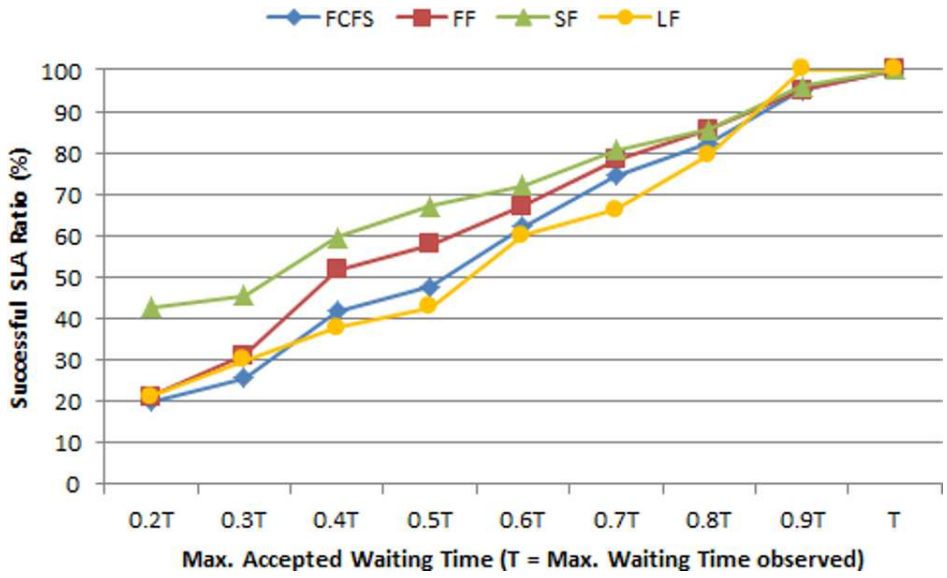


Figura 4: Successful SLA ratio per maximum accepted waiting time.

order to satisfy conflicting objectives of running applications in terms of scaling and/or migration. Alternatively, in [13] a framework for Cloud resource management based on self–tuning fuzzy controllers is described. It provides support for multiple–objective control and service differentiation.

Focusing now on service brokerage, in [14], a brokering service framework aimed at choosing the most appropriate service offered by different Cloud service providers depending on user QoS requirements (e.g., throughput and latency) is described. On the other hand, in [15], a brokering service framework focused on SaaS Cloud providers is presented. Both proposals consider the use of the Pareto optimization technique for the optimal decision.

There are also studies and algorithms in the context of utility computing, where the resource provisioning is conditioned by budget constraints.These proposals are very interesting within the resource orchestration field since they can be useful in order to maximize the return from the Cloud investment (ROI). In [16], a control–based approach for resource provisioning in terms of time and budget constraints is presented.

This proposal focuses on the dynamic CPU cycles and memory assignment for the VMs in order to improve the overall QoS. In [17], a set of three algorithms for resource provisioning of scientific workflow ensembles on IaaS Clouds is presented. These algorithms push for the user-prioritized workflows maximization that can be completed given time and budget constraints, and have shown to achieve important performance results. Moreover, in [18] two auto–scaling mechanisms are proposed. Unfortunately, [12] [16] [17] and [18] proposals lack of an evaluation on a real Cloud environment, or do not consider a multiple scheduling policy architecture [13], while [14] and [15] proposals provide passive brokerage mechanisms that do not interact dynamically with the underlying infrastructure. Furthermore, the efforts performed by Cloud framework providers in terms of Cloud management focus on infrastructure management and service delegation rather than service coordination.

To sum up, all these works and Cloud frameworks provide interesting features within their field. However, there is still a lack of an architecture for a global service–aware smart management within Cloud environments.

5. Conclusions and Future Work

The massive expansion of Cloud computing in conjunction with the amount of services that this technology can provide is enormous, and keeps growing every day in order to satisfy actual and new demands. This fact complicates the service management, and can impact negatively on the amount of fulfilled SLAs, or result in resource overprovisioning. Consequently, increasing the amount of satisfied SLAs is considered the keystone in order to improve the QoS perceived by users and also improve the confidence on the service provider.

The purpose of this work has been to design and provide an extensible framework,

named Nacreous, able to consolidate and manage multiple services within IaaS Cloud environments, and optimise them by allowing to implement different adaptive and complex deployment policies, scheduling heuristics, forecasting and proactive mechanisms.

The results achieved by the successful implementation of Nacreous on the private Cloud testbed deployed at *University of Castilla–La Mancha* show that it is possible to implement the proposed architecture. Moreover, Nacreous initial capabilities have been tested and the results show that it is able to handle the underlying infrastructure, attend users, keep track of the requests and infrastructure, and provide VMs and services based on complex virtual infrastructures.

Experimental results show that the system is able to attend multiple service requests, offering their associated services depending on the user needs.

Some future work guidelines are: to perform a thorough evaluation on the performance impact of multiple deployment policies, to analyse the behaviour of different metrics through the use of different scheduling algorithms, and to conduct a forecasting algorithms evaluation within Nacreous.

Referencias

[1] Hadoop. Apache hadoop. Web page at `http://hadoop.apache.org/`, Last access: 21st January, 2014.

[2] Twitter. Social networking and microblogging service. Web page at `http://twitter.com/`, Last access: 5th February, 2014.

[3] Facebook. Social utility that connects people. Web page at `http://www.facebook.com/`, Last access: 5th February, 2014.

[4] P. Anderson. What is Web 2.0? Ideas, technologies and implications for education. In *JISC Online Report.*, 2007.

[5] Alain Andrieux et al. Web Services Agreement Specification (WS-Agreement) GFD-R-P.192. Technical report, October 2011.

[6] OCCI. Open cloud computing interface. Web page at `http://occi-wg.org/`, Last access: 26th January, 2014.

[7] CentOS. The Community ENTerprise Operating System. Web page at `http://www.centos.org/`, Last access: 14th January, 2014.

[8] KVM. Kernel based virtual machine (kvm). Web page at `http://www.linux-kvm.org/`, Last access: 15th January, 2014.

[9] OpenNebula. The open source solution for data center virtualization. Web page at `http://opennebula.org/`, Last access: 20th January, 2014.

[10] Javier Conejero, Omer Rana, Peter Burnap, and Jeffrey Morgan. Scaling Archived Social Media Data analysis using a Hadoop Cloud. In *IEEE 6th International Conference on Cloud Computing (CLOUD)*, Santa Clara, CA, USA, june 2013.

[11] COSMOS. Cardiff on-line social media observatory. Web page at `http://www.cs.cf.ac.uk/cosmos/`, Last access: 5th February, 2014.

[12] Omar Abdul-Rahman et al. Multi-Level Autonomic Architecture for the Management of Virtualized Application Environments in Cloud Platforms. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011.

[13] Jia Rao et al. QoS Guarantees and Service Differentiation for Dynamic Cloud Applications. *Network and Service Management, IEEE Transactions on*, 10(1):43–55, 2013.

[14] M. Usha et al. An Efficient QoS Framework for Cloud Brokerage Services. In *Cloud and Services Computing (ISCOS), 2012 Intl. Symp. on*, pages 76–79, 2012.

[15] Elarbi Badidi. A Cloud Service Broker for SLA-based SaaS provisioning. In *Information Society (i-Society), 2013 International Conference on*, pages 61–66, 2013.

[16] Qian Zhu and G. Agrawal. Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments. *Services Computing, IEEE Transactions on*, 5(4):497–511, 2012.

[17] M. Malawski et al. Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, 2012.

[18] Ming Mao and M. Humphrey. Scaling and Scheduling to Maximize Application Performance within Budget Constraints in Cloud Workflows. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 67–78, 2013.