Computer Networks 57 (2013) 795–810

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Mechanisms for SLA provisioning in cloud-based service providers

Emiliano Casalicchio [*], Luca Silvestri

Department of Civil Engineering and Computer Science Engineering, University of Roma "Tor Vergata", Roma, Italy

ARTICLE INFO

Article history:
Available online 24 November 2012

Keywords:
Cloud computing
Quality of service
Service level agreement
Resource provisioning
Autonomic computing

ABSTRACT

A challenge in cloud resource management is to design self-adaptable solutions capable to react to unpredictable workload fluctuations and changing utility principles. This paper analyzes the problem from the perspective of an Application Service Provider (ASP) that uses a cloud infrastructure to achieve scalable provisioning of its services in the respect of QoS constraints.

First we draw a taxonomy of IaaS provider and use the identified features to drive the design of four autonomic service management architectures differing on the degree of control an ASP have on the system. We implemented two of this solutions and related mechanism to test five different resource provisioning policies. The implemented testbed has been evaluated under a realistic workload based on Wikipedia access traces on Amazon EC2 platform.

The experimental evaluation performed confirms that: the proposed policies are capable to properly dimension the system resources making the whole system self-adaptable respect to the workload fluctuation. Moreover, having full control over the resource management plan allow to save up to the 32% of resource allocation cost always in the respect of SLA constraints.

1. Introduction

The extreme complexity of cloud systems calls for novel adaptive management solutions for scalable, maintainable, cost-effective cloud provision, at all software stack layers. The management of cloud infrastructures and services can be operated from different perspectives having in mind opposite or at least conflicting goals. Cloud providers are interested in optimizing resources usage and costs. On the contrary, the customer of cloud services, for example an Application Service Provider (ASP), wants to minimize the cost payed to buy virtualized resources (e.g. computational power, storage, and network bandwidth) and, at the same time, it would be capable to properly size the outsourced resources in a way that allows to honor the service levels agreed with its clients.

One of the promises of cloud computing is to facilitate ASPs in starting up and providing their services avoiding the costs to build, manage and maintain a data center infrastructure. Therefore, to realize this promise it is fundamental to investigate mechanisms (models, architectures and policies) to support QoS provisioning at service level. The service provisioning (or application-level resource provisioning) problem [1–14], that we address in this paper, is related to the management of cloud resources from the infrastructure/platform customers (e.g. ASP) viewpoint. A complementary problem, out of the scope of this paper, is the dynamic QoS provisioning at infrastructure and platform level [3,15,16,9,17–23]. This problem consider the cloud provider viewpoint in managing the infrastructure resources to satisfy SLAs contracted with the infrastructure and/or platform customers, maximizing its revenue. A common goal of the researches framed in these two research fields is to provide solutions to enable self-adaptation.

In foregoing research [24,25] we studied Infrastructure as a Service (IaaS) providers features enabling the

* Corresponding author. Tel.: +39 0672597732.
E-mail addresses: casalicchio@ing.uniroma2.it (E. Casalicchio), silvestri@ing.uniroma2.it (L. Silvestri).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.10.020

implementation of mechanism for the autonomic management of virtualized resources and we formulated the ASP resource management as an optimization problem, proposing both reactive and proactive heuristic policies that approximate the optimal solution. The main results of our analysis were that IaaS providers do not deliver all services necessary to implement an autonomic service management solutions, neither deliver integrated autonomic service management solutions. Moreover, services that enable autonomic resource management have a low degree of customization. From the results of this analysis we proposed four architectures (named Extreme ASP control, Full ASP control, Partial ASP Control, and Limited ASP control) that allow to implement autonomic management of cloud resources and that provide different degrees of customization and control over the autonomic cycle phases [26]. The formulation of the ASP resource management as an optimization problem let us to define optimal and suboptimal resource allocation policies, tested by means of simulation.

In this paper we study mechanisms for autonomic service provisioning focusing our attention on: (i) architectures supporting self adaptation and (ii) algorithms to plan the system adaptation. The proposed mechanisms has been implemented in a prototype and tested using Wikibench [27], a web hosting benchmark that generates real traffic by replaying real workload obtained from Wikipedia access traces. Although there are recent works [13,28,14] discussing architectural solutions for autonomic service provisioning, none of them jointly compared architectural solutions and planning policies, as we do in this work.

In this paper we extend our previous research as follows:

- We refined our survey on features and services offered by seventeen of the most known IaaS public providers (see Table 1) and we classified IaaS providers features into a taxonomy. The goal of our taxonomy is to help end-users understanding and choosing the more suitable and advantageous options in the wild and complex word of cloud provider offers.
- We provided an implementation of the Partial ASP Control, and Limited ASP control architectures using features and services of the Amazon Elastic Compute Cloud (EC2) Infrastructure.
  Concerning the other two architectures we proposed, the Full ASP control differ from the Partial ASP control architecture only for the implementation, on the ASP side, of the load balancer and the VM monitors (see Section 4). Therefore, to reduce the effort devoted in prototype implementation, we concentrate our attention only on the Partial ASP solution. The Extreme ASP control solutions, as described in Section 4, is thought for ASPs that have their own infrastructure and therefore is out of the scope of this paper.
- We proposed and implemented five reactive resource allocation policies: *Reactive 1 step early* (r-1), *Utilization-based, One alarm* (UT–1Al); *Utilization-based, Two alarm* (UT–2Al); *Latency-based, One alarm* (LAT–1Al); *Latency-based, Two alarm* (LAT–2Al). The r-1 policy is based on custom workload prediction model and on a queueing network system model that allows to compute the needed resources. The UT-$*$ and LAT-$*$ policies leverage the Amazon Auto Scaling service and are based on the direct evaluation of the system utilization and latency, respectively. These policies determine the allocation or deallocation of new resources on the basis of the evaluation of fixed thresholds.
- We set up a real testbed and we evaluated, under a realistic workload, the advantages and drawbacks of implementing and running the Partial ASP control versus the Limited ASP control architecture. The evaluation has been done instrumenting the system with the allocation

Table 1
A selection of IaaS provider features (updated February 2011). "Y" in the load balancing column means "Yes, but policy not specified". The SLA refers only to availability (Avail).

Provider	Customization model	Billing model	Load balancing	SLA (Avail)
Amazon Web Service [40]	Partial	1 h	LL	99.95%
AT& T Synaptic [41]	Full	1 h	Y	99.9%
CloudSigma [32]	Full	5 min	–	100%
ElasticHosts [35]	Full	1 h	–	100%
FlexiScale [34]	Full	1 h	–	100%
GoGrid [42]	Partial	1 h	RR, LL	100%
JoyentCloud [43]	Partial	1 month	Y	100%
layeredtech [44]	Full	1 month	–	100%
Locaweb [36]	Partial	1 month	–	99.9%
Opsource [45]	Full	1 h	Y	100%
Rackspace [46]	Partial	1 h	–	100%
ReliaCloud [39]	Partial	1 h	RR, LL, SI	100%
RSAWEB Cloud servers [37]	Partial	1 month	–	ND
SliceHost [47]	Partial	1 month	–	ND
Storm On demand [38]	Partial	1 h I	RR, LL, HI	100%
Terremark vCloud express [48]	Partial	1 h	LL	100%
VPSNET [33]	Partial	1 min	–	100%

policies implemented and comparing the system capability to satisfy a given Service Level Objective (SLO), the system responsiveness and the resource allocation cost.

The experience developed during the prototype implementation phase confirmed that the maturity level of currently available cloud services, even though rapidly raising, is still quite low, and that an efficient ready-to-use solution for autonomic management of cloud systems seems still far to be available. Moreover, the experimental evaluation performed validates our thesis, that is having the full control over the resource management plan has its advantages in term of achievable performances and reduced resource utilization costs.

Finally, the experimental results show that: (i) The proposed policies are capable to properly dimension the system resources making the whole system self-adaptable respect to the workload fluctuation. (ii) In case of ramp workload, the partial ASP control architecture, equipped with the r-1 policy allow to save up to 15% of resource cost, respect to the Limited ASP control solution equipped with the threshold based heuristics. (iii) In case of bursty workload and system adaptation evaluated every 1 s, the partial ASP control architecture allow to save from 17% to 32% of resources cost respect to the Limited ASP control solution. The only case where the threshold based heuristics allow to have better performances is when the system is stressed with a bursty workload and the adaptation decision is evaluated every 5 min. In this scenario the Limited ASP control solution equipped with latency based allocation policies allow to save the up to the 20% of cost respect the other solutions.

The results of this research let us to conclude that, in order to facilitate ASP (of any size and budget) to really use the cloud computing paradigm, cloud providers should offer a stronger, more sophisticated and highly customizable support to autonomic resource management.

The paper is organized as in the following. Related works are discussed in Section 2. Section 3 introduces the survey on IaaS providers and discusses the proposed taxonomy. In Section 4 are described the autonomic architectures we proposed. The implementation of the Partial and Limited ASP control architectures and the description and implementation of allocation policies are described in Section 5. Section 6 presents: the workload and the testbed setup to conduct experiments; the experimental results carried onto evaluate the system performance and responsiveness. Concluding remarks are given in Section 7.

2. Related works

In this paper we consider mechanisms for service level provisioning confining our attention on architectures for self adaptation and policies for computational resources, or more in general Virtual Machines (VMs), allocation.

2.1. Architectures for autonomic cloud-based systems

In the literature there are different proposal for architectures of autonomic cloud-based systems. In [28] is presented the idea that also cloud systems should be organized according to the autonomic loop MAPE-K [26] (Monitor, Analyze, Plan, Execute, and Knowledge). In such a way, cloud-based applications should be able to automatically react to changing components, workload, and environmental conditions minimizing operating costs and preventing SLA violations. In [3] an autonomic resource manager for hosting service platform with a two level architecture is proposed. The manager allows to determine: (1) the number of VM to allocate for each application (VM provisioning); and (2) the VM to physical machine mapping (VM packing). The authors of [9] propose an autonomic two-level resource management system with local controllers at the virtual-container level and a global controller at the resource-pool level. The authors of [4] illustrate an autonomic layered cloud architecture, Infrastructure as a Service, Platform as a Service (PaaS) and Software as a Service (SaaS). For each layer there are different goals, sensors and actuators and a different feedback adaptive loop has to be implemented. The architectural solutions we provided are inspired to the above cited work and based on previous research of one of the authors [29].

2.2. Resource provisioning policies

Concerning policies for resource provisioning at application level, research results can be classified on the basis of the system considered, the approach used to study the problem, the model formulated, and the proposed solutions. Moreover, all these works put forward solutions to empower self-adaptation. We classified the works in literature using the following five categories: (1) *Allocation policies* proposed, that could be exact solutions of an optimization problem or heuristic algorithms. (2) *Reference architecture*, that is single or multi tier services. (3) *Analysis approach* used to evaluate the proposed solution, i.e. analytical model, simulation, prototype or a mix. (4) *Type of workload* used to evaluate the proposed solution. (5) *Metrics* used to drive the adaptation and or to define the service level considered.

2.2.1. Allocation policies

In the literature there is a great variety of allocation policies proposed that can be classified in two main categories: optimal policies, e.g. [3,14,5,10,9]; and heuristic policies, e.g. [12,30,25].

In [3] the authors formulate the VM provisioning and VM to physical machine mapping problems as two Constraints Satisfaction problems with the goal to maximize a global utility function. In [14] is proposed an Integer Programming formulation of the problem of scaling up and scaling down VM instances by considering two aspects of a cloud application: performance and budget. The objective is to finish all the submitted jobs before user specified deadline minimizing the allocation cost. The proposed cloud auto-scaling mechanism compute the optimal solution of the formulated problem to automatically scale computing instances on the basis of workload information and performance desire. The authors of [5] propose a probabilistic model to optimize monetary costs, performance and reliability given user and application requirements and dynamic conditions. The optimal instance type and bid price is computed checking for all relevant combinations of the considered parameters. In [10] a dynamic provisioning technique for multi-tier Internet applications is presented. An analytical model based on queuing networks is used to determine how much resources to allocate at each tier of the applications and a combination of predictive and reactive methods is used to determine when to provision these resources. SLAs consider the percentile of the response time. A completely different optimization approach is proposed in [9]. Autonomic resource allocation is realized through the interaction of the local and global controllers using fuzzy logic to efficiently and robustly deal with the complexity of the virtualized data center and the uncertainties of the dynamically changing workloads. The global controller receives requests for physical resources from local controllers and allocates the resources among them finding the optimal allocation that maximizes a profit function.

Different heuristics for dynamic reallocation of VMs according to current resources requirements, while ensuring reliable QoS, are proposed in [30] (the authors suppose to leverage mechanisms for live migration of VMs). In [19] a multi-tiered resource scheduling scheme and an heuristic algorithm to optimize resource allocation among services in VM based data centers is presented. The heuristic has been tested with three types of interactive workload (Web service, DB service and Office Service-virtualization). Resource flowing is modeled using optimization theory and a global resource flowing algorithm that preferentially ensure performance of critical services degrading other services performance is presented. In a previous work [25] we proposed both reactive and proactive heuristic policies that leverage on information about the system performance history and that can be applied at runtime to perform SLA-aware resource management for Application Service Providers in the Cloud. Moreover, many works (e.g. [2,17,23]) address the resource provisioning problem using control theory and feedback controllers. The solution determined are sub-optimal. The author of [23] propose an adaptive resource control system that integrates the Kalman filter into feedback controllers to dynamically adjust the resource shares to individual tiers in order to meet application-level QoS goals while achieving high resource utilization. The authors of [17] propose an adaptive resource control system that dynamically adjusts the resource shares to individual tiers in order to meet application-level QoS goals while achieving high resource utilization. The controller algorithms were designed based on input–output models inferred from empirical data using a black-box approach. In [2] is presented an external controller (feedback control) that users can utilize to allocate or deallocate cloud resources for dynamic applications. APIs to enable more effective control are proposed. Proportional thresholding is the control policy utilized and the considered parameters are CPU utilization and the workload intensity.

In this paper we propose four heuristics policies that, unfortunately, cannot be compared with other solution because differences in the considered system model.

2.2.2. Reference architecture

In literature could be found work assessing single tier systems and papers studying multi tier applications. In the case of multi-tier systems, VM provisioning can be done at a single layer or for every architectural layer. In [3,4,9,10,17] are proposed solutions for multi-tier provisioning. For example: in [3] the authors proposed an autonomic resource manager for hosting service platform with a two level architecture; in [10] the authors presented a dynamic multi-tier provisioning for Internet applications; in [9] is proposed a two-level resource management system with local controllers at the virtual-container level and a global controller at the resource-pool level.

In all the other previously cited works, as in this paper, is consider a single-tier architecture. However, in our case, the extension of the proposed allocation policies to a multi-tier scenario is straightforward.

2.2.3. Analysis approach

The widely used analysis approach in this field is the evaluation, by prototype, of the proposed solutions (algorithms or mechanisms).

Prototypes are realized using public clouds like Amazon EC2 [5] and Windows Azure [14] or private clouds (mostly based on Xen hypervisor [10,12,19,23]). For example in [12] a prototype has been Implemented on the top of the Xen hypervisor to perform predictive resource scaling. Also the system presented in [10] has been Implemented using real traces on 40 Xen/Linux machines.

Less realistic but still used analysis approaches are simulation (e.g. [30,3,13]) or analytical models [6]. In [30] the Cloudsim simulator [31] has been used to evaluate heuristic policies to obtain energy efficient allocation of VMs in cloud data centers. In [3] simulation is used to test the optimal allocation of VM. In [13] the authors used simulation to evaluate a middleware architecture that enables SLA-driven dynamic configuration, management and optimization of cloud resources.

In [6] is proposed an analytical performance model, with two interactive job classes, with the aim to determine the minimum number of resources (servers) needed to satisfy SLAs. The proposed scheduling disciplines are evaluated analytically.

In this work we evaluated the proposed architectures through the performance analysis of a system prototype built on top of Amazon EC2.

2.2.4. Type of workload

The workload used to stress a resource management policy depends on the specific system and case study. However, we have can classified all the considered workload in two main categories: interactive or batch. Some works (e.g. [5,14]) consider batch applications, others [7,9,12,17,23] interactive applications (e.g. e-commerce applications, auction sites, wikis).

Among works that consider interactive workloads, in [7] is proposed Oceano, a prototype of a highly available, scalable, and manageable infrastructure for e-business computing utilities. In [9] the proposed resource management system is implemented on a virtualized data center testbed and evaluated using applications that are representative of e-business and Web-content delivery scenarios. Both synthetic and real-world Web workloads are used to evaluate the effectiveness of the approach. To test the prototypes some works ([12,17,23]) use RuBIS, an auction site prototype modeled after eBay.com that is used to evaluate application design patterns and application servers performance scalability. In [17] the authors consider also the TPC-W benchmark to evaluate the proposed solution.

Our work defines and evaluates allocation policies for interactive web applications (i.e. Mediawiki tested with Wikibench generated workload) since we consider more challenging to perform adaptation in a field characterized by an higher degree of variability (i.e. bursty workload) and by stricter QoS requisites (i.e. low response times).

2.2.5. Metrics

In literature various metrics have been proposed as indicators to describe service level and therefore to perform adaptation. SLA indicators usually translate in constraints in optimization problems that, as described before, can be solved using exact or heuristic approaches. Many works consider CPU utilization (e.g. [2,3,12]), other solutions are based on the response time (e.g. [6,10,14]). Cost and energy consumption are considered in [30]. Few work consider other metrics, for example [5] take account of security and reliability.

In this work, to determine the resource needed to satisfy SLAs, we consider request arrival rate and CPU utilization as system model parameters, and the latency (i.e. response time) as constraint.

3. IaaS providers taxonomy

Today, the market of cloud providers is a wild. Several companies enter the business of cloud computing organizing their portfolio around infrastructure, platform and software/application services. While an end-user (e.g. an ASP) has a rich choice, it is difficult to compare alternatives.

This work tries to shed light in the world of public IaaS providers analyzing seventeen of them (listed on the first column of Table 1). Our analysis is oriented to identify the interesting characterization elements in an autonomic resource management perspective, that is we identify the feature useful (and needed) to realize self-adaptable solution. In the specific we consider the type of SLAs, the billing model and the resource management features proposed to the end-user. From our study we identified seven main attributes (see Fig. 1) to characterize an IaaS provider: VMs Customization model, Resource Usage Billing model, SLA model, Type of Interface, Load Balancing services, Monitoring tools and Auto Scaling services.

The **Customization model** is intended as the degree of VMs customizability offered by IaaS providers. We identified two types of customization models: Full and Partial. In the first model the end user can specify and, in many cases, modify at runtime, the amount of processing power, memory and storage of every VM. In the Partial customization model the IaaS provider offers a limited set of predefined VM types characterized by different capacity and characteristics. In this case a VM instance cannot be modified once running. For what concern system images, while some providers allow users to load (and save) their own system images, others allow only to choose from a set of predefined images (e.g. Linux, Windows or Solaris OS images).

The **Billing Model** is related to the policy adopted to bill the usage of VMs, storage and network resources. For what concern the VMs billing model, different temporal scales (per minute, per hour or per month basis) are adopted by different providers. The majority of the considered IaaS providers applies a 1 h based billing. Only two providers (CloudSigma [32] and VPSNET [33]) have a shorter billing period (5 min and 1 min respectively) while others have a per-month based billing. It is important to remark here that we are considering only VM usage billing and not the one for extra services such as load balancing, network usage, storage, and auto scaling. Furthermore it is worth to mention that, for the sake of automated VM provisioning, monthly based billing is not acceptable since it does not allow to react quickly to sudden traffic changes. Concerning the Storage and Network resources usage model the billing is per bytes used and transferred respectively.

Service Level Agreement model. Almost all the considered providers guarantee a minimum Service Level to users. In case of SLA violations users are partially or totally refunded by their IaaS provider. All the considered providers have SLAs based only on availability requirements (from 99.9% to 100% on a monthly or yearly basis). In many cases availability (or uptime) is referred to the cloud infrastructure (power and network) and single machines uptime is not considered, however a maximum time period for machine recovery is often defined. Some providers (e.g. FlexiScale [34]) guarantee physical servers uptime while others (e.g. CloudSigma [32], Elastic Hosts [35]) guarantee the uptime of virtual servers. Most of the considered providers also define a set of conditions in which SLAs may be violated without penalties (e.g. during scheduled maintenance). No provider currently considers SLAs based on performance indicators such as response time.

The **Interface Type** is intended how the user can interact with the cloud infrastructure. Broadly, IaaS providers offer access through remote terminal (i.e. SSH), web based GUI and/or APIs to interact with, configure and program VM instances and other resources. The availability of APIs is mandatory to implement ASP resource management. Almost all of the considered providers offer, in addition to SSH access, both a web based GUI and APIs to interact with VM instances and other resources. The only exception are Locaweb [36] that offers only SSH access and RSAWEB Cloud Servers [37] that offers SSH and GUI, but not APIs.

Load Balancing is the capability to distribute the incoming load among various VMs. The load balancing feature is characterized both by the balancing policy used and by the capability to support sticky sessions. Moreover, load balancing can be performed at hardware or software level. The offered load balancing policies are state-blind (e.g. round robin (RR)) or state-aware (e.g. least loaded (LL)). A particular state-aware balancing policy called Historical Intelligence (HI) that predicts the appropriate node to use by historical data analysis is offered by Storm [38] while ReliaCloud [39] offers a state-blind policy (SI) that performs balancing applying an hash function on source IP addresses. In most of the cases the balancing policy is fixed by the provider and cannot be modified by users. Stickiness

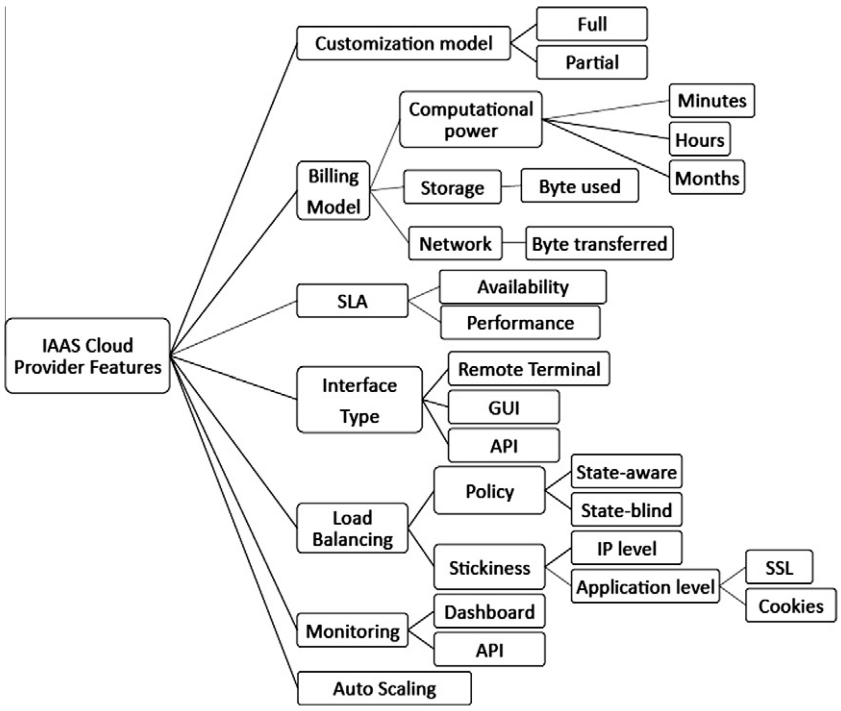


Fig. 1. The taxonomy of features offered by the considered IAAS cloud providers.

can be supported either at application or IP level. At the best of our knowledge, only Amazon offers a cookie based (application level) stickiness.

The **Monitoring Services** refer to mechanisms, offered by the IaaS provider, allowing users to monitor and collect system performance metrics (e.g. VM CPU and memory usage, load balancing statistics, etc.) at a fine level of granularity (e.g. 1–5 min). Many providers offers a dashboard showing usage statistics and performance indicators or have an alarm system that warns users by email in case of failures. However the dashboard, usually accessible through a web portal, is useless to implement an autonomic system. For our purpose, advanced built-in monitoring services accessible using APIs are needed. If these services are missing, users have to implement their own monitoring agents to collect data and a monitoring manager component that aggregates and integrates collected data.

Auto Scaling Services refers to the capacity of the IaaS provider to automatically add/remove VMs on the basis of observed performance metrics (e.g. CPU utilization, number of requests, average response time, etc.).

Table 1 shows a selection of the features implemented by the IaaS cloud providers we considered. Features not shown are that offered by almost all the providers. Advanced monitoring and Auto Scaling services are omitted since, at the best of our knowledge, only Amazon Web Service is currently offering this kind of services.

4. Autonomic QoS-aware resource provisioning architectures

As previously introduced, we consider an Application Service Provider that offers a Web service or a Web application. The ASP can lease computational and storage capacity from a public IaaS provider. In this way the ASP can scale its performance in case of unpredictable workload fluctuations and can avoid start-up costs. We suppose the application can be totally or partially replicated on a variable number of *VMs*.

As mentioned in Section 2 the effort of the community goes in the direction of providing solutions for the autonomic management of cloud resources that is, the system adaptation is performed following the MAPE-K cycle [26] (Monitor, Analyze, Plan, Execute all on the basis of a shared knowledge). A MAPE-K based system is capable, to select, allocate or deallocate resources when are detected changes in the workload, system state and SLA. In this paper we concentrate our attention on results related to the analysis and planning phase, however different research work proposed solutions framed at all the stages of the MAPE-K cycle and applied to all the levels of the cloud stack [28,3,9,4].

As shown in Fig. 2, the components of an autonomic QoS-aware service provisioning architecture are:

Performance and workload monitoring components. These modules are the sensors of the QoS-aware service provisioning system and are in charge of collecting information on the performance state of the computational, storage and network resources used, as well as on the workload submitted by users (e.g. requests rate, type, size). Through *Monitors* relevant performance indexes (e.g., response time, network traffic, CPU utilization, etc.) and workload information are collected, aggregated and elaborated.

The SLA Analyzer component. This module is in charge of elaborating and analyzing data collected by monitors. Moreover, the analyzer should trigger the Planner component if

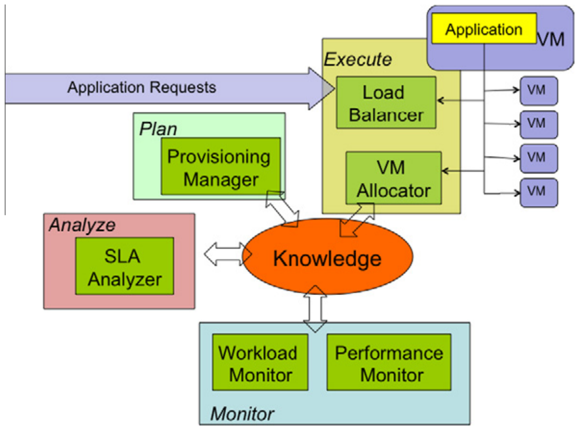


Fig. 2. The autonomic QoS-aware service provisioning architecture.

significant changes in the data pattern, e.g. SLA violations and abnormal workload fluctuations, are detected. Data obtained by the monitors (average response time, observed and expected arrival rate, etc.) are examined by this component to check whether adaptive actions are needed to guarantee the requested level of QoS.

The Planner (or Provisioning Manager) component. This module, instrumented with a system model, evaluates the system adaptation actions (e.g. configuration changes) that must be operated to guarantee SLAs and optimize costs. Reconfiguration is triggered by the Analyzer and the system model is usually parameterized with information collected by the performance and workload monitors and elaborated by the Analyzer. If an adaptation is needed, the *Planner* decides, on the bases of the system performance model and of a resource allocation algorithm, the corrective actions to take (e.g. how many VMs to allocate or deallocate).

The Resource Manager (or VMs Allocator) component, that is in charge to actuate the strategies determined by the planner to properly dimension and manage resources, e.g. allocate and deallocate VMs, resize the capacity of running VMs, increase or decrease the storage.

The Load Balancer component, that distributes requests among the instantiated resources, e.g. application servers, web server, and databases, running on leased virtual machines. This module should be capable of guaranteeing, at least, session persistency.

With the proposed taxonomy we show that an ASP has various options to manage cloud resources and services, ranging from: to completely use an IaaS integrated solution (with a limited degree of customization), to build its own autonomic service provisioning system using some features and services offered by an IaaS provider, or to implement all the required features and services building a private cloud.

In the following we propose and discuss four different design solutions, namely: *Extreme ASP control*, *Full ASP control*, *Partial ASP Control*, and *Limited ASP control*.

4.1. Extreme ASP control

As shown in Fig. 3, in this architecture all the cloud layers are under control of the Application Service Provider.

Performance measures are collected by *Monitoring Agents* placed on the VMs and on the Load Balancer.

The *Performance Monitor* and *Workload Monitor* periodically collect performance indexes related to the application, the virtualized resources usage and the workload intensity. Moreover, depending on its sophistication, the monitoring components may aggregate, estimate and forecast the system performance level and resources demand.

The *SLA Analyzer* determines whether a new allocation decision has to be taken (for example if a violation is observed or predicted or at fixed time intervals) and, if necessary, triggers the Provisioning Manager.

The *Provisioning Manager* is the core of the system. As before mentioned it is equipped with a system model that is solved to determine the adaptive action to take in order to maintain the system performance level also if the system state and workload condition change. The system model is parameterized with observed and estimated system state information from the performance and workload monitors. The system model is typically used to derive an optimization problem. The solution of the optimization model produce a resource management plan, that determines the adaptation actions to take. The adaptation actions are typically determined solving an optimization problem (exact solution) or through heuristic algorithms (sub-optimal solution).

The resource management plan is actuated by the *VM Allocator* service, responsible for the allocation and deallocation of VMs.

The *Load Balancer* distributes incoming requests among active VMs and is capable to manage sticky user sessions. The load balancer may be either one-way or two-way.

Finally, the *Knowledge* module is intended as the set of mechanisms used to share information among all the other modules. Typically there are sensors that acquire data (e.g. the performance and workload monitors), software module that clean/post-process data and a repository (e.g. a database) where data are stored. Moreover there is a protocol used to exchange information (e.g. each module can extract data directly from the database, or a system module can provided data as input parameters of the API used to invoke the functionalities offered by another module. A

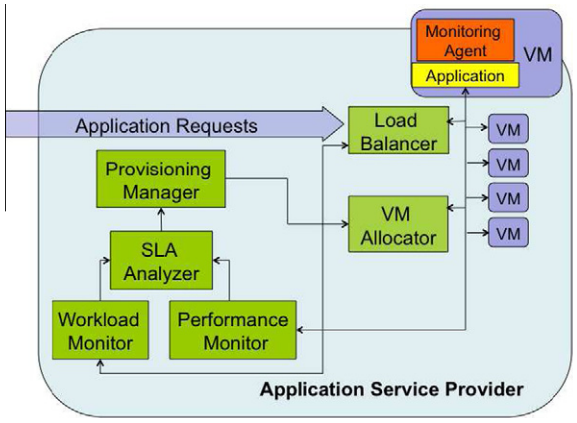


Fig. 3. Extreme ASP control architecture.

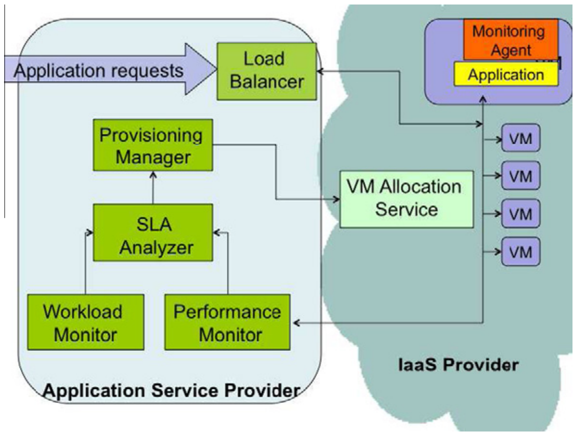


Fig. 4. Full ASP control architecture.

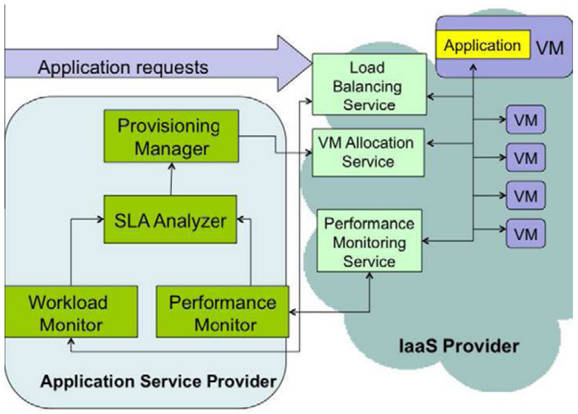


Fig. 5. Partial ASP Control architecture.

more detailed description of the Knowledge module is provided in [29].

The Extreme ASP control is a solution suitable for in-house data centers (or private clouds), where no resources are leased from IaaS providers: the ASP has total control over all the components at the high cost to be forced to build its own data center.

4.2. Full ASP control

In this architectural configuration (see Fig. 4), the ASP has total control over all MAPE-K cycle phases. VMs are managed by the ASP using IaaS provider VM Allocation Service APIs. The VMs and applications performance are directly monitored by the ASP installing monitoring agents on the VMs. A possible variant of this architecture could be realized if a *Performance Monitoring Service* is provided. In this case the IaaS manages its own monitoring agents that collect aggregated VMs and load balancer performance indexes and provide them to the Performance and Workload Monitors.

This architecture has several benefits. First, the ASP has total control on the Analyze and Plan phases of the autonomic cycle. Second, the ASP can avoid the high costs for starting-up and running a data center, and finally the ASP can take advantage of the high scalability and availability guaranteed by a public cloud infrastructure. The main drawback is that the ASP is required to set up a robust and scalable network infrastructure and a Load Balancer to manage the incoming flow of requests (and outgoing traffic). To overcome this issue, of course reducing the degree of control, load balancing features offered by many IaaS (8 out of 17) could be used.

4.3. Partial ASP control

This architecture overcome some drawbacks of the Full ASP control above mentioned, but leave the ASP in control of the Analyze and Plan phases (see Fig. 5).

The performance and workload monitors elaborate data collected by the IaaS Performance Monitoring Service and pass them to the SLA Analyzer. When the SLA Analyzer detects the need to change the number of allocated resources, it invokes the Provisioning Manager that determines the adaptive actions to perform. Therefore, the Provisioning Manager uses the VM Allocation service APIs to actuate the adaptation actions.

In our vision, this solution is a good tradeoff between the need of the ASP to have total control over the components that perform the adaptation (i.e. Performance Monitor, Workload Monitor, SLA Analyzer and Provisioning Manager) and the opportunity to exploit the scalable and robust load balancing and monitoring functionalities offered by IaaS providers. For example to setup a custom and robust load balancer an ASP must acquire, at least, gigabit bandwidth connection and a dedicated load balancing device supporting gigabit throughput, plus costs of personnel. On the contrary, using a Cloud load balancing service is cheaper and reduce infrastructure and personnel costs.

Concerning the Performance Monitoring service, if it does not provide enough information for the adaptation algorithm, additional monitoring agents could be installed by the ASP.

4.4. Limited ASP control

This architecture (see Fig. 6) can be implemented only with IaaS providers that offer auto scaling functionalities. In this case the IaaS provider controls all the MAPE-K phases: we assume that the SLA Analyzer and the Provisioning Manager are part of the Auto Scaling Service offered by the IaaS provider. The ASP is only responsible to set the appropriate scaling rules on the IaaS provider auto scaling component. This architecture variant is very easy and fast to implement but suffers the drawbacks imposed by the limited customizability of autoscaling services currently offered by IaaS providers (see Section 3).

5. Implementation

In this section we describe how we implemented the Partial and Limited ASP control architectures using Amazon Elastic Compute Cloud services. In particular, the Limited

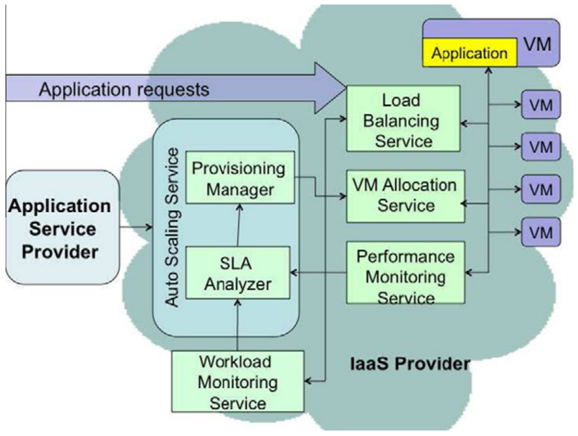


Fig. 6. Limited ASP architecture.

ASP control implementation, that we take as reference point in the remaining of the paper, uses the Amazon Auto Scaling service.[1] Moreover, we describe also the allocation policies used and finally we discuss implementation issues and lessons learned.

5.1. Partial ASP control

Single VMs are EC2 on-demand instances of the same size (e.g. EC2 m1.small instances) placed behind an Elastic Load Balancer in one or more availability zones inside the same EC2 Region (at the moment, Amazon Load Balancers cannot span multiple EC2 Regions). The target web application is completely replicated on every VM and data are centralized. This simplifying assumption allows us to consider the application as a whole without taking account of interactions between its layers (otherwise resource replication and distribution have to be considered for every layer) and data synchronization issues.

Incoming requests are managed by the Elastic Load Balancer that distributes them among active VMs using the Least Loaded policy. The Performance Monitoring Service is realized through Amazon CloudWatch. CloudWatch is a web service that allows to collect, analyze, and view system and application metrics. Moreover, CloudWatch allows to aggregate both VM and LoadBalancer metrics for a period that can range from one minute to two weeks.

The Performance Monitor collects, using CloudWatch API `GetMetricStatistics`, the *Average Response Time* seen by the Load Balancer (i.e. average value of the `Latency` metric) every minute.

In the same way the Workload Monitor gets from CloudWatch the *Total number of incoming requests* processed by the LoadBalancer (i.e. the sum of the `RequestCount` metric). Moreover, the Workload Monitor uses the collected information to forecast the arrival rate [11] in the next hour. The forecast, depending on the prediction method used, may be performed directly by a custom software module in the Performance Monitor or by an dedicated external component. We implemented the second solution using Matlab.

The SLA Analyzer collects observed and predicted metric values from the monitors and triggers the Provisioning Manager at fixed time intervals (e.g. every 5 min).

The Provisioning Manager is instrumented with the performance model of the system, that is a M/G/1 queue network (as in our previous work [11]). When the provisioning manager is triggered by the SLA Analyzer it solves the model and determine the number of VMs that have to be allocated in the short-medium term (from 1 to 5 min to one hour) to guarantee SLAs.

The various components of the Autonomic Architecture we implemented (Performance and Workload Monitor, SLA Analyzer and Provisioning Manager) have been realized as Java modules running under JRE 1.6 and using AWS SDK for Java 1.1.8 APIs to interact with EC2 services. We do not test the scalability of the autonomic architecture itself. Of course, this issue is important but it does not represent an immediate limitation. While the SLA analyzer and the provisioning manager work on a time scale of minutes, the performance monitor and workload monitor work on the scale of seconds. In the workload scenarios consider these modules are not the system bottleneck, however, an optimized and scalable implementation should be investigated.

Once the Provisioning manager determines the allocation/deallocation actions that should be taken, it invokes the VM Allocation Service to perform them using `LaunchtInstances` and `TerminateInstances` APIs.

Once allocated/deallocated, the VM instances are added to/removed from the Load Balancer using `RegisterInstancesWithLoadBalancer` and `DeregisterInstancesFromLoadBalancer` APIs.

5.2. VMs allocation policies

In the Partial ASP control implementation we use a simple, but effective, reactive allocation policy, named *Reactive 1 step early* (r-1), that works as follows. r-1 observes the requests arrivals and computes the average arrival rate over time slots of 1–5 min. Measured the average arrival rate λ_{i-1} for the time slot $i-1$, the arrival rate for the time slot i is estimated as $\widehat{\lambda_i} = (1+a)\lambda_{i-1}$, where $a > 0$.

Computed the estimated arrival rate and supposing that the service rate μ and the maximum response time allowed R_{max} are known, the minimum number of VMs $x_{i,\min}$ needed to guarantee R_{max} is determined by

$$x_{i,\min} = \frac{\hat{\lambda}_i R_{max}}{\mu R_{max} - 1} \tag{1}$$

The deallocation policy is the following: A VMs is deallocated only if no more needed, that is if at the beginning of time slot i the number of allocated VMs is greater then $x_{i,\min}$, and if the billing period (typically 1 h) is expired (or is going to expire in few minutes). The assumption behind this deallocation policy is that does not make sense to deallocate a resources for which we already paid.

As previously mentioned the limited ASP control solution is directly implemented using the Amazon Auto Scal-

[1] http://aws.amazon.com/autoscaling/.

ing service. Amazon Auto Scaling offers the possibility to define allocation/deallocation strategies based on CloudWatch metrics values. In particular, Amazon provide mechanisms to set-up alarms on every CloudWatch metric and to decide what action to take when the threshold specified by the alarm is violated. We use these mechanisms defining the threshold values, determined on the basis of a tuning campaign, and what adaptation action to take (i.e. how many VMs have to be allocated/deallocated) when a threshold violation is detected.

In the specific we set up the following four policies:

- *Utilization-based, One alarm* (UT–1Al). This policy scales up and down VMs using a threshold on the value of the average CPU utilization of all the VMs registered within the load balancer. In the specific if the utilization exceeds 62% an instance is allocated while, if it falls under 50% an instance is deallocated.
- *Utilization-based, Two alarms* (UT–2Al). This policy uses a double threshold on average CPU utilization to scale up/down. If the utilization is greater than 62% a single instance is launched while, if exceeds the second threshold of 70% two instances are launched. In the same one VM is deallocated if the utilization went below the 50% and two VM are deallocated if the utilization value is less than 25%.
- *Latency-based, One alarm* (LAT–1Al). This policy is similar to the UT–1Al described above except for the fact that, for scaling up, a threshold on the average latency is set. In particular, a new EC2 instance is launched if the average latency is greater than 0.2 s. Since the latency value is not appropriate to determine a deallocation threshold (it never falls under a minimum value of about 0.08 s), to scale down the threshold on the minimum utilization of 50% is still used.
- *Latency-based, Two alarms* (LAT–2Al). This policy is similar to the UT–2Al but uses two thresholds on the maximum latency value to scale up. A VM is allocated if the average latency is greater than 0.2 s, two if it is greater than 0.5 s. The deallocation policy is the same as in UT–2Al.

In all the policies we defined, periodically is evaluated the possibility to take an adaptation action. The evaluation periods we consider are equal to 1 and 5 min. For the r-1 policy timing is directly managed by the SLA analyzer we implemented. For the threshold based policies we used a cool down interval (i.e. the minimum time interval between two adaptation actions) equals to the evaluation period.

6. Experimental evaluation

As stated in the introduction our goal is to evaluate the advantages and drawbacks of implementing and running the Partial ASP control architecture (equipped with the *r-1* allocation algorithm) versus the Limited ASP control architecture (equipped with the *UT-** and *LAT-** heuristics. The evaluation has been done instrumenting the systems with the allocation policies described in Section 5.2 and comparing the system capability to satisfy a given Service Level Objective, the system responsiveness, and the allocation costs of the implemented policies. As responsiveness we define the capability of a system to promptly react to unexpected load variations. We do not define any specific metric for the responsiveness but it is estimated through the analysis of the response time and allocated VMs time series.

The metrics used for comparison are the following:

- Latency (or Response time) collected by the CloudWatch monitor at the load balancer. The response time is collected (and averaged) every minute. We used both the time series and the empirical CDF to compare performance and to give indications on system responsiveness.
- Number of Allocated VMs, evaluated by CloudWatch. We consider both the time series and the total value. We remarks that the number of allocated VMs is a cumulative value that does not take into account the identity of the allocated instances.
- VMs Allocation Cost, evaluated analyzing the log of VMs allocation and deallocation actions. In the specific, we compute the total and hourly cost. We considered only variable costs, determined by the VMs we run. The cost for all the other resources and services (CloudWatch, Elastic Load Balancer, back-end server and Database) has not be considered since it is the same for both the considered implementations and it is constant over all the experiments.

Both system implementations are evaluated under the same load conditions (see Section 6.1), that are characterized by:

- a smooth variation of the workload intensity, and
- burst of requests of unpredictable and heavy intensity (unpredictability has been modeled alternating burst of requests with periods of low intensity).

The fairness of our comparison is given by the fact that both architecture are implemented using the Amazon EC2 infrastructure and that we made an high number of run to masquerade the unpredictability and unrepeatability of a real environment.

From the experimental results emerged that (see Section 6.3):

- Having full control over the resource management plan have its advantages in term of achievable performances and reduced resource utilization costs.
- The proposed allocation policy, r-1, outperforms the threshold based policies implemented using the Auto Scaling service.
- Latency and responsiveness benefit of short performance and workload evaluation periods if the allocation/deallocation decision is taken periodically rather than triggered by events.

6.1. Workload generation

Right now, benchmarking cloud services is an open issue and no standard solutions, neither largely accepted solutions are available. One of the main reason is that the benchmark depends on the type of target service (e.g. IaaS, PaaS, SaaS, DaaS) and the goal of the evaluation. For example, CloudHarmony [49] provides benchmarking services to test performance (CPU, I/O, Applications), network, uptime. Of our interest is a benchmark capable to reproduce the behavior of a typical Web 2.0 application and to test the elasticity, responsiveness and performance of an autonomic cloud manager. At the best of our knowledge, CloudStone [50] as been the first Web 2.0 benchmark for cloud systems. CloudStone is a toolkit consisting of an open-source Web 2.0 social calendar application (Olio – http://incubator.apache.org/olio/) and a set of automation tools, based on Faban (http://java.net/projects/faban/), for generating load and measuring its performance in different deployment environments. Today, the benchmark is not more maintained and we experienced that the current available version suffers from various implementation issues. Another useful benchmark for distributed large scale applications is WikiBench [27], a Web hosting benchmark allowing to stress-test systems designed to host Web applications, from single components (e.g. application servers, database, load balancers) to Cloud computing platforms. WikiBench uses MediaWiki (the application used to host wikipedia.org) as stress test target application, manipulates real data (actual database dumps of the wikipedia website), and generates real traffic by replaying real Wikipedia traces. However, WikiBench is not completely useful for the purpose of our performance analysis because the Wikipedia traces [51] do not represent a very bursty workload. Wikipedia workload intensity varies in a very periodical way and, usually, increases not more than of 100% in a 12 h period. For this reason we think that WikiBench it is not appropriate to evaluate elasticity, stability and responsiveness of a cloud application subject to a very bursty traffic.

Therefore, we decided to create our own workload generator assembling: MediaWiki as the Web 2.0 target application, scaled wikipedia traces to generate the requests [51], and httperf [52,53] to control the request generation rate and statistics collection. In this way we have been able to generate not only traffic with a smooth and periodical variation of intensity (*smooth ramp workload*) but also controlled bursts of requests ranging from the 33% to the 300% of the base traffic (*bursty workload*), as shown in Fig. 7.

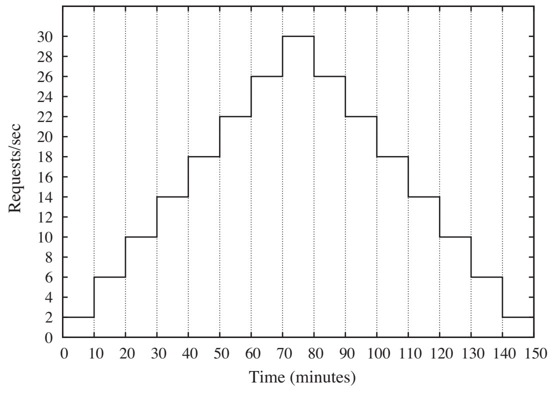
6.2. Testbed setup

As previously introduced we implemented our testbed by means of the Amazon EC2 infrastructure (see Fig. 8). In the specific we used:

- From 1 to 10 Amazon EC2 m1.small instances (Linux 32 bit machines having one virtual core with1 EC2 Compute Unit and 1.7 GB of memory). Each VM runs, as application server, Apache 2.2.16 with PHP module installed (PHP version 5.3.6). Each VM replicates and executes the front-end part of the MediaWiki web application (version 1.16.4). The MediaWiki back-end database is centralized and runs on a dedicated node.
- One Amazon EC2 m1.large instance (a Linux 64 bit machine with 2 virtual cores with 2 EC2 Compute Units each, 7.5 GB of memory and MySQL 5.1.52 DBMS) implementing the database server that runs the MediaWiki back-end. This solution avoid multi-tier load balancing and data consistency problems.
- The Amazon Elastic Load Balancer that distributes the traffic among the VMs.
- An EC2 m1.small instance, located in the same availability zone of the application and back-end servers, to run the workload generator.
- An EC2 m1.small instance to run the components of the Partial ASP control architecture we implemented (Performance and Workload Monitor, SLA Analyzer and Provisioning Manager).

The VMs, the database, the load balancer and the workload generators run all in the same availability zone (us-east-1a). Placing the workload generators and the servers in the same availability zone we tried to isolate the components of the total response time due to queuing and computation reducing at the minimum the effects of network latency.

Once installed the MediaWiki servers, the DB has been populated with the Wikipedia database dump used in



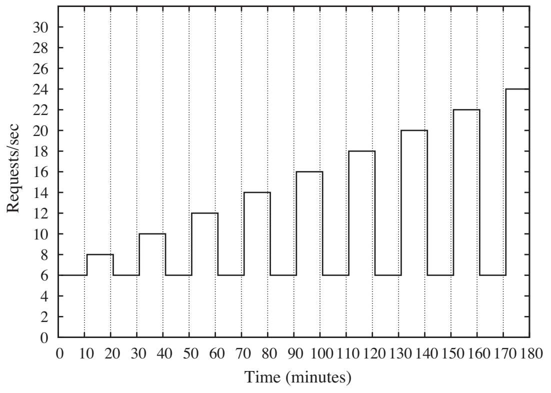


Fig. 7. Two example of stress-test workload generated using MediaWiki and Httperf. On the left the smooth ramp workload and on the right the bursty workload.

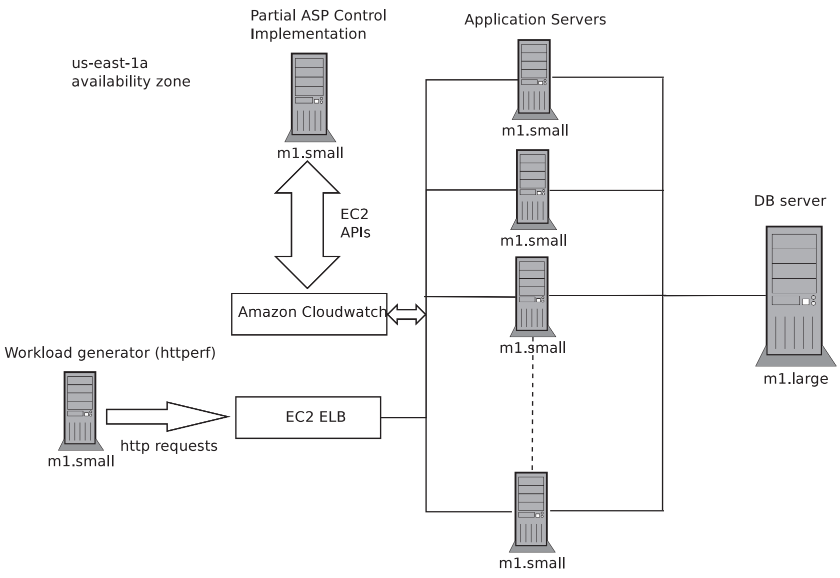


Fig. 8. Architecture of the implemented testbed.

Wikibench (snapshot of January, 3 2008) that contains about 7 millions of wiki pages.

As previously introduced, we generated the workload starting from real traces. We used a sample (10%) of the real Wikipedia access traces during the period from September 17th to October 1st, 2007. The trace contains only the requests for the english version of Wikipedia web pages. (The access log has been pruned using the `Tracebench` tool provided with WikiBench). Finally, we generated the desired load intensity through Httperf `family wlog`, a workload generator that iterates requests from a list of URLs addresses from a real trace at the rate specified by the user.

To evaluate the capacity of a single VM and the workload intensity it is capable to handle in normal load conditions we proceeded as follows.

First, empirically, we evaluated that an EC2 m1.small serves four requests per second with an utilization of about 62%. These results has been obtained averaging the results of multiple runs of the same workload at different hours of the day and day of the week. This evaluation has been necessary because we observed that the performance isolation for the VMs is not complete and that the service rate of the single VM is influenced by the utilization level of the physical host it runs on and by live VM migrations (factors that are not under the control of the Amazon EC2 infrastructure user).

Second, considering a simple M/G/1 model and that the system is stable ($\rho = \lambda/\mu < 1$) we evaluated the service rate of a single VM as 6.45 requests per second.

Finally, supposing the ASP has to guarantee a maximum response time (the SLO) of 0.5 s, we were capable to determine the minimum number of VMs needed to handle a given load and to satisfy the SLO. The minimum number of VMs is evaluated using Eq. (1) (we consider an M/M/m queuing model).

Moreover, empirically we evaluated that the EC2 m1.large instance used to host the MediaWiki database is capable to guarantees that, for the level of traffic we submit to it (up to 40 req/s), the back-end server never represents the system bottleneck.

6.3. Performance and responsiveness analysis

The first set of experiments shows the behavior of the r-1 allocation policy, the UT–1Al policy and the LAT–1Al policy for the smooth ramp workload (Fig. 7) when the allocation/deallocation action is taken every minute. We chose these three policies as representative of the whole set we consider, because the double threshold based policies have essentially the same behavior of the single threshold in case of smooth ramp workload.

Fig. 9 shows the allocated VMs and the response time. Comparing the trend of the workload (request per seconds) and the trend of allocated VMs, is clear that both Partial ASP control and Limited ASP control are capable to adapt the system capacity according to the workload fluctuations. In the rump up phase of the workload (0–70 min), the latency measured for the LAT–1Al algorithm is higher than the latency measured using the r-1, therefore the LAT–1Al policy pay the use of less VMs. The strange behavior at the beginning of the experiment (5–20 min) is due to an apparent incapability to manage the incoming workload when only two VM are allocated. Indeed, after the new VM are allocated, the latency time goes down.

Moreover, in this scenario all the policies are capable to maintain the response time below 0.5 s, the specified SLO. The r-1 VM allocation profile is different from the others because VMs are deallocated, if needed, only when the 1 h billing period is expiring. This allows to have a more stable response time in the ramp up phase of the workload (from 0 to 70 min). The only exception, common also to the other policies, is the spike of the response time at time 20, when 2 VMs are not enough to manage 10 requests per second. The allocation policy reacts after a couple of

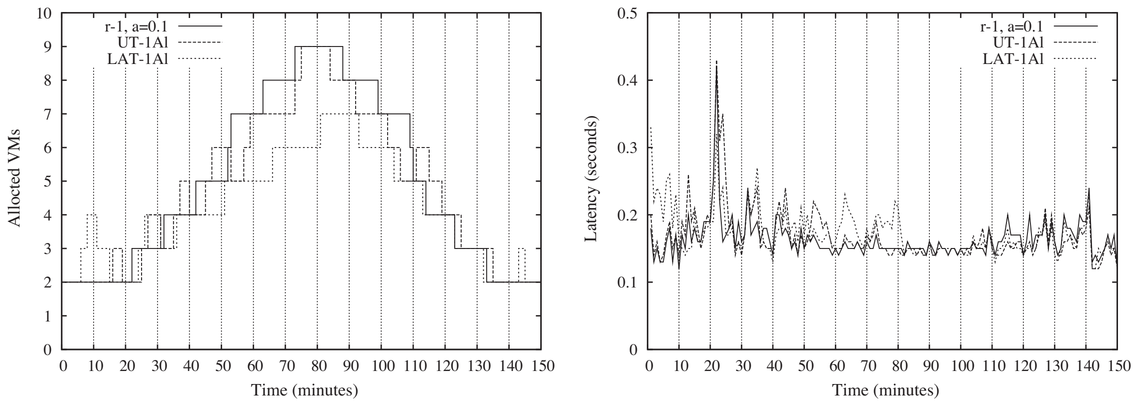


Fig. 9. The allocated VMs time series (left) and the Latency time series (right) for the smooth ramp workload.

Table 2
The allocation cost for the smooth ramp workload.

	Total hours	Total cost	Cost/hour
r-1	12	1.02	0.408
UT–1Al	17	1.445	0.578
LAT–1Al	16	1.36	0.544

minutes and therefore the system is not capable to offer the proper service rate. After a new VMs is allocated the response time jump back to lower values.

Therefore, in the worst case, the responsiveness of the allocation policies can be quantified in 2 min given by the time to recognize a change in the workload intensity or system state, and the time to actuate the allocation of new VMs.

Table 2 compares the allocation costs. The total hours column shows the total number of hours billed to the ASP. r-1, obviously, outperforms the threshold based policies. The difference of the total cost over a three our experiment, with a peak usage of 9 VMs, is negligible but, saving about the 15% of the cost is, of course, significant over a longer utilization period and for an higher number of VMs.

In the second set of experiments we compared all the allocation policies under the bursty workload (see Fig. 7). We considered two cases, namely *1-min* and *5-min*. In the 1-min case the allocation/deallocation decision is taken every minute and in the 5-min case every 5 min.

Results for the first case are shown in Figs. 10 and 11. Comparing Figs. 7 and 10 (allocated VMs) is evident that also in this case the system (both architectures) is capable to self-adapt its configuration to contrast the workload fluctuation. However the higher intensity of the workload impact the system responsiveness. that range from 2 to 10 min when the burst exceeds the 130% of base workload (i.e. starting from time 70 min). The empirical CDF plot shows that r-1 is capable to satisfy the SLO in more then the 95% of observation. The UT–1Al policy follows with a SLO satisfaction in more than the 87% of the cases. All the other policies offer worst performances (from 79% to 82%).

The allocation cost is reported in Table 3. The r-1 allocation strategy allows to save about the 32% versus the UT–2Al, LAT–2Al and LAT–1Al policies, and about the 17% compared to the UT–1Al policy.

When allocation/deallocation decisions are taken every 5 min the system is more stable (i.e. VMs are not hysterically added and removed), as shown by the time series of

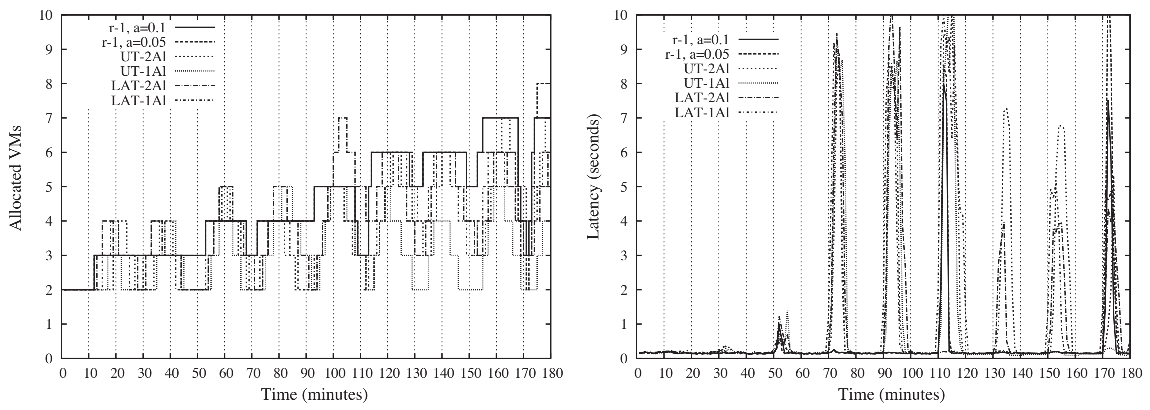


Fig. 10. The allocated VMs time series (left) and the Latency time series (right) for the bursty workload, and for an evaluation time period of 1 min.

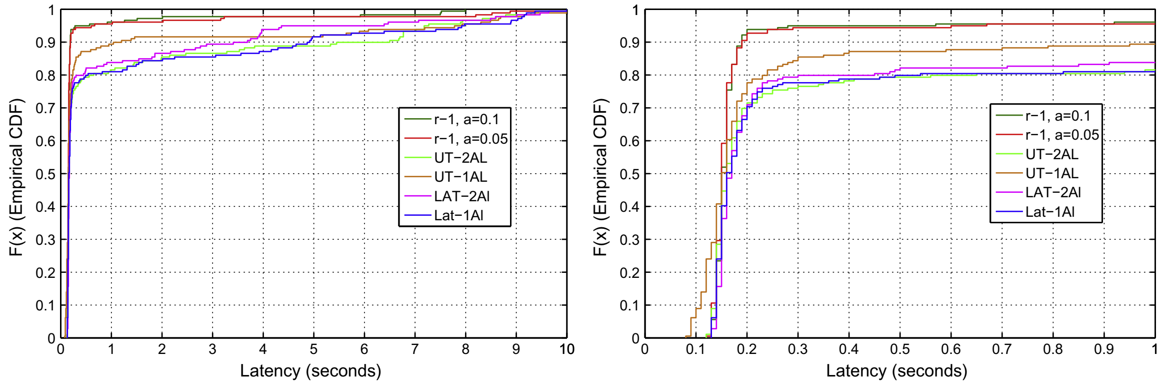


Fig. 11. The empirical CDF of the Latency for the bursty workload, and for an evaluation time period of 1 min. On the left the whole plot and on the right a zoom for Latency values less than 1 s.

Table 3
The allocation cost for the bursty workload in the 1-min case.

	Total hours	Total cost	Cost/hour
r-1, $a = 0.1$	19	1.615	0.538
r-1, $a = 0.05$	19	1.615	0.538
UT–2Al	28	2.38	0.793
UT–1Al	23	1.955	0.651
LAT–2Al	29	2.465	0.821
LAT–1Al	27	2.295	0.765

Table 4
The allocation cost for the bursty workload and evaluation time of 5 min.

	Total hours	Total cost	Cost/hour
r-1, $a = 0.1$	18	1.53	0.51
r-1, $a = 0.05$	17	1.445	0.481
UT–2Al	17	1.445	0.481
UT–1Al	18	1.955	0.51
LAT–2Al	15	1.275	0.425
LAT–1Al	14	1.19	0.39

VM allocation (see Fig. 12) and by the lower allocation cost (see Table 4). However, for the threshold based policies, the lower allocation cost impacts negatively on the latency, the responsiveness and, therefore, the capability to satisfy the SLO (see Fig. 13). Only the r-1 ($a = 0.1$) policy performs better than in the 1-min case, with a probability to satisfy the SLO equal to 1. The r-1 ($a = 0.05$) had almost the same behavior. All the other policies are characterized by the probability of the system to satisfy the SLO ranging from 0.7 to 0.83 (see Fig. 13), while in the 1-min case the same probability was greater than 0.79.

Moreover, observing Latency CDF (Fig. 13), it is immediate to note that the distribution of the response time has a longer queue than in the 1-min case (Fig. 11). This means that the system did not react quickly to workload fluctuations and, when a burst of request arrived, ASP users experimented an higher response time for a longer time period.

Finally, we remark that the allocation cost decrease for all the threshold based allocation policies and in the specific the Latency based allocation policies allow to save about the 20% of the allocation cost (compared to the other strategies).

7. Conclusions

This paper brought to the community several contributions.

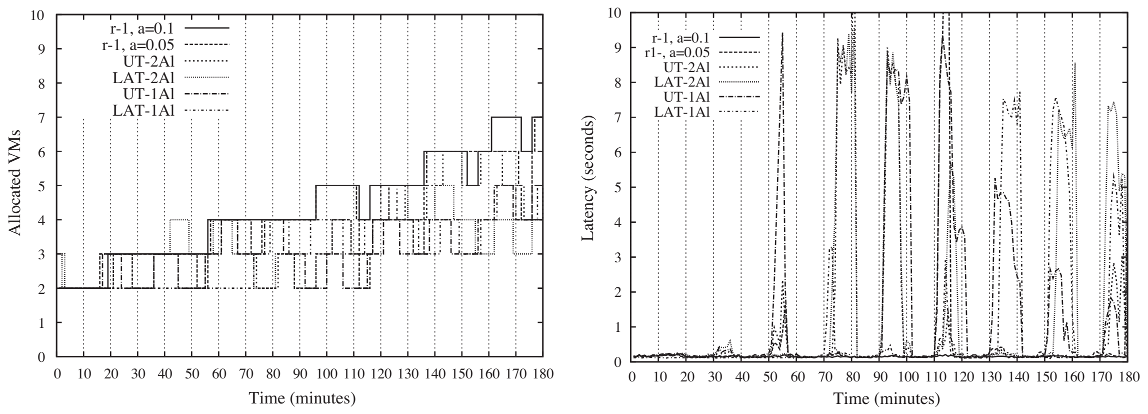


Fig. 12. The allocated VMs time series (left) and the Latency time series (right) for the bursty workload, and for an evaluation time period of 5 min.

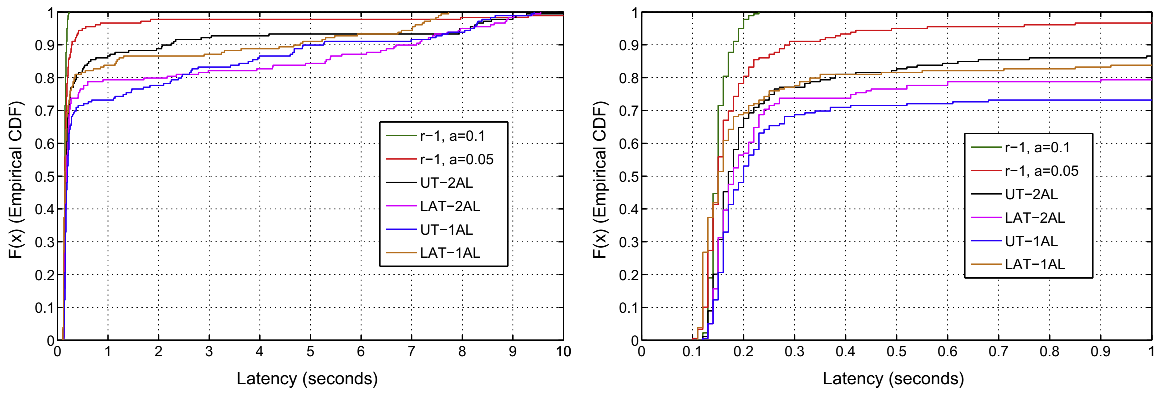


Fig. 13. The empirical CDF of the Latency for the bursty workload, and for an evaluation time period of 5 min. On the left the whole plot and on the right a zoom for Latency values less the 1 s.

First we provide a survey on features and services offered by seventeen of the most known IaaS public providers (see Table 1) and we classified IaaS providers features into a taxonomy, useful for end-users to understand and choose the more suitable and advantageous options in the wild and complex word of cloud providers offers.

Second we provided an implementation of the two autonomic service provisioning architectures using features and services of the Amazon Elastic Compute Cloud (EC2) Infrastructure. We share the implementation issues and lessons learned with the community. The experience developed during the prototype implementation phase confirmed that the maturity level of currently available cloud services, even though rapidly raising, is still quite low, and that an efficient ready-to-use solution for autonomic management of cloud systems seems still far to be available.

Third we proposed and implemented five resource allocation policies. One is based on custom workload prediction model and on a queueing network system model that allows to compute the needed resources. The other, leveraging on the Amazon Auto Scaling service, are based on the direct evaluation of the system latency and utilization, and determine the allocation or deallocation of new resources on the basis of the evaluation of fixed thresholds. All the proposed policies use a reactive approach.

Finally we set up a real testbed and we evaluated, under a realistic workload, the advantages and drawbacks of implementing and running the Partial ASP control versus the Limited ASP control architecture. The experimental results show that: (i) The proposed policies are capable to properly dimension the system resources making the whole system self-adaptable respect to the workload fluctuation. (ii) In case of ramp workload, the partial ASP control architecture, equipped with the r-1 policy allow to save up to 15% of resource cost, respect to the Limited ASP control solution equipped with the threshold based heuristics. (iii) In case of bursty workload and system adaptation evaluated every 1 s, the partial ASP control architecture allow to save from 17% to 32% of resources cost respect to the Limited ASP control solution. The only case where the threshold based heuristics allow to have better performances is when the system is stressed with a bursty workload and the adaptation decision is evaluated every 5 min. In this scenario the Limited ASP control solution equipped with latency based allocation policies allow to save the up to the 20% of cost respect the other solutions.

The take home message from this research is that, in order to facilitate ASP (of any size and budget) to really use the cloud computing paradigm, cloud providers should offer a stronger, more sophisticated and highly customizable support to autonomic resource management.

Acknowledgment

This research has been partially supported by a Grant from the AWS in Education program, 2010–2011 (http://www.ce.uniroma2.it/cloud/).

References

[1] W. Zeng, Y. Zhao, J. Zeng, Cloud service and service selection algorithm research, in: Proc. of 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC '09), ACM, 2009, pp. 1045–1048.
[2] H.C. Lim, S. Babu, J.S. Chase, S.S. Parekh, Automated control in cloud computing: challenges and opportunities, in: Proc. of 1st Workshop on Automated Control for Datacenters and Clouds (ACDC '09), ACM, 2009, pp. 13–18.
[3] H. Nguyen Van, F. Dang Tran, J.-M. Menaud, Autonomic virtual resource management for service hosting platforms, in: Proc. of 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD '09), IEEE Computer Society, 2009, pp. 1–8.
[4] M. Litoiu, M. Woodside, J. Wong, J. Ng, G. Iszlai, A business driven cloud optimization architecture, in: Proc. of 2010 ACM Symposium on Applied Computing (SAC '10), ACM, 2010, pp. 380–385.
[5] A. Andrzejak, D. Kondo, S. Yi, Decision model for cloud computing under sla constraints, in: Proc. of 18th IEEE/ACM Int'l Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '10), IEEE Computer Society, 2010.
[6] Y. Hu, J. Wong, G. Iszlai, M. Litoiu, Resource provisioning for cloud computing, in: Proc. of 2009 Conf. of the Center for Advanced Studies on Collaborative Research, CASCON '09, ACM, 2009, pp. 101–111.
[7] S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, B. Rochwerger, Oceano – SLA based management of a computing utility, in: Proc. of 7th IFIP/IEEE Int'l Symposium on Integrated Network Management, pp. 855–868.
[8] M. Kallahalla, M. Uysal, R. Swaminathan, D.E. Lowell, M. Wray, T. Christian, N. Edwards, C.I. Dalton, F. Gittler, SoftUDC: a software-based data center for utility computing, IEEE Comput. 37 (2004) 38–46.
[9] J. Xu, M. Zhao, J.A.B. Fortes, R. Carpenter, M.S. Yousif, On the use of fuzzy modeling in virtualized data center management, in: Proc. of 4th Int'l Conf. on Autonomic Computing (ICAC '07), IEEE Computer Society, 2007.

[10] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood, Agile dynamic provisioning of multi-tier internet applications, ACM Trans. Auton. Adapt. Syst. 3 (2008) 1–39.
[11] E. Casalicchio, L. Silvestri, Medium/long term SLA provisioning in cloud-based service providers, in: Proc. of 2nd Int'l ICST Conf. on Cloud Computing (CloudComp 2010), 2010.
[12] Z. Gong, X. Gu, W. John, Press: Predictive elastic resource scaling for cloud systems, in: Proc. of 2010 Int'l Conf. on Network and Service Management, IEEE, 2010, pp. 9–16.
[13] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, E. Turrini, Qos-aware cloud, IEEE Int. Conf. Cloud Comput. 0 (2010) 321–328.
[14] M. Mao, J. Li, M. Humphrey, Cloud auto-scaling with deadline and budget constraints, in: Proc. of the 11th ACM/IEEE International Conference on Grid Computing (Grid 2010), 2010.
[15] G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, C. Pu, Generating adaptation policies for multi-tier applications in consolidated server environments, in: Proc. of 5th Int'l Conf. on Autonomic Computing (ICAC '08), IEEE Computer Society, 2008, pp. 23–32.
[16] I. Cunha, J. Almeida, V. Almeida, M. Santos, Self-adaptive capacity management for multi-tier virtualized environments, in: Proc. of 10th IFIP/IEEE Int'l Symp. on Integrated Network Management (IM '07), pp. 129–138.
[17] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem, Adaptive control of virtualized resources in utility computing environments, SIGOPS Oper. Syst. Rev. 41 (2007) 289–302.
[18] Y. Song, Y. Li, H. Wang, Y. Zhang, B. Feng, H. Zang, Y. Sun, A service-oriented priority-based resource scheduling scheme for virtualized utility computing, in: Proc. of 15th Int'l Conf. on High Performance Computing (HiPC '08), LNCS, vol. 5374, Springer, 2008, pp. 220–231.
[19] Y. Song, H. Wang, Y. Li, B. Feng, Y. Sun, Multi-tiered on-demand resource scheduling for vm-based data center, in: Proc. of 9th IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGRID '09), IEEE Computer Society, 2009, pp. 148–155.
[20] Y. Chen, T. Wo, J. Li, An efficient resource management system for on-line virtual cluster provision, in: Proc of IEEE Int'l Conf. on Cloud Computing (CLOUD '09), IEEE Computer Society, 2009, pp. 72–79.
[21] M. Andreolini, S. Casolari, M. Colajanni, Dynamic load management of virtual machines in a cloud architectures, in: Proc of 1st Int'l Conf. on Cloud Computing (ICST CLOUDCOMP 2009), 2009.
[22] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of 2nd Int'l Symposium on Networked Systems Design & Implementation (NSDI '05), USENIX Association, 2005, pp. 273–286.
[23] E. Kalyvianaki, T. Charalambous, S. Hand, Self-adaptive and self-configured cpu resource provisioning for virtualized servers using Kalman filters, in: Proc. of 6th Int'l Conf. on Autonomic Computing (ICAC '09), ACM, 2009, pp. 117–126.
[24] E. Casalicchio, L. Silvestri, Architectures for autonomic service management in cloud-based systems, in: Proc. of IEEE symposium on Computers and Communications, Workshop on Management of Cloud Systems (MoCS 2011), 2011.
[25] V. Cardellini, E. Casalicchio, F. Lo Presti, L. Silvestri, Sla-aware resource management for application service providers in the cloud, in: Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on, pp. 20 –27.
[26] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (2003) 41–50.
[27] Wikibench, The Realistic Web Hosting Benchmark, 2011. <http://www.wikibench.eu/>.
[28] I. Brandic, Towards self-manageable cloud services, Comput. Software Appl. Conf. Annu. Int. 2 (2009) 128–133.
[29] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F.L. Presti, R. Mirandola, Moses: a framework for QoS driven runtime adaptation of service-oriented systems, IEEE Trans. Software Eng. 99 (2011).
[30] A. Beloglazov, R. Buyya, Energy efficient allocation of virtual machines in cloud data centers, in: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pp. 577–578.
[31] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Pract. Exp. 41 (2011) 23–50.
[32] Cloudsigma, 2011. <http://www.cloudsigma.com>.
[33] Vpsnet, 2011. http://vps.net.
[34] Flexiant Flexiscale Public Cloud, 2011. <http://www.flexiant.com/products/flexiscale/>.
[35] Elastichosts, 2011. http://www.elastichosts.com/.
[36] Locaweb Cloud Server Pro, 2011. http://www.locaweb.com.br/solucoes/cloud-computing.html.
[37] Rsaweb Cloud Servers, 2011. <http://www.rsaweb.co.za/cloud-servers>.
[38] Storm on Demand, 2011. <http://www.stormondemand.com/cloud-server/>.
[39] Reliacloud, 2011. <http://www.reliacloud.com/>.
[40] Amazon Web Services, 2011. <http://aws.amazon.com/>.
[41] At& t Synaptic, 2011. <https://www.synaptic.att.com/clouduser/synaptic%_welcome.htm>.
[42] Gogrid, 2011. <http://www.gogrid.com/>.
[43] Joyentcloud, 2011. <http://joyentcloud.com>.
[44] Layeredtech, 2011. <http://layeredtech.com/cloud-computing/virtual-machine-hosting/>.
[45] Opsource, 2011. <http://www.opsource.net/Services/Cloud-Hosting>.
[46] Rackspace Hosting, 2011. <http://www.rackspacecloud.com/>.
[47] Slicehost, 2011. <http://www.slicehost.com>.
[48] Terremark vcloud Express, 2011. <http://vcloudexpress.terremark.com/>.
[49] Cloudharmony: Benchmarking the Web, 2011. <http://cloudharmony.com/>.
[50] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, O. Fox, D. Patterson, Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement Tools for Web 2.0, 2008.
[51] G. Urdaneta, G. Pierre, M. van Steen, Wikipedia workload analysis for decentralized hosting, Comput. Networks 53 (2009) 1830–1845.
[52] D. Mosberger, T. Jin, httperf – a tool for measuring web server performance, SIGMETRICS Perform. Eval. Rev. 26 (1998) 31–37.
[53] Httperf: The httperf http Load Generator, 2011. <http://code.google.com/p/httperf/>.

Emiliano Casalicchio, Ph.D., is a researcher in the Department of Computer Science, Systems and Production of the University of Roma "Tor Vergata", Italy. Since 1998 his research is mainly focussed on large scale distributed systems with a specific emphasis on performance oriented design of algorithms and mechanisms for resource allocation and management. Domains of interest have been locally distributed web servers, enterprise grid, mobile systems, Service Oriented Architectures and Cloud Systems He is authors of about 70 publications on international journals and conferences, and He serves as reviewer for international journals and conferences (IEEE, ACM, Elsevier, Springer, etc.). The research of E. Casalicchio is and has been founded by the Italian Ministry of Research, CNR, ENEA, the European Community and private companies. Moreover he is and has been involved in many Italian Research project and EU Research projects.

Luca Silvestri is currently a Ph.D. student in Computer Engineering at the University of Roma "Tor Vergata". He received in 2009 the Laurea degree (Master degree) in Computer Engineering from the University of Roma, Tor Vergata. His research interests are in the field of performance oriented design, focusing particularly on cloud computing, service oriented computing, reactive and proactive monitoring of distributed systems, service selection algorithms in SOA.