Enhancing an Autonomic Cloud Architecture with Mobile Agents*

A. Cuomo[1], M. Rak[2], S. Venticinque[2], and U. Villano[1]

[1] Università del Sannio, Benevento, Italy
{antonio.cuomo,villano}@unisannio.it
[2] Seconda Università di Napoli, Aversa (CE), Italy
{massimiliano.rak,salvatore.venticinque}@unina2.it

Abstract. In cloud environments application scheduling, i.e., the matching of applications with the resources they need to be executed, is a hot research topic. Autonomic computing provides viable solutions to implement robust architectures that are enough flexible to tackle scheduling problems. CHASE is a framework based on an autonomic engine, designed to optimize resource management in clouds, grids or hybrid cloud-grid environments. Its optimizations are based on real-time knowledge of the status of managed resources. This requires continuous monitoring, which is difficult to be carried out in distributed and rapidly-changing environments as clouds. This paper presents a monitoring system to support autonomicity based on the mobile agents computing paradigm.

1 Introduction

Cloud computing [15] is evolving at a steady pace, putting itself forward as a convenient way for structuring and deploying applications and infrastructures both in the commercial and in the academic world. The crosscutting nature of the cloud paradigm ("everything as a service") has promoted its diffusion into many different areas of computer engineering, ranging from applications (Software-as-a-Service) to development platforms (Platform-as-a-Service) to the provisioning of basic computing resources as processing units, storage and networks (Infrastructure-as-a-Service). It could be argued that the very basic concept of "resource", whether it is a piece of software, a developing platform or a hardware component, has changed due to the use of resource *virtualization*, the driving technological force behind the cloud paradigm. Virtualization of resources brings new opportunities (e.g., the acquisition of virtually infinite computing elements) which are partly balanced by new issues and challenges (assessing the effect of virtualization on system performance, extracting information on the physical resources "hidden" behind a virtual interface, ...).

* The work described in this paper has been partially supported by the MIUR-PRIN 2008 project "Cloud@Home: a New Enhanced Computing Paradigm" and by the FP7-ICT-2009-5-256910 (mOSAIC) EU project.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 94–103, 2012.

In the HPC context the current trend is the integration of grids and clouds into unified architectures [9,11,2,17]. To this aim, we have recently proposed *cloudgrid*, a novel architecture for the integration of clouds and grids, and developed its implementation, PerfCloud [13,4]. Among other things, PerfCloud makes it possible to set up a cloud-based provision of computing resources taken from an existing grid. The leased virtual resources are usually organized in virtual clusters, which are automatically integrated in the underlying grid [4].

One of the main open issues in hybrid cloud-grid architectures is *application scheduling*, that is, the matching of applications to the resources they need to be executed. There exists a wide and consolidated body of literature dealing with the scheduling of applications in grids, which is a complex problem by itself. But, nowadays, schedulers should also address the highly dynamic nature of hybrid cloud-grid architectures.

We are currently working on a scheduler which leverages the principles of *autonomic computing* to add self-optimization capabilities to hybrid cloud/grid architectures [18]. Autonomic computing [10] has emerged as a paradigm able to cope with complex and rapidly-mutable environments. Currently, high research effort is directed to investigate how to design cloud and grid environments endowed with autonomic management capabilities [3,11,16]. CHASE, our prototype implementation of autonomic engine, has been designed to tackle the scheduling problem in PerfCloud. However, its design is not tied to the architecture of a *cloudgrid*, but it is sufficiently general to be integrated with any cloud, grid or hybrid cloud-grid environment.

This paper takes a further step towards the definition of an autonomic cloud architecture by showing how some common tasks which are required in this architecture, namely system configuration mining and monitoring may be easily achieved by employing *mobile agents*. Mobile agents are the last evolution of mobile code-based systems. They add mobility to the well-known and largely appreciated features of ordinary software agents systems, such as reactivity, proactivity, communication and social ability. In essence, a mobile agent is a program, which is able to migrate across the network bringing over its own code and execution state. To support our autonomic architecture, a set of specialized agents has been devised that can extract information about the system configuration and perform monitoring tasks.

The rest of the paper is organized as follows. First of all, a synthetic overview of the architecture of CHASE, originally presented in [18], is given, focusing on the requirements for extracting system information and monitoring. Then the mobile agent platform is presented, showing how it can satisfy these requirements. A concrete use case is dealt with in section 4. After a presentation of related work, the paper closes with our final considerations and a sketch of future work.

2 An Autonomic Engine for Managing Clouds

The CHASE architecture (Figure 1) has been described in [18]. At its core, it is a resource manager whose choices are driven by the predictive evaluation of the application performance for different possible resource assignment scenarios. Performance predictions are obtained by simulation of the application for a given resource assignment by means of a discrete-event simulator which acts on the basis of information both static (essentially, the system configuration) and dynamic (system load).

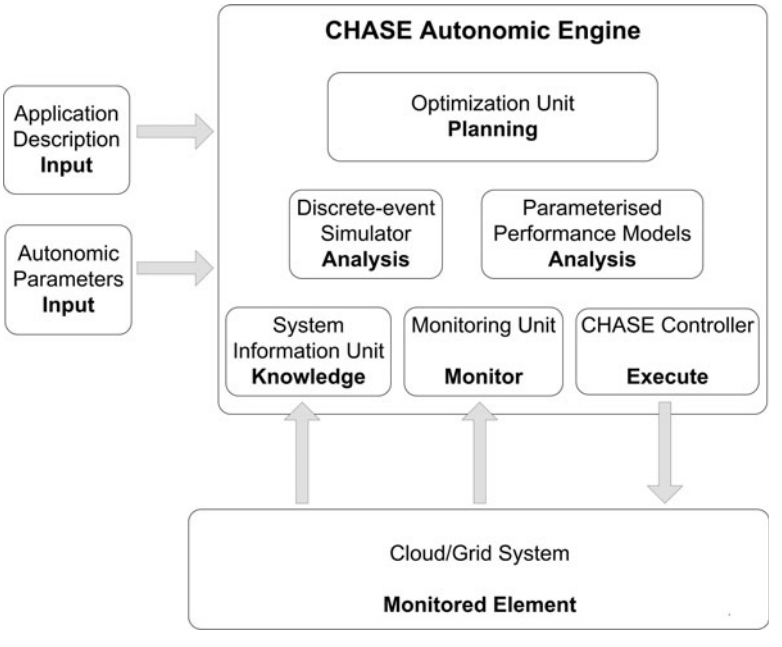

Fig. 1. CHASE architecture

The engine is designed as an autonomic manager capable of controlling different elements as clouds or grids. Figure 1 shows the logical mapping from the CHASE components to the corresponding autonomic building blocks described in [10]. The **Input** to the autonomic engine consists of an *Application Description*, which is a synthetic representation of the application behavior used for driving the simulations, and one or more *Autonomic Performance Parameters*, which can be application performance goals, or constraints on the selection of the resources. The input section is more thoroughly described in [18].

The architectural building blocks are:

– **Planning**, consisting of the Optimization Unit (OU). This is the "smart" component of the architecture. It selects a minimal set of possible resource assignments that have to be simulated in order to find the best one.

- **Analysis**, which is the component capable of performing system evaluation. For a given system configuration, the performance prediction is performed by the Discrete-event Simulator. This is a Java-based prototype that represents the evolution of our previous heterogeneous system simulator, HeSSE [14].
- **Knowledge**, the module responsible for obtaining information about the configuration of the controlled system. We will generally call such information *resource metadata*. The Knowledge module is implemented by the System Information Unit. Subsection 2.1 introduces the requirements for this module, whereas the next section introduces the proposed solution based on mobile agents.
- **Monitor**, which is implemented by a monitoring unit responsible for obtaining dynamic information about the system. The unit can possibly generate alerts when failures or degradations that may cause violations of performance contracts have occurred or are about to occur, thus making it possibly to apply countermeasures.
- **Execute.** The Execute module, implemented by the CHASE Controller, is the system actuator. It is interfaced with the underlying cloud/grid platform, and translates the devised resource assignment plan into actual cloud resource management commands (e.g., virtual machine creation/start/stop, application launch/termination).

2.1 Gathering Resource Metadata

To perform the predictions, the autonomic engine needs information about the the resources of the underlying cloud/grid system. These *resource metadata* can be classified in two macro areas: system configurations and system benchmarks figures. Table 1 reports the current organization of these metadata, as used by our autonomic system.

System configuration is the set of resource parameters that describe how a specific element (system, node, network) is configured. This comprises, for example, frequencies and microarchitecture for the computing elements or latencies and bandwidths for the networks.

System benchmark figures are selected measurements of the system. The outcome of the benchmarks are used to tune the simulator for accurate performance prediction.

The main requirements for resource metadata gathering are:

- *Flexible Structure.* It is not easy, neither desirable, to impose a rigid structure to the classes of information that must be gathered. As an example, new hardware may be introduced that has not been taken into account in the system configuration specification.
- *Mining Capabilities.* While some cloud/grid systems directly expose resource metadata, this is not generally true. Even when some metadata are present, they may be not detailed enough to serve as a basis for configuring the simulations. After all, the intrinsic goal of a cloud is to hide the complexity

of the underlying infrastructure. Thus, methods as general as possible must be devised to extract these information directly from the system.
- *Support for Extending the Benchmarks.* Defining once and for all which benchmarks will be supported is not a viable solution. It is widely recognized that no single benchmark can represent how the system will behave with every possible application. Better benchmarks may be defined later in time, and the possibility of dynamically changing the measurements to be performed is of paramount importance.

Table 1. Resource metadata as gathered by the Knowledge Module

System Configuration		
Scope	**Name**	**Description**
System	No. of nodes	Number of nodes that are available to the system
System	Networks	Available networks (e.g. 10Gb Ethernet, Infiniband)
Node	Hypervisor	Virtual machine monitor of a node (e.g. Xen, KVM, VirtualBox, or none for physical nodes)
Node	No. of CPU	Number of CPUs, number of cores per CPU
Node	Amount of memory	Amount of main memory, amount and configuration of cache memories
Node	Network interface cards	Per-node interfaces to the system networks
System Benchmark Figures		
Benchmark Class		**Description**
CPU		Raw computational speed benchmarks (includes FLOPS and MIPS measurements)
Memory		Sequential/Random read/write access, cache access times
Network		Includes latency and bandwidth benchmarks
Disk		File creation/read/write benchmarks

2.2 Requirements for Monitoring

Static configuration of the resources is just a part of the whole picture: measurements are needed to gain knowledge about the current status of the resources. A monitoring infrastructure is required to evaluate at least two parameters: system load and system availability.

System load refers to all the metrics that can be gathered to measure the load of the different resources (CPUs, networks). These data are fed to the simulator to contextualize the performance prediction process to a specific system load (current or future).

System availability comprises data about the liveness of the resources. Periodical checks must be performed to verify that an entire virtual machine or a single application are still up and running.

3 Introducing Mobile Agents

3.1 The mAGDA Toolset

The mAGDA toolset runs on an agent-enabled cloud composed of one or more virtual machines, and accessible by a WSRF/REST interface that is available to all authenticated users. Service binding is statically defined, but service execution can take place on any nodes belonging to the cloud. An agent-enabled cloud consists of a mobile agents platform distributed on different virtual machines which are connected by a real or a virtual network. Figure 2 shows how a client can use a WSRF/REST stub to invoke agent based services. A software agent, to be installed on a front-end, acts as a proxy between the WSRF/REST and agents technology by translating incoming requests into ACL messages (ACL: Agent Communication Language). Specialized agents are distributed in the cloud to perform their specific role. An agent platform is composed of Jade containers. Jade is a FIPA[1] compliant agent technology developed in Java. A container is a peer-to-peer application that supports execution of agents. We will call agent-node a machine that hosts at least an agent container.

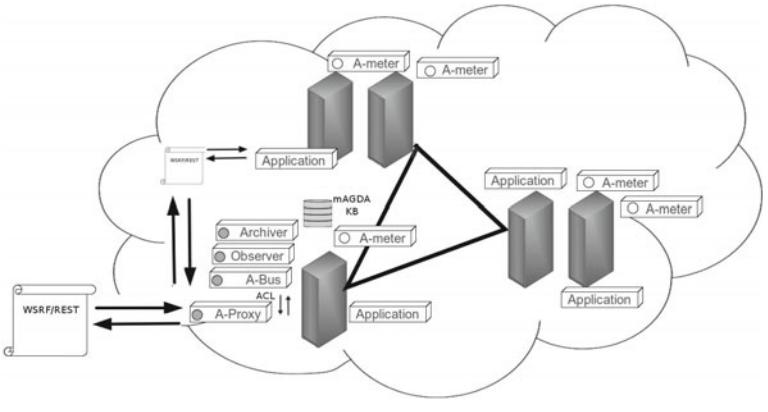

Fig. 2. mAGDA architecture

Any type of agent service can be built on top of this infrastructure: in the following, the set of agent based services that has been conceived to support monitoring and management of cloud resources at infrastructure level will be described. Agents will be in charge of configuring the system, performing measures, computing statistics, collecting and managing information, detecting critical situation and applying reactions according to the decisions notified by the CHASE autonomic engine.

[1] Foundation of Intelligent Physical Agents, `www.fipa.org`

3.2 Supporting the Knowledge Function with mAGDA

The expected output from mAGDA is a profile of the cloud that is composed of a static and a dynamic part. The static part corresponds to resource metadata, as described in section 2.1. It includes information about what are the available resources, how they are connected and their qualitative and quantitative technological parameters. An example could be a virtual machine with 2 dual core x64 CPUs, a given amount of memory and storage, a 100Mb/s network connection and a software installation. The knowledge of the available computing infrastructure is necessary to be aware of the expected application performance and how the resource can be managed optimally. When this information is not made available by the infrastructure, it can be collected by sending agents on the target resource, at start-up or in course of its operation. mAGDA mobile agents can also carry with them benchmark codes, which are executed on the target resource producing benchmark figures which are collected in the cloud profile. The profile also includes information about the configuration of the monitoring infrastructure, as the number of agents, the monitored resources, the active statistics and the controlled performance indexes.

3.3 Supporting the Monitoring Function with mAGDA

Besides the static configuration, mAGDA has to provide a dynamic knowledge about the actual usage of resources. The dynamic part of the cloud profile is updated during the monitoring activity, whose requirements have been described in section 2.2. The monitoring data is composed of statistics of performance parameters which are periodically measured by dedicated agents. To support autonomic behavior, the choice of performance parameters and statistics to be gathered is not fixed, but can be tailored with the configuration of the agent. The monitoring service of mAGDA has been designed as a multi-agent system that distributes tasks among specialized agents. It contemplates both static and mobile agents: the former are responsible for performing complex reasoning on the knowledge base, so they are statically executed where the data reside; the latter usually need to move to the target resources in order to perform local measurements or to get system information. The *Archiver* is a static agent that configures the monitoring infrastructure, collects and stores measurements, computes statistics. According to the parameters to be monitored, the kind of measurement and the provider technology, the archiver starts different *Meters*, which are implemented as mobile agents that the Archiver can dispatch where it needs. *Observers* periodically check a set of rules to detect critical situations. They query the Archiver to know about the statistics and eventually notify applications if some checks have failed. Applications can use a *Agent-bus* service to subscribe themselves for being alerted about each detected event. They can also invoke mAGDA services to start, stop or reconfigure the monitoring infrastructure. Finally, applications can access the complete knowledge base to retrieve information about cloud configuration, monitoring configuration, statistics and the history of past failed checks.

4 An Example Use Case: The Cloud@Home System

Cloud@Home [5] is a project, funded by the Italian Government, which aims at merging the cloud and *volunteer* computing paradigms. Cloud@Home collects infrastructure resources from many different resource providers and offers them through a uniform interface, with an *Infrastructure as a Service* (IaaS) model. The resource providers can range from commercial cloud providers to academic partners, or even to individually volunteered desktop machines. Cloud@Home gives great emphasis to the management of Service Level Agreements (SLAs) and Quality of Service (QoS), and provides dedicated components for these tasks. Through these components, Cloud@Home is capable of performing virtual machine creation on the resource it manages, possibly governed by a resource-oriented Service Level Agreement. CHASE and mAGDA have been designed to enrich the Cloud@Home system with application-oriented performance prediction capabilities and a monitoring infrastructure. A prototype implementation has been realized to verify the effectiveness of the approach. In the prototype scenario, a Cloud@Home installation has been deployed on two academic clusters, PoweRcost and Vega, respectively placed at University of Sannio and Second University of Naples. As a test case, CHASE had to provide an application scheduling for a well-known HPC code, the NAS/MPI 2.4 LU Benchmark, on the IaaS cloud provided by Cloud@Home. This cloud was composed of 8 virtual machines which had been previously obtained from the system. The scheduling goal was the minimization of execution time. The engine workflow went on as described in the following:

1. Input - The performance goal, together with an application description, is fed to the CHASE engine.
2. Knowledge - CHASE instructs mAGDA to launch mobile agents on the C@H virtual machines to recover system configuration. The agents recover information about the structure of the nodes and the network and launch benchmarks to obtain performance data. Most of this information does not vary between executions and is thus stored to allow immediate retrieval in future application requests.
3. Monitoring - CHASE instructs mAGDA to launch mobile monitoring agents on every virtual machines. The agents periodically report information about system load and availability.
4. Planning - CHASE begins selecting possible variations of resource assignments that have to be simulated. Techniques to reduce the number of configurations have been described in [18].
5. Analysis- Every chosen configuration, together with system load data, is fed to the simulator, which reports predicted execution time to the engine.
6. Planning - The planning unit compares the predictions and chooses the best configuration.
7. Execute - The engine schedules the launch of the application on the selected resources.

The correct minimized execution time was selected: results of these execution have been reported in [18].

5 Related Work

A wide body of literature deals with resource management in physical grids [12]. The resource provisioning problem in virtualized systems has been tackled in the Shirako system from Duke University [8] and in the Haizea architecture from University of Chicago/Argonne National Laboratory [19]. Both systems hinge on the concept of *leases*, contracts that define which resources are assigned to users and the duration of these assignments. Compared to these solutions, our engine presents an application centric perspective, where the user has not to provide direct resource requests, but can express his needs in terms of desired application performance. An approach more similar to the CHASE one is used in AppLeS [1], a methodology for adaptive application scheduling on physical Grids. In AppLeS, applications are associated with a customized scheduling agent that monitors and predicts available resource performance an dynamically generates a schedule for the application. Some previous work has been done on the application of mobile code to monitoring and management of resources, mainly in the grid context [20,7]. Few applications of mobile agents to cloud systems are starting to spread, most of which tackle issues different than the ones covered here, like intrusion detection [6] and cloud federation [21].

6 Conclusions

In this paper we have shown how some key parts of an autonomic system can be implemented through the use of mobile agents. These enhancements have been applied to CHASE, an autonomic engine for the development of self-optimizing applications in cloud environments. Two key functions in the CHASE architecture are system configuration mining and resource monitoring. These have been implemented through mAGDA, a mobile agent based platform. A prototypal use case has been developed on the Cloud@Home platform. Future work will focus on testing the engine on a larger set of real applications and cloud platforms.

References

1. Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., et al.: Adaptive computing on the grid using AppLeS. IEEE Transactions on Parallel and Distributed Systems 14(4), 369–382 (2003)
2. Blanco, C., Huedo, E., Montero, R., Llorente, I.: Dynamic provision of computing resources from grid infrastructures and cloud providers. In: Proceedings of the 2009 Workshops at the Grid and Pervasive Computing Conference, pp. 113–120. IEEE Computer Society (2009)
3. Brandic, I.: Towards self-manageable cloud services. In: 33rd Annual IEEE International Computer Software and Applications Conference, pp. 128–133 (2009)
4. Casola, V., Cuomo, A., Rak, M., Villano, U.: The *CloudGrid* approach: Security and performance analysis and evaluation. Future Generation Computer Systems (to be published, 2012), special section: Quality of Service in Grid and Cloud Computing

5. Cuomo, A., Di Modica, G., Distefano, S., Rak, M., Vecchio, A.: The Cloud@Home Architecture - Building a Cloud infrastructure from volunteered resources. In: CLOSER 2011, The First International Conference on Cloud Computing and Service Science, Noordwojkerhout, The Netherlands, May 7-9 (2011)
6. Dastjerdi, A.V., Bakar, K.A., Tabatabaei, S.: Distributed intrusion detection in clouds using mobile agents. In: 2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences, pp. 175–180. IEEE (2009)
7. Di Martino, B., Rana, O.F.: Grid performance and resource management using mobile agents. In: Performance Analysis and Grid Computing, pp. 251–263. Kluwer Academic Publishers, Norwell (2004)
8. Grit, L., Irwin, D., Yumerefendi, A., Chase, J.: Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration. In: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing, p. 7. IEEE Computer Society (2006)
9. Keahey, K., Foster, I.T., Freeman, T., Zhang, X.: Virtual workspaces: Achieving quality of service and quality of life in the grid. Scient. Progr. 13(4), 265–275 (2005)
10. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
11. Kim, H., et al.: An autonomic approach to integrated hpc grid and cloud usage. In: 2009 Fifth IEEE International Conference on e-Science, pp. 366–373. IEEE (2009)
12. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of grid resource management systems for distributed computing. Software: Practice and Experience 32(2), 135–164 (2002)
13. Mancini, E.P., Rak, M., Villano, U.: Perfcloud: Grid services for performance-oriented development of cloud computing applications. In: WETICE, pp. 201–206 (2009)
14. Mazzocca, N., Rak, M., Villano, U.: The Transition from a PVM Program Simulator to a Heterogeneous System Simulator: The HeSSE Project. In: Dongarra, J., Kacsuk, P., Podhorszki, N. (eds.) PVM/MPI 2000. LNCS, vol. 1908, pp. 266–273. Springer, Heidelberg (2000)
15. Mell, P., Grance, T.: The nist definition of cloud computing (2009)
16. Murphy, M.A., Abraham, L., Fenn, M., Goasguen, S.: Autonomic clouds on the grid. Journal of Grid Computing 8(1), 1–18 (2010)
17. Ostermann, S., Prodan, R., Fahringer, T.: Resource Management for Hybrid Grid and Cloud Computing. Computer Communications, 179–194 (2010)
18. Rak, M., Cuomo, A., Villano, U.: CHASE: an Autonomic Service Engine for Cloud Environments. In: WETICE 2011 - 20th IEEE International Conference on Collaboration Technologies and Infrastructures. pp. 116–121. IEEE (2011)
19. Sotomayor, B., Keahey, K., Foster, I.: Combining batch execution and leasing using virtual machines. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing. pp. 87–96. ACM (2008)
20. Tomarchio, O., Vita, L.: On the use of mobile code technology for monitoring grid system. In: CCGRID 2001: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, p. 450. IEEE Computer Society, Washington, DC (2001)
21. Zhang, Z., Zhang, X.: Realization of open cloud computing federation based on mobile agent. In: IEEE International Conference on Intelligent Computing and Intelligent Systems, ICIS 2009, vol. 3, pp. 642–646. IEEE (2009)