CHASE: an Autonomic Service Engine for Cloud Environments

Massimiliano Rak
Seconda Università di Napoli,
Dipartimento di Ingegneria dell'Informazione,
Aversa, Italy
massimiliano.rak@unina2.it

Antonio Cuomo, Umberto Villano
Università del Sannio,
Dipartimento di Ingegneria,
Benevento, Italy
{antonio.cuomo, villano}@unisannio.it

***Abstract*—The rapidly spreading cloud computing paradigm delegates to the network the provision of most resources, even those strictly linked to hardware as storage and CPU time. This approach enables the development of applications which may exploit a variable amount of resources in a flexible way, so as to satisfy the actual load of requests coming from the users. A side effect of such flexibility is that optimization has to be more focused on user-perceived performance indexes than on resource usage. This paper takes a step in this direction, presenting the design and development of CHASE, an autonomic engine designed to optimize the scheduling of virtual machines in a cloud environment. The paper illustrates the CHASE architecture and its application in two different contexts: in PerfCloud, an environment for IaaS provision based on cloud and grid integration, and inside Cloud@Home, a project whose objective is to build a cloud using volunteer-based resources. Some preliminary experimental results based on HPC applications are presented.**

I. Introduction

Since its inception, *cloud computing* [1] 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. As a matter of fact, 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). As a result, potentially the entire design space associated to many engineering problems has to be restructured. Previously accepted solutions may have to be adapted to take into account the nature of cloudified computation. In the HPC context, for example, we see grids and clouds converging into unified architectures where the very basic concept of resource has changed due to the use of resource virtualization, the driving technological force of the cloud paradigm [2], [3], [4], [5], [6]. We have recently proposed *cloudgrid*, a novel architecture for the integration of clouds and grids, and presented its implementation, PerfCloud [7], [8]. 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 [8].

Changing the way in which resources are provisioned has a strong impact on the scheduling of scientific applications, that is, on the matching of applications to the resources they need to execute. Virtualization offers new opportunities (e.g., the acquisition of virtually infinite resources) and poses new challenges (assessing the effect of virtualization on system performance, extracting information on the physical resources "hidden" behind a virtual interface). There exists a vast and consolidated body of literature dealing with the scheduling of applications in grids, which is a complex problem by itself. But nowadays the schedulers should also address the highly dynamic nature of hybrid cloud-grid architectures.

Over the last years, autonomic computing [9], [10] has emerged as a paradigm able to cope with complex and rapidly-mutable environments. Currently research is investigating how to design cloud and grid environments endowed with autonomic management [11], [12], [13]. The work we present here is a step in this direction, as it focuses on the introduction of *self-optimization* capabilities in cloud and grids. This paper introduces CHASE, our prototype implementation of an autonomic engine designed to tackle the scheduling problem in PerfCloud, adding self-optimization capabilities. However, our proposal is not tied to the architecture of a *cloudgrid*, but it is sufficiently general to be integrated with any cloud, grid or hybrid environment.

The rest of the paper is organized as follows. First of all, the proposed autonomic architecture is presented, describing in detail the design of CHASE. Applications of the prototype are then illustrated showing its integration with PerfCloud and its use in an on-going research project. Then the results of our first experimentations are presented, discussing how the NAS Parallel Benchmarks are autonomically scheduled on the *cloudgrid*. The following section sets our work in the context of current literature. The paper closes with our comments on the CHASE architecture and a discussion on future work.

II. Rationale and Architecture of an Autonomic Resource Manager

Traditionally, resource managers adopt a *resource-centric* approach, in that their goal is to manage the system so as to make efficient use of available resources. It is often silently assumed that an efficient utilization of the resources leads to an efficient scheduling of the applications. Since this is not always the case, a mismatch may arise between the system and the application perspective. In this situation, it is up to the application developer to estimate the quantity of resources needed by his application to achieve a given performance level, and to use overprovisioning to mitigate the effect of an inappropriate (from his point of view) scheduling.

Our engine is instead based on an *application-centric* approach. The CHASE scheduler is driven by the predictive evaluation of the application performance for different possible resource assignment scenarios, and explores the solution space until some user-supplied constraint is satisfied. In our system, performance predictions are obtained by simulation of the application for a given resource assignment by means of a discrete-event simulator. Since our focus is primarily on HPC applications, our first tests involve applications made up of independent processes interacting through an MPI-compliant message passing middleware. However, CHASE can be used also in different fields. Most of the times, its use in different contexts requires just a change the user-supplied metric (for transactional software, for example, the completion time used for HPC applications should be replaced by response time or throughput). As will be discussed below, the exploration of the whole solution space involves huge computational complexity, due to exponential growth of the possible combinations of resource assignments and mappings. This entails the need to use “smart” techniques to reduce the number of simulation runs required.

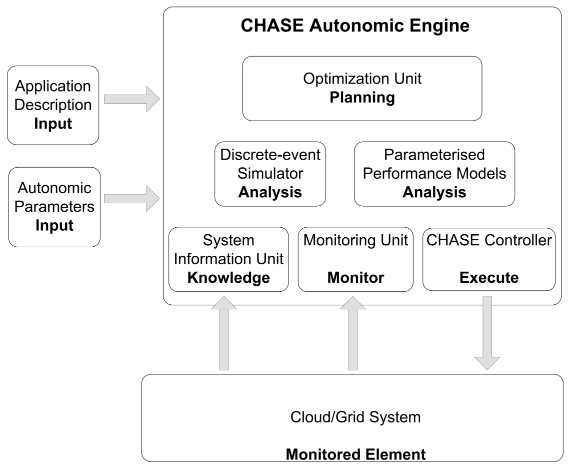

CHASE Autonomic Engine
Application Description Input
Autonomic Parameters Input
Optimization Unit Planning
Discrete-event Simulator Analysis
Parameterised Performance Models Analysis
System Information Unit Knowledge
Monitoring Unit Monitor
CHASE Controller Execute
Cloud/Grid System Monitored Element

Fig. 1: CHASE architecture

Figure 1 highlights the main components of the proposed autonomic system, logically mapped to the corresponding autonomic building blocks described in [9]. Below we will discuss the function of these blocks in an orderly way.

A. Input

Basically, to perform the autonomic prediction service, CHASE requires an *Application Description* and one or more *Autonomic Performance Parameters*. The *Application Description* is a synthetic representation of the application behavior used for driving the simulations. The description captures the fundamental operations (the most relevant as far as performance is involved) performed by the application, and is expressed in MetaPL, an XML-based modeling language whose schema provides support for the main constructs (procedure calls, loops, message-passing operations) found in common scientific languages and libraries.

The choice of a dedicated language requires some additional comment. Most tools in literature tend to work directly on the application code (in binary or source format) or on other automatically-derivable representations like *application traces*, which are ordered sequences of the most relevant events (e.g., CPU bursts, communication events) occurred during a previous execution of the application – possibly on another platform. This approach has the clear advantage that application descriptions can be obtained without requiring additional effort from the application programmer. On the minus side, it is difficult to use application traces to predict the behavior of an application with a different number of processes from the one used during the tracing. Our “artificial” application model, on the other hand, is easily parameterizable in the number of processes or in the number of loop iterations. In fact, this flexibility is also present in the original application source or binary code, but the model straightforwardly obtained from the sources is too complex and is likely to lead to too long simulation times. On the other hand, the MetaPL model can be designed to capture only the essential behavior of the application, discarding the sections of code which have small impact on system performance. Another advantage of a synthetic modeling language is that it can also be used as an initial specification of the application, in the spirit of languages like UML: this minimizes the additional cost of modeling (if, as in good practice of software engineering, an initial model has to be provided in any case) and leads the way to deriving skeleton implementations from the model.

The *Autonomic Performance Parameters* make it possible to specify constraints on the selection of the resources to comply with the application performance goals. Two types of parameters are currently defined:

- *Constraint*, by which the application developer can impose constraints on the selection of the resources. Predefined constraints make it possible to limit such parameters as the total number of nodes to be used, the total number of processes, or the number of processes per node.
- *Target*, which defines the objective function and specifies if it has to be minimized or maximized. The most commonly used target parameters we use are total execution time (minimized) and parallel efficiency (maximized).

B. Planning

The Optimization Unit (OU) is the “smart” component of the architecture. It is able to reduce the number of simulations that have to be performed in order to find the best configuration of resource assignments. It is worth pointing out that the configuration space is almost always of intractable magnitude. Let us consider, for example, a 16-nodes (small) cluster in which every node has 2 CPUs available. Then let us suppose to limit the allocation of processes to nodes to 2, in order to avoid overbooking processors. Even in this simple and heavily constrained scenario, the number of possible allocations is 3^{16} (about 43 millions). It is not clearly feasible to perform such a high number of simulations in time comparable with the "think time" introduced by other schedulers. However, it can be easily recognized that considering every possible combination of resources and allocations brings a considerable amount of redundancy. For example, let us consider a cluster composed of 3 identical machines (say A, B, C), equally loaded, on which we have to allocate 2 processes (P1, P2). We expect that the schedules $\{P1 \rightarrow A, P2 \rightarrow B\}$ and $\{P1 \rightarrow B, P2 \rightarrow A\}$ are substantially equivalent. Similarly, the three schedules $\{P1 \rightarrow A, P2 \rightarrow B\}$, $\{P1 \rightarrow A, P2 \rightarrow C\}$, $\{P1 \rightarrow B, P1 \rightarrow C\}$ are also expected to show identical performance.

The Optimization Unit uses different techniques to exploit the solution space symmetries. First of all, the entire system is modeled as a fully connected graph (fig. 2a), in which every node corresponds to a computational node and the arcs represent virtual transmission links (even if a physical link does not exist, the nodes can communicate through switching or routing). Associated with every node is information as the hostname, the number of CPUs and the current load present on the node. Every arc instead has a weight that represents the estimated latency between nodes. The information needed to build the system graph is contributed by the System Information Unit (described later in this section).

Once the system model is available, the OU starts a multiple-step partitioning of the graph using several different partitioning criteria. The first partitioning is performed by examining communication properties. The nodes whose mutual connection latency is above a certain threshold are put in distinct subsets (fig. 2b). This makes it possible, for example, to identify computer clusters and to put in different subsets the nodes belonging to different clusters. To support execution of HPC applications, first the OU makes the simplifying assumption that a resource assignment can only collect nodes from the same subset. If a subset of the required size has not been found, the latency threshold is raised (this will lead to larger subsets). This operation is possibly repeated until a node subset of the required size is eventually found. The rationale here is that this scheme leads to the resource allocation with the minimum latency between nodes, a key requirement for HPC applications. After that (at least) one suitable set has been found, a second partitioning is made inside each subset grouping together the CPUs similarly loaded (fig. 2c). Once these partitionings have been carried out, the number of alternative configurations to be simulated is drastically reduced, since the nodes in every subset behave in the same way as far as resource assignment is concerned. In other word, picking *any* resource from a node or another inside a given subset leads to similar performance.

C. Analysis

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]. The simulator is able to predict the execution time of distributed message-passing applications executed on heterogeneous nodes. New simulation models have been developed for virtualized applications, taking into account the presence of the hypervisor and the associated overheads. A key issue with the simulator is that it must be able to perform very fast simulations of application execution. This is of paramount importance: if a simulator is designed to support scheduling decisions, it has a limited "think time" available to perform the needed evaluations. Obviously, simulation speed has a price. It is commonly recognized that system simulation trades off speed for accuracy and simulation scalability. Our simulator obtains a lower accuracy (errors between 20% and 30% of measured time) than well known cycle-accurate simulator, which are slower by three or four orders of magnitude. Our simulator makes it possible to model the computational (CPU bursts) and communication sections of an execution run *for different target machines and networks*. This is done through the use of parameterized models both for computing and communication. Computational parameterization is done through the application of a *scaling factor* that accounts for the different performance level between a *reference* machine and the target one. This scaling factor is *not* calculated as a ratio between the frequencies of the processors (recognized in literature as inadequate), but through the execution of different benchmarks on the reference and target machine. Network parameterization requires the execution of benchmarks for calculating actual latency and bandwidth, or more complex parameters as the ones used in the LogP model.

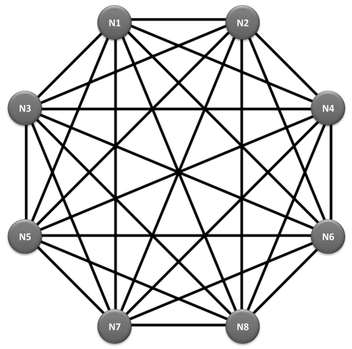
The Performance Models are the pluggable simulator classes that give to CHASE the flexibility needed to operate in heterogeneous environments. The modularity of the simulator makes it possible to substitute the performance models (e.g. network, processing, virtualization model) with small effort.

D. Knowledge

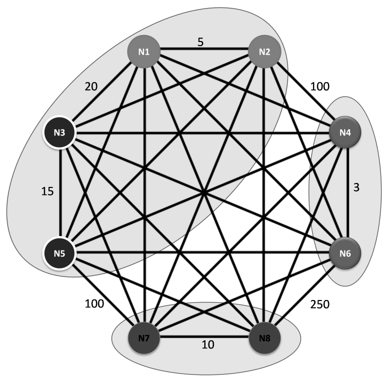
The CHASE System Information Unit (SIU) is responsible for acquiring knowledge about the underlying cloud/grid/cloudgrid system. These informations belong substantially to two classes: resource parameters and benchmark results. The resource parameters are used by the Optimization Unit to build up the system model (see above). The benchmark results are needed by the simulator to predict the execution time on heterogeneous nodes. In the current CHASE prototype, the SIU has been designed to interact with the PerfCloud environment, which provides explicit API calls to export the underlying physical machine configuration and reference benchmark data. Work is in progress to extract a meaningful set of informations from systems which do not expose this information (e.g., commercial clouds).

E. Monitoring

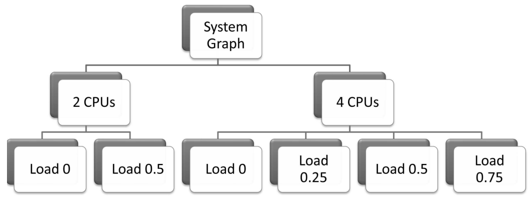
The CHASE Monitoring Unit is responsible for interacting with the cloud or grid monitoring systems. The monitoring



(a) *Before Network Partitioning.*



(b) *After Network Partitioning.*



(c) *Decision Tree for CPU-based partitioning.*

Fig. 2: Partitioning of the System Graph by the Optimization Unit

subsystem is used to inform the other parts of possible failures or degradations that may cause violations of performance contracts, so that countermeasures can be applied. At the state of the art, CHASE performs only performance predictions at application startup time. Once the application has been launched on the autonomically chosen nodes, no further optimization is performed. We are working to extend this component so as to add further autonomic capabilities to the system (e.g., to support load migration).

F. *Execute*

The plan devised by the Optimization Unit (i.e., the list of selected resources on which the application should be launched) is translated into actual cloud resource creation commands by the CHASE Controller. This component is able to interact with the cloud system to operate the creation and launch of virtual machines. Currently, the CHASE controller is able to interact with the PerfCloud API for managing the virtual machine lifecycle, but its design makes it very straightforward to add subclasses for the management of different systems.

III. CHASE IN THE PERFCLOUD FRAMEWORK

PerfCloud [7] is a *cloudgrid* [8] implementation which offers services for the creation and management of virtual computing resources on the top of a computing grid. The resources leased to users by the PerfCloud system are virtual clusters (VC), i.e., sets of preconfigured virtual machines (VMs) connected by a virtualized network. The VCs are automatically integrated in the existing grid. The cloud user has administration rights on the leased VCs, but he cannot manage the physical resources. The PerfCloud architecture can be integrated with CHASE to make up an autonomic IaaS provision with predictable user application performance on the newly instantiated virtual clusters. To illustrate this integration, some details are needed about PerfCloud interface. *PerfCloud* provides a service oriented interface on top of existing virtual machine engines (Xen, KVM, VMWare) and schedulers (OpenNebula[15], Nimbus[16]). It offers resource abstraction (in terms of WSRF [17]) that enables services (and service invokers) to manage directly the functionalities offered by virtualization engines and VM schedulers. As shown in Figure 3, PerfCloud resource representation (hexagons in the picture) can be Virtual Workspaces or an abstraction of VMs and VCs.

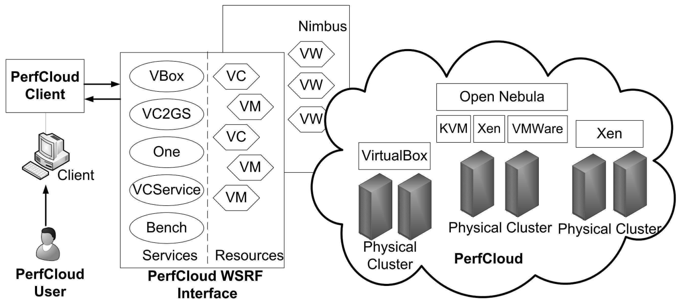


Fig. 3: A particular of the *PerfCloud* Interface

The PerfCloud API offers a large set of services related to different cloud technologies. A summary of some of the basic services is provided here:

- VC2GS, ONE, VCservice, Vbox: they are the main VC interfaces. They implement the Virtual Workspaces WSRF and can be invoked to set up the virtual clusters. The first two interact with Nimbus and OpenNebula services, respectively. VCservice and Vbox are instead standalone, in that they directly control the hypervisors of the physical nodes (Xen and VirtualBox, respectively).
- BenchGS, BenchDB: The first starts up a set of benchmarks on the target VC, which can be useful to measure the performance of the virtual cluster. BenchDB starts up a dedicated virtual machine provided with a MySQL database to collect benchmark results [18].

When CHASE is used on top of PerfCloud, it acts as an intermediary between the client and the PerfCloud service-oriented Interface. This mediation creates an higher level of abstraction than basic IaaS. PerfCloud (or other IaaS systems) makes it possible to ask “get me a virtual cluster made of 1 frontend and 3 nodes”, the enriched CHASE+PerfCloud interface accepts requests of the kind “get me a virtual cluster capable of running this application in less than 2 hours”.

When CHASE receives such a request, it firstly retrieves from PerfCloud the configuration of the system it is managing. Then, it retrieves benchmark data through the PerfCloud BenchService. With this data, CHASE operates the autonomic process described in the previous section, which culminates with the invocation of one of the VC creation services to instantiate the virtual cluster.

IV. CHASE in the Cloud@Home System

Cloud@Home is a project funded by the Italian Government. It aims at merging the cloud and *volunteer* computing paradigms. Cloud@Home [19] collects infrastructure resources from many different resource providers and offers them through a uniform interface, in a *Infrastructure as a Service* (IaaS) perspective. These resource providers can range from commercial cloud providers to academic partners, or even to individually volunteered desktop machines. Cloud@Home puts great emphasis on the management of Service Level Agreements (SLAs) and Quality of Service (QoS) and provides dedicated components for that (respectively, the SLA Manager and the Resource&QoS Manager, RQM in short). Inside the Cloud@Home Architecture, CHASE is a relevant component used during the service negotiation phase: when the Cloud@Home SLA Manager needs to evaluate the sustainability of user requests, it transmits to the RQM component a formalized description of the QoS Level explicitly requested by the user. The RQM forwards this description to the CHASE engine, along with an application description and information about the current state of system, in terms of resource availability and load. CHASE performs the autonomic prediction and informs the RQM whether a configuration matching the requested QoS has been found or not.

V. Experiments

As a first experiment for assessing the capabilities of CHASE, we have deployed its prototype implementation together with a PerfCloud installation. In the testbed, PerfCloud "cloudifies" an 8-nodes cluster in which each workstation is powered by 1 Intel Xeon "Nocona" running at 2.80 Ghz, with 1MB L2 cache. A MetaPL description of the NAS Parallel Benchmark LU [20] has been manually derived. A first execution of CHASE is done submitting this description and asking the system to minimize the execution time. A second execution was made with the constraint of using at least 4 nodes and the maximization of parallel efficiency as objective. For comparison, measurements are taken from real executions of LU (from NPB version 2.4). Logging was activated, to make CHASE print the evaluation of all the tried configurations, instead of printing just the chosen one. Table I shows the results. The configuration selected in the first execution was confirmed from direct measurement to be the one with lowest execution time. In the second run, the selected configuration was again confirmed to be the correct one. The prediction error is very low, which is an encouraging result, considered that the LU benchmark is not a toy example but an HPC application that leverages all the principal machine subsystems.

VI. Related Work

A wide body of literature deals with resource management in traditional (i.e., non-virtualized) grids. A survey and a taxonomy of such management systems is presented in [21]. The resource provisioning problem in virtualized systems has been tackled in the Shirako system from Duke University and in the Haizea architecture from University of Chicago/Argonne National Laboratory. Both systems hinge on the concept of *leases*, contracts that define which resources are assigned to users and the duration of these assignments.

Shirako [22] is a flexible leasing infrastructure with models for controlling different leasable resources, including virtual machines. As compared to our CHASE system, Shirako takes into account different optimization issues, such as the sizing of the *sliver* (the quantity of node resources assigned to a specific virtual machine) and the possibility to migrate virtual machines. It provides autonomic capabilities to "turn the knobs" that control these two aspects. It does not provide an infrastructure for startup performance prediction.

Haizea [23] is a resource manager that integrates advance reservations with batch scheduling. To support advance reservations (the possibility to reserve usage of resources in a future time interval), a batch scheduler must eventually suspend the best-effort jobs running when the reservation time is approaching. This is a complex task that requires the use of a checkpoint infrastructure. The key idea in Haizea is to ease this process by substituting the concept of job (both best-effort and reserved in advance) with leases of virtual machines, which are easily suspendable and resumable by their nature. Currently Haizea does not support autonomic adaptive behavior during the execution of the application. A substantial difference with our design is that it is based on a resource-centric perspective: the submitting user must know which resources he needs to run the application. In contrast, CHASE takes an application-centric perspective: the user submits the application, together with some performance parameters that must be respected, and the system decides the quantity of resources to be assigned to the application.

An approach more similar to the CHASE one is used in AppLeS [24], a methodology for adaptive application scheduling on computational Grids. Applications are associated with a customized scheduling agent that monitors available resource performance and generates dynamically a schedule for the application. In common with our design, AppLeS agents are capable of performing performance predictions, based on the Network Weather Service [25]. Since AppLeS is targeted to physical grids, it does not provide any support for modeling resource virtualization.

VII. Conclusions

In this paper we have introduced CHASE, an autonomic framework for the development of self-optimizing applications in cloud environments. CHASE uses a simulation-based approach to predict application performance behavior both in terms of resource usage and of response time. Applications are preliminary described in a high level language. We have

TABLE I: CHASE evaluations vs. direct measurements for LU NAS Parallel Benchmark

		Execution time			Parallel Efficiency		
#Nodes	#VMs per Node	Measured	Predicted	Error	Measured	Predicted	Simulation time
1	1	3222.40	3089.77	-4.11%	1	1	0.18
2	1	1729.31	1673.60	-3.22%	0.93	0.923	3.61
4	1	928.30	965.79	3.93%	0.87	0.80[2]	12.21
8	1	542.55	552.85[1]	1.90%	0.74	0.70	31.17

[1] Chosen with: Target = min{execution Time}
[2] Chosen with: Constraint={numNodes >= 4} and Target = max{parallelEfficiency}

presented the architecture of this autonomic engine, clarifying the use case scenarios and the models developed. The target applications considered till now are explicitly parallel HPC codes, even if the approach is in principle more general, and not constrained to them.

We have shown how to integrate the autonomic engine in two different contexts: PerfCloud, a software for IaaS provision based on cloud and grid integration, and Cloud@Home, a project whose aim is to build up a cloud provider using volunteer-based resources. We have presented some preliminary experimental results obtained using a real-world HPC application (the NAS LU benchmark). These point up correctness of the approach used, even if they cannot be considered a full validation of the solution.

Our future work will focus on a complete validation of the approach, adopting a large set of real applications, on the optimization of the prediction module (simulator) and the experimentation of the approach on non-HPC applications.

Acknowledgements

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.

REFERENCES

[1] P. Mell and T. Grance, "The nist definition of cloud computing," http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc, 2009.

[2] K. Keahey, I. T. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the grid," *Scient. Progr.*, vol. 13, no. 4, pp. 265–275, 2005.

[3] S. AjayKumar, C. Nachiappan, and K. Periyakaruppan, "Enhancing Portable Environment using Cloud and Grid," *Computing*, pp. 728–732, 2009.

[4] C. Blanco, E. Huedo, R. Montero, and I. Llorente, "Dynamic provision of computing resources from grid infrastructures and cloud providers," in *Proceedings of the 2009 Workshops at the Grid and Pervasive Computing Conference*. IEEE Computer Society, 2009, pp. 113–120.

[5] R. Buyya and R. Ranjan, "Special section: Federated resource management in grid and cloud computing systems," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1189–1191, 2010.

[6] S. Ostermann, R. Prodan, and T. Fahringer, "Resource Management for Hybrid Grid and Cloud Computing," *Computer Communications*, pp. 179–194, 2010.

[7] E. P. Mancini, M. Rak, and U. Villano, "Perfcloud: Grid services for performance-oriented development of cloud computing applications," in *WETICE*, 2009, pp. 201–206.

[8] V. Casola, A. Cuomo, M. Rak, and U. Villano, "The *CloudGrid* approach: Security and performance analysis and evaluation," *Future Generation Computer Systems*, 2012, special section: Quality of Service in Grid and Cloud Computing. To be published.

[9] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," in *Computer*, vol. 36, num. 1. IEEE Computer Society Press, 2003, pp. 41–50.

[10] IBM Corp., *An architectural blueprint for autonomic computing*. USA: IBM Corp., October 2004, www-3.ibm.com/ autonomic/pdfs/ ACBP22004-10-04.pdf.

[11] I. Brandic, "Towards self-manageable cloud services," in *2009 33rd Annual IEEE International Computer Software and Applications Conference*. IEEE, 2009, pp. 128–133.

[12] H. Kim *et al.*, "An autonomic approach to integrated hpc grid and cloud usage," in *2009 Fifth IEEE International Conference on e-Science*. IEEE, 2009, pp. 366–373.

[13] M. Murphy, L. Abraham, M. Fenn, and S. Goasguen, "Autonomic clouds on the grid," *Journal of Grid Computing*, vol. 8, no. 1, pp. 1–18, 2010.

[14] N. Mazzocca, M. Rak, and U. Villano, "The transition from a PVM program simulator to a heterogeneous system simulator: The HeSSE project," in *Recent Advances in PVM and MPI, Lecture Notes in Computer Science*, J. Dongarra et al., Ed., vol. 1908. Berlin (DE): Springer-Verlag, 2000, pp. 266–273.

[15] Distributed Systems Architecture Research Group, "Opennebula project," Universidad Complutense de Madrid, Tech. Rep., 2009, http://www.opennebula.org.

[16] University of Chicago, "Nimbus project," 2009, http://workspace.globus.org/clouds/nimbus.html.

[17] T. Banks, "Web Services Resource Framework (WSRF)–Primer v1. 2," *OASIS Committee Draft*, 2006.

[18] M. Rak, A. Cuomo, and U. Villano, "A Service for Virtual Cluster Performance Evaluation," in *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*. IEEE, 2010, pp. 249–251.

[19] A. Cuomo, G. Di Modica, S. Distefano, M. Rak, and A. Vecchio, "The Cloud@Home Architecture - Building a Cloud infrastructure from volunteered resources," in *CLOSER 2011, the First International Conference on Cloud Computing and Service Science, 7-9 May 2011, Noordwojkerhout, the Netherlands*, 2011, accepted for publication.

[20] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber *et al.*, "The NAS parallel benchmarks," *International Journal of High Performance Computing Applications*, vol. 5, no. 3, p. 63, 1991.

[21] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, no. 2, pp. 135–164, 2002.

[22] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "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*. IEEE Computer Society, 2006, p. 7.

[23] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *Proceedings of the 17th international symposium on High performance distributed computing*. ACM, 2008, pp. 87–96.

[24] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf *et al.*, "Adaptive computing on the grid using AppLeS," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, no. 4, pp. 369–382, 2003.

[25] R. Wolski, N. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 757–768, 1999.