Cloud@Home: Performance Management Components

Rocco Aversa[1], Dario Bruneo[3], Antonio Cuomo[2], Beniamino Di Martino[1], Salvatore Distefano[3], Antonio Puliafito[3], Massimiliano Rak[1], Salvatore Venticinque[1], and Umberto Villano[2]

[1] Dipartimento di Ingegneria dell'Informazione,
Seconda Università di Napoli
{rocco.aversa,beniamino.dimartino,massimiliano.rak,salvatore.venticinque}@unina2.it
[2] Dipartimento di Ingegneria,
Università del Sannio
{villano,antonio.cuomo}@unisannio.it
[3] Dipartimento di Matematica,
Università di Messina
{apuliafito,dbruneo,sdistefano}@unime.it

Abstract. This paper shows the design and the mode of operation of the SLA/QoS subsystem devised for Cloud@Home, a cloud environment based on voluntarily-offered resources currently under development in the context of a PRIN-MIUR funded project.

Keywords: Cloud Computing, Volunteer Computing, Performance, SLA

1 Introduction

Cloud computing is a new emerging paradigm that merges a large set of different technologies and solutions. The main idea driving the cloud computing approach is that all resources, hosted by providers in large datacenters, should be accessed through the network with a service-oriented model. Even if the basic principle is relatively simple, it is very hard to give a precise and clear definition of cloud computing [12]. Volunteer computing is a type of distributed computing in which computer owners donate their computing resources (essentially, processing power and storage space) to one or more "projects". The goal is to build up a single infrastructure from small resources, distributed and administrated independently.

The Cloud@Home project (shortly, C@H) is a proposal for a new enhanced paradigm which integrates both cloud and volunteer computing approaches. It is supported by a grant from the Italian Government in the context of MIUR-PRIN 2008, and has received letters of interests from relevant companies active

The work described in this paper has been partly supported by MIUR-PRIN 2008 project "Cloud@Home: a New Enhanced Computing Paradigm" and by II Univ. of Naples PRIST 2009, "Fruizione assistita e context aware di siti ...".

in the cloud computing field. The main idea behind the Cloud@Home project is to build up a cloud by collecting many different kinds of voluntarily-offered resources. This cloud can provide computing, storage and sensor resources both to the contributing "volunteers" and to commercial users, for free or for charge, guaranteeing service levels, negotiated and agreed upon at service request time.

The Cloud@Home project is thoroughly described in a companion paper [13]. In this paper, we focus on the architectural solution proposed to tackle the performance problems, which are particularly complex in a system made up of distributed and independently-administered resources. The remainder of this paper is structured as follows. The next section opens with a brief overview of the Cloud@Home architecture, followed by a detailed description of the components devised for performance management. Section 3 describes how these components, which perform tasks as performance evaluation, prediction and management, have been integrated in the Cloud@Home performance subsystem. In the last section we draw the conclusions and outline the future work.

2 Cloud@Home Architecture

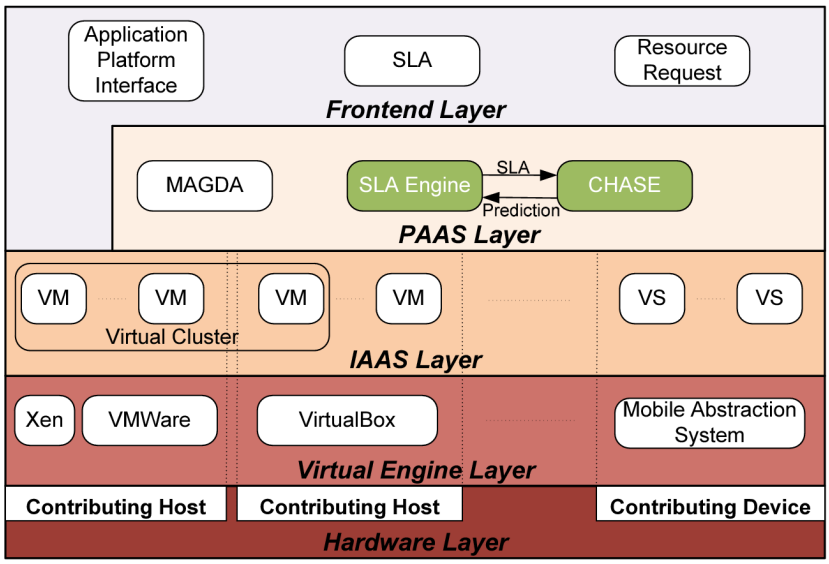


Fig. 1: The *Cloud@Home* architecture

Figure 1 shows the layered architecture of Cloud@Home. The **Hardware Layer** collects all the physical resources available. These range from clusters, datacenter and computing grids to PCs, notebooks or even smartphones (the last are mainly used to exploits their *sensors*).

The **Virtual Engine Layer** offers a *virtualized* version of the physical resources available. The virtual engines have a key role in cloud computing, because they guarantee the full independence of the resources as they are perceived by the users from the physically available ones.

The **C@H IaaS Layer** offers the typical cloud *Infrastructure as a Service* services. It is composed of a set of different cloud middlewares, previously developed by research units involved in the project (*PerfCloud*[7] and *Clever*[14]). A *discovery* layer, which may be implemented in a centralized or distributed way, grants uniform access to the available resources. The discovery layer might be extended in order to support open-source cloud middleware (such as Eucalyptus[11], Nimbus[2] or Open Nebula[1]) or commercial cloud providers (e.g., Amazon EC2). In fact, this extension is not planned within the Cloud@Home project, even if it will be possibly made in the context of future research.

The **C@H PaaS Layer** integrates a set of components that offer a high-level view of the cloud infrastructure and aim at providing the functionalities (services) needed to face the performance problems mentioned previously. Its main components are the SLA Engine, the CHASE autonomic engine, the mobile agents platform and the mobile device middleware.

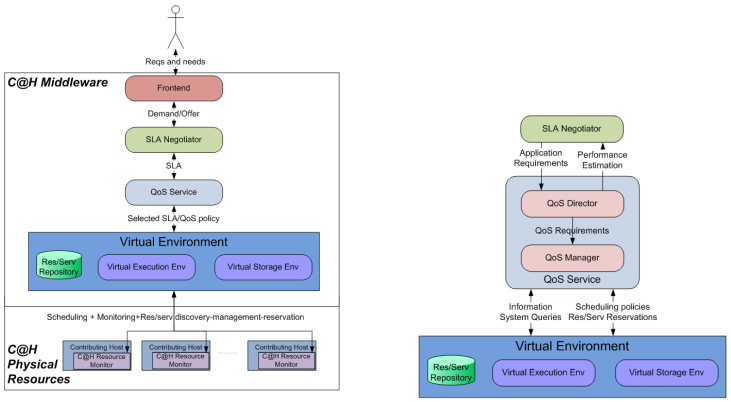
The **C@H Frontend Layer** is a *vertical* layer (in that it cooperates with all the others) that operates as an interface between final users and the Cloud@Home components. It provides services for user request of resources, for request enriched with a SLA, or for starting up mobile agent-based applications.

The focus in this paper is on the subsystem for SLA/QoS management, in green in Figure 1. This subsystem has to provide suitable mechanisms to negotiate SLAs and to guarantee QoS on the top of the virtual environments making up the platform. The proposed architecture involves two components for performance management: SLAEngine and CHASE (developed by two different units in the project organization). The first component is involved with resource utilization monitoring and SLA management, whereas the latter (CHASE) enables the application to integrate autonomic self-optimization features. In the following we will briefly describe the two tools, showing successively how they are integrated in the Cloud@Home platform.

2.1 SLAEngine

Two specific components of the Cloud@Home middleware have been identified in the SLA Engine system. These are the SLA and the QoS subsystems, which ensure the seamless execution of applications (Figure 2a).

SLA Negotiator The SLA Negotiator is responsible for the service lifecycle. This includes service definition, deployment, management, monitoring, compliance and termination. Users call for services that incorporate contracts made up of business terms, with no reference to raw resources. Service invocations are enriched with SLAs, adopting languages such as SLAng[3]. The system manages the service by configuring and deploying it automatically. In order to meet the contract terms, the system is monitored and the SLAEngine takes any required action, such as the re-configuration or re-allocation of resources. Finally, when



(a) QoS/SLA and Virtual environment architecture

(b) interactions with the C@H infrastructure

Fig. 2: Cloud@Home SLA and QoS subsystems

the service fruition finishes, the system frees the resources. It should be noted that the ultimate goal for the infrastructure is the maximization of the number of services executed. The SLA Negotiator needs to monitor service performance levels, in order to detect dynamically contract violations.

QoS service The QoS service is responsible for the management of the cloud resources and the services needed to achieve the application requirements established by the SLA negotiator. It translates the application requirements (expressed in terms of high-level parameters such as time execution, throughput, transaction rate) into low-level criteria related to computing, storage and network distributed resources. The QoS service also has an important role during the negotiation phase, carried out by the SLA Negotiator. In fact, it acts as an estimator able to predict the computational load generated by applications and the corresponding performance obtained.

Fig. 2b shows the communication between the SLA Negotiator and the QoS service. It is possible to identify two different sub-modules in the QoS service: the *QoS director* and the *QoS manager*. The former accepts requests from the SLA Negotiator and invokes the QoS manager to submit applications with appropriate QoS criteria, in terms of computing and network services. In other terms, the QoS director translates the high-level QoS specifications of the SLA Negotiator into the low level requirements (in terms of raw resource requests) for the QoS manager. Moreover, the QoS director has to inform the SLA Negotiator about the estimated processing time of a particular application.

In order to perform the right choices and to provide correct information to the SLA Negotiator, the QoS director has to estimate the computational, storage and network resources needed for the execution of the applications. This goal can be accomplished querying the cloud information system to check the availability of the requested resources.

2.2 The CHASE Component

CHASE (Cloud@Home Autonomic Service Engine) is based on the MAWeS framework, which was developed to support the predictive autonomicity in web service-based architectures [6]. MAWeS itself leverages on the MetaPL language [9] and the HeSSE simulation environment [10]. The first is used to describe the software system and the interactions inside it; the latter, to describe the system behaviors and to predict performances using simulation. Following a bottom-up approach, we will briefly describe the features of this tools and then show at the end of the section how CHASE operates on top of them.

HeSSE. HeSSE is a simulation tool that allows to simulate the performance behavior of a wide range of distributed systems for a given application, under different computing and network load conditions. It makes it possible to describe distributed heterogeneous systems by interconnecting simple components, which reproduce the performance behavior of a section of the complete system.

MetaPL. MetaPL is an XML-based meta-language for parallel program description and prototyping [9, 4]. It provides a core language that can be extended through *Extensions* (XML DTDs) to support different programming paradigms and to enrich the semantics of the description. A MetaPL program can be processed by *filters* (XSLT transformations) to produce different program *views*: one of these set of filters can produce traces to be simulated in HeSSE, thus providing reliable performance predictions even at the early phases of software development [8].

MAWeS. The MAWeS framework relies on MetaPL to run HeSSE simulations and to obtain performance data. The user can specify through the autonomic MetaPL language extension [5, 6], the set of parameters that can be modified by the optimization engine. MAWeS will automatically perform a set of simulations varying the values of these parameters to find the set of values that optimizes the software execution according to one or more criteria (e.g., shortest execution time).

MAWeS is structured in three layers (Figure 3). The *MAWeS frontend* includes a standard client application interface, `MAWeSclient`, providing the general services that can be used and extended to develop new applications. The *MAWeS Core* exploits environment services (i.e., the services offered by the environment to monitor and to manage itself) and the *MetaPL/HeSSE WS interface*, using the application information contained in the MetaPL description to find out optimal execution conditions.

CHASE. It is now easy to illustrate how the Cloud@Home Autonomic Service Engine operates. CHASE acts as a special client of MAWeS. It hands in to MAWeS the configuration of the cloud system (in a format suitable for HeSSE) and of the service/application (in MetaPL format) and receives in response the optimal configuration parameters. In order to perform these tasks, the MAWeS framework must also be extended to predict the behavior of virtualized applications, which is an ongoing work within the Cloud@Home project.

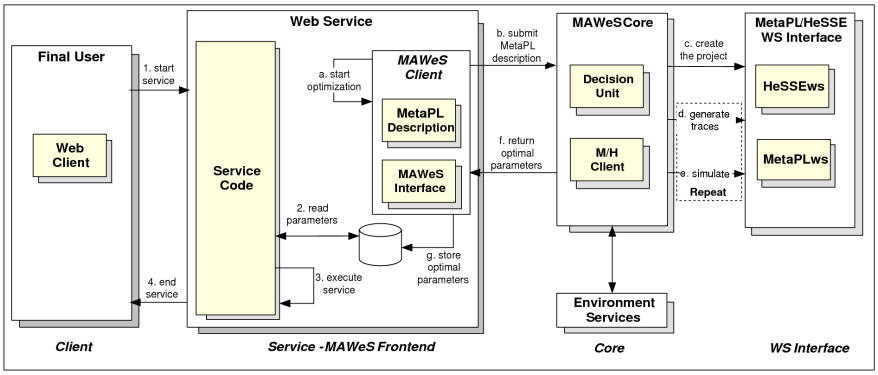


Fig. 3: The three layers of the MAWeS Framework. The client functionalities in the Frontend can be integrated in applicative software

3 C@H Component Integration

SLAEngine and CHASE have similar aims, but follow different approaches:

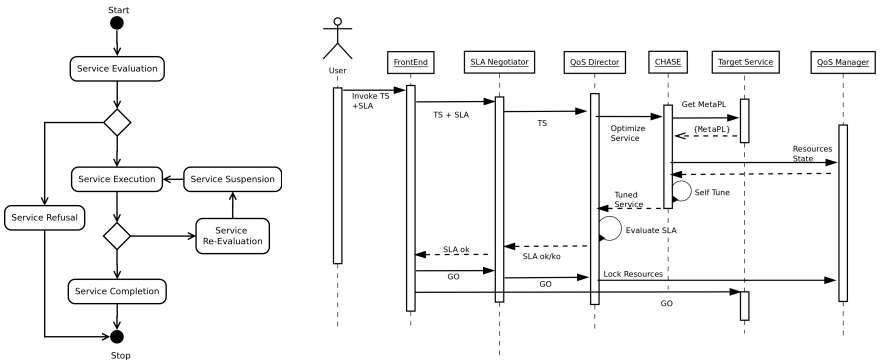
- **SLAEngine** accepts a task submission enriched by a SLA description, which describes the application requirements. The SLA Engine accepts, refuses or negotiates it, deploys the needed resources and monitors the application execution, in order to grant that SLA requirements are respected.
- **CHASE** accepts a task submission enriched by a MetaPL description, which describes the application behavior together with application optimization parameters, predicts the application execution performance in different configurations, and returns the optimal application configuration.

In fact, the two components focus on different characteristics of the application execution. The SLA Engine takes into account the resource usage optimization and so aims at granting resources quality and availability (i.e., reserving a network with granted bandwidth to an application). On the other hand, CHASE focuses on application execution optimization, choosing the suitable resources that can lead to the optimization of user-oriented application performance indexes (e.g., response time), even when the application behavior changes (i.e., there is a variation of the number of processes making up the task or of the way in which data are partitioned). The two approaches are orthogonal: SLAEngine can change the resources available to the application, but cannot change the application behavior. CHASE does not directly manage resources (just asks for them), but it, through MetaPL, knows which parameters affect the application behavior and it is able to change them optimally. In Cloud@Home the two functionalities are integrated in order to get the advantages of both approaches.

In order to clarify how the invocation of a service behaves, Figure 4a illustrates the SLA-based service invocation process. Just after the invocation, an evaluation step takes place (the Service Evaluation activity), in order to predict

the best configuration in terms of both resource usage and application behavior. If the result respects the SLA requirements, the Service Execution activity will follow; otherwise, a negotiation step driven by the SLA Engine will take place. In the current state of the project, the negotiation phase is absent, and so the service invocation is possibly rejected (Service Refusal). During service execution, a warning may occur if a SLA requirement is nearly to be violated (i.e., a resource crashes, an unpredicted load is detected, ...). In this case, the service is re-evaluated, taking into account the actual resource state, and possibly suspended and re-executed with a new resource utilization scenario.

The CHASE-SLAEngine integration implements the above described behavior, as shown in Figure 4b. The user invokes the service through the C@H Frontend. This starts up the request collecting both the SLA request and the application behavior description (for simplicity's sake, the diagram does not show the user/Frontend interaction). The service invocation is redirected to the SLA Negotiator and to the QoS Director. This latter is integrated in CHASE, and asks it for an optimal application behavior. CHASE makes its prediction by evaluating a set of different resource configurations and finding the optimal one, than returns the result to the QoS director which evaluates the SLA requirements and checks if they are respected. If so, the result is returned to the SLA Negotiator and to the FE, which finally starts up the service.



(a) C@H service invocation (b) SLAEngine-CHASE Interaction

Fig. 4: Cloud@home SLAEngine-CHASE Interaction

4 Conclusions and Future Work

The adoption of the volunteer computing paradigm poses problems related to unreliability and low performance. In the Cloud@Home project, whose objective is to build a cloud environment based on volunteer resources, we propose two components that jointly aim at supporting SLA/QoS requirements. One of them

focuses on resource utilization, the other on application behavior. In this paper we have shown how both components work, and how it is possible to integrate them in order to obtain a very flexible system for SLA/QoS management. We will apply the proposed approach and evaluate it in the project case studies.

References

1. Fontán, J., Vázquez, T., Gonzalez, L., Montero, R., Llorente, I.: OpenNebula: The open source virtual machine manager for cluster computing. In: Open Source Grid and Cluster Software Conference (2008)
2. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the use of cloud computing for scientific workflows. In: eScience, 2008. eScience'08. IEEE Fourth International Conference on. pp. 640–645. IEEE (2009)
3. Lamanna, D., Skene, J., Emmerich, W.: Slang: A language for defining service level agreements. In: Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems-FTDCS. pp. 100–106 (2003)
4. Mancini, E., Mazzocca, N., Rak, M., Villano, U.: Integrated tools for performance-oriented distributed software development. In: Proc. SERP'03 Conf. vol. 1, pp. 88–94. USA (Jun 2003)
5. Mancini, E., Rak, M., Torella, R., Villano, U.: A simulation-based framework for autonomic web services. In: Proc. of the Eleventh International Conference on Parallel and Distributed Systems. pp. 433–437. Fukuoka, Japan (Jul 2005)
6. Mancini, E., Rak, M., Torella, R., Villano, U.: Predictive autonomicity of web services in the MAWeS framework. Journal of Computer Science 2(6), 513–520 (2006)
7. Mancini, E.P., Rak, M., Villano, U.: Perfcloud: Grid services for performance-oriented development of cloud computing applications. In: Proc. of Emerging Technologies for Next generation GRID (ETNGRID-2009/WETICE-2009). pp. 201–206 (2009)
8. Martino, B.D., Mancini, E., Rak, M., Torella, R., Villano, U.: Cluster systems and simulation: from benchmarking to off-line performance prediction. Concurrency and Computation: Practice and Experience 19(11), 1549–1562 (2007)
9. Mazzocca, N., Rak, M., Villano, U.: The MetaPL approach to the performance analysis of distributed software systems. In: Proc. of 3rd International Workshop on Software and Performance (WOSP02). pp. 142–149. IEEE Press (2002)
10. Mazzocca, N., Rak, M., Villano, U.: The transition from a PVM program simulator to a heterogeneous system simulator: The HeSSE project. In: J. Dongarra et al. (ed.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, LNCS. vol. 1908, pp. 266–273. Springer-Verlag, Berlin (DE) (2000)
11. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. pp. 124–131. IEEE Computer Society (2009)
12. Peter Mell and Tim Grance: The nist definition of cloud computing (2009)
13. Puliafito, A., et al.: The Cloud@Home Project: Towards a New Enhanced Computing Paradigm. In: Euro-Par 2010 Workshops - Parallel Processing (2010), this volume
14. Tusa, F., Paone, M., Villari, M., Puliafito, A.: CLEVER: A cloud-enabled virtual environment. In: Computers and Communications (ISCC), 2010 IEEE Symposium on. pp. 477–482. IEEE (2010)