Optimized Cloud Resources Management Based on Dynamic Scheduling Policies and Elasticity Models

Nikolaos Chelmis, Dimosthenis Kyriazis, Marinos Themistocleous
Department of Digital Systems
University of Piraeus
Piraeus, Greece
e-mail: {helnik, dimos, mthemist}@unipi.gr

***Abstract*—Nowadays, the interest on Cloud Computing as a technical and business best practice has grown to great length. There is a huge pool of available cloud applications and services offered to end users. As application requirements, reflected to resources requirements (i.e., network, storage, computing capacity), are set by the application provider, a key issue relates to the resulting elasticity needs and their modeling. In addition to elasticity needs, application providers aim to maximize their customer base while considering the associated costs. To this end, business models are needed in order to attract customers while considering cost constraints. Their aim is to optimize the performance of the "investment" for resources compared to the expected number of customers. Nevertheless, the latter is directly linked to the provided quality of service and users' quality of experience. To this direction, in this paper, we present a mechanism that dynamically maps scheduling policies with the planned and estimated resources based on varying needs.**

Keywords-Cloud Computing; Infrastructure as a Service (IaaS); Elasticity; Monitoring; Scheduling Policy; Quality of Service.

I. INTRODUCTION

Cloud computing as a whole has rapidly evolved and became one of the most challenging paradigms of Information Technology. It has gained popularity for its ability to enable fast and effective access to large pools of virtualized resources and services that are dynamically provisioned to adjust to variable workloads and usage optimization [1]. This pool of resources is typically exploited by a pay-as-you-go [2] pricing model with the cost of using a cloud asset depending on the resources consumed. To this direction, cloud computing offers mechanisms to automatically scale applications in order to meet user's needs, thus making possible for them to rapidly adapt their resources to the workload minimizing the cost of overprovisioning.

There are three main classes in the cloud services stack which are generally agreed upon [1]: (i) Infrastructure as a Service (IaaS) where the provider sells access to computers upon which any software can run. The resources, which are in most cases virtual, are expressed in terms of processing power, memory, storage capacity, etc. (ii) Platform as a Service (PaaS), where an environment for application developers to deploy their code is offered. (iii) Software as a Service (SaaS) where customers pay to use an application that is hosted on a remote provider. The service provider manages the software and the underlying infrastructure. The main focus of this paper is the IaaS layer since it has great potential in further revolutionizing the way compute resources are provisioned and consumed.

While scalability enables smooth application execution even when number of users grows, two approaches are mainly used to make new resources available [2]: (i) Vertical scalability (scale up/down) increases or decreases the resources (commonly the CPU number, the memory or bandwidth) of an element in the system. (ii) Horizontal Scalability (scale in/out) replicates or removes instances of system elements (usually Virtual Machines - VMs) to balance the workload. Elasticity is the ability to scale an infrastructure on demand within minutes (seconds in an optimum case) to avoid under-utilization and over-utilization of resources. Scalability is a prerequisite for elasticity, but it does not take into consideration how fast or how often scaling actions can be performed thus it is not directly related to how well the actual resource demands are matched by the provisioned resources an any point of time [4].

What is more, modern business trends highlight opportunities for service provisioning via cloud infrastructure to the end users. Three main models have prevailed: (i) direct service provision to the end user (e.g., Dropbox), (ii) use of cloud infrastructure for an organization's internal purposes (e.g., internal network of a bank) and (iii) use of cloud infrastructure to provide a service to the end users.

The current paper focuses on the third model, which is being exploited by application providers / owners - referred as brokers. The service provided may be any software system, consisting of one or more components. Its architecture, in terms of components, interfaces and logic, is considered to be known and can be precisely described by the provider. As application requirements, regarding the resources, are formulated by the aforementioned brokers, a key issue relates to the resulting elasticity needs and their modeling. Elasticity and requirements are dependent on the following parameters: (i) application's nature (i.e., use of multiple processors), (ii) usage (i.e., variable exponential growth of end-users) and (iii) infrastructure vendors (i.e., resource availability). In order to analyze elasticity requirements, models that meet the above parameters and allow use of the analysis results to provide resources based on demand are required.

In addition to elasticity issues, application providers / owners aim to maximize their customer base while considering the cost. Towards this direction, dynamic scheduling policies that suggest ways to attract customers are required, with an ultimate goal to optimize the "investment" made for resources compared to the foreseen number of customers. Nevertheless, the expected number is directly linked to the Quality of Service (QoS) provided and users' Quality of Experience (QoE). For example, if provisioned resources follow demand, a number of users will have to wait for resource's availability and hence the use of service.

In economic terms, the overprovisioned resources are easily measured but underprovisioned is way harder. End users who were unable to use the application or experienced performance degradation (lower levels of QoS) may never become returning customers and discredit the service. Based on the above, a dynamic model, efficient for business, regarding both current users and expected ones compared to elasticity needs, is required. Nonetheless, the efficiency of scheduling policy needs to be linked to the required resources that accommodate the users' expectation regarding QoS. The linkage / mapping should allow elasticity models to be followed compared to different scheduling policies (aggressive, passive, neutral) resulting in the optimal resource management. Furthermore, scheduling policies should be able to be switched a dynamic way during runtime. In this paper, we present a mechanism (overview depicted in Figure 1) enabling the latter, which has been developed and validated in the framework of the PinCloud project [5] with different stakeholders in the eHealth domain.

A mechanism, the overview of which is presented in Figure 1, enabling the latter is presented in this paper.

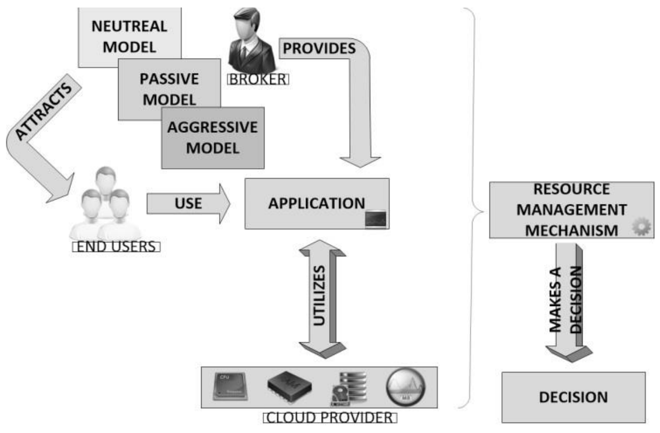

Figure 1. Use of Mechanism

The added value of the proposed mechanism lies on the incorporation of techno-economic factors for the management of resources in cloud environments. The mechanism enables the outcomes of a business - expressed through the corresponding models - simulation process to be considered during runtime in order to trigger resource provisioning decisions. Runtime adaptation takes place based on both the business goals of the application provider and the emerging requirements from the end-users.

The remainder of this paper is structured as follows: Section 2 presents the related work that solves the problem of elasticity in cloud platforms. Section 3 proposes three dynamic scheduling policies, while in Section 4 the architecture of the proposed mechanism's architecture is being analyzed. In Section 5, mechanism is evaluated and the results are discussed. Finally, Section 6 concludes our work and discusses open areas.

II. RELATED WORK

Elasticity, a term originally defined in physics, is considered one of the central attributes of the cloud. Cloud providers use the term in advertisements and even in the naming of products or services and implement it in different degrees.

Amazon EC2 [6] allows VMs to scale vertically in order to reciprocate with resource requirements. Customers can change resource requirements; but, in order to achieve horizontal scalability a cluster of VMs must be created and configured according to needs. This is a manual process, which does not include application configuration of a VM. Amazon EC2 enables the preparation of the VM but not the automatic configuration, which is a major requirement for an elastic platform.

Microsoft's Windows Azure [7] consists of three main components providing a set of services to cloud users for running applications and storing data. Azure offers specific VM instances with predefined sizes (CPU, Memory). Automatic scaling is offered through application rules via a configuration file specified by users.

Google App Engine [8] is optimized for web applications. It handles the deployment, monitoring and launching of service instances making use of Google's core engine. Automatic scalability is transparent to the application providers / owners with no option for the developer to write his own scaling rules based on application's specific needs.

Amazon offers a service called Spot Instances [9], an elasticity solution based on cost. Spot Instances are virtual servers sold per hour via an action. Based on bids and available capacity Amazon determines a price (Spot Price) and if the maximum bid price exceeds the current Spot Price the request is fulfilled.

RightScale [10] is an application management platform for clouds addressing the Infrastructure-as-a-Service (IaaS) service model. RightScale provides control and elasticity capabilities, assisting the user to design, deploy and manage applications on a number of underlying clouds (i.e., Amazon, Rackspace, private solutions like CloudStack, OpenStack and others). Various monitoring metrics are used and users can define alerts based on these metrics.

OnApp [11] is a software package for IaaS cloud providers. It states that it enables replication and redimensioning on VMs allowing changes manually or automatically, based on rules defined by user and metrics obtained by the monitoring mechanism. Lim et al. [12] proposed an automatic mechanism based on a target range for a specific system metric, rather than a threshold to trigger actions. The key point is that the system reacts when the defined metric is outside the range, reducing resources allocations.

Internal provider resources management and use of elasticity is addressed by Meng et al. [13]. According to the authors, management tasks (like VM creation, migration, etc.) are expensive in terms of computation and more likely to occur in bursts. Lack of resources to handle this workload will affect users' applications performance. Based on this observation, TIDE: a self-scaling framework for virtualized data center management, was proposed. The main idea is to treat management workload the way application workload would be treated. When bursts in management workload are encountered by TIDE, it powers up dynamically additional server management instances. When burst subsides, management instances' physical resources can be used for user application workloads.

Comparing to the approaches discussed above, the proposed resource management mechanism addresses the issues of resource management from a broker's perspective. To take full advantage of elasticity, while an elastic infrastructure provides the required functionality, business adoption is constrained by the associated costs compared to the actual and foreseen service usage. Applications should have the ability to dynamically exploit the infrastructure according to workload changes in a dynamic and cost-efficient way. The presented mechanism proposes resource management (in terms of resource requirements specification) taking into consideration both the forecasted elasticity needs of the service in relation with the scheduling policy being followed by the service owner.

III. SCHEDULING POLICIES

Application providers / owners use different business models in order to maximize their profit. Among other parameters (like cost for supplies, man hours etc.) these models also include scheduling policies which take into consideration the resources that application needs to operate inside the predefined QoS. Dynamic scheduling policies reflect how the current and the forecasted number of users relate to the application needs (and thus resource requirements) for specific predefined QoS levels. Given that the provided service can be charged based on different models, three main policies are proposed (i) Aggressive, (ii) Passive, and (iii) Neutral. Each model guarantees different response time, therefore aims to different customer base size. Concept of man hours, cost for supplies, and others are out of the scope of this work; therefore, they are not taken into consideration. The core set of parameters incorporated in the considered scheduling policies follow:

1. Cloud Resources
   - (a) Type (CPU, Memory)
   - (b) Availability
2. Usage
   - (a) Number of users currently using application
   - (b) Target users
   - (c) Number of acquired users
3. Cost
   - (a) Cost of acquiring resources
   - (b) Gain from users acquired

The above parameters are directly and dynamically linked / related to each other. For instance, increase of users means more revenue but it also means increase in response time. In order for the response time to be maintained within specific limits, additional resources may need to be acquired. Accordingly, decrease in the number of users means lower response time, reducing the need for resources.

In order to create the dynamic scheduling policies, the concepts of minimum (t^L) and maximum (t^U) response time are introduced. Response time should never exceed any of these two limits. Another key element taken into consideration, which can alter application's response time, is spin-up time [14]. The amount of time needed, since initial acquisition request, for required resources to be ready for use is called spin-up time. Weighting factors are included in the scheduling policies in order to define the requested amount or resources. All the attributes used for the creation of the scheduling policies are summarized in Table 1.

TABLE 1. ATTRIBUTES USED FOR SCHEDULING POLICIES

Component	Description
Goal Users	Number of target Users
RT	Response Time
t^L	Best – Minimum Response Time
t^U	Worst – Minimum Response Time
t^{SU}	Spin Up Time
initDep	Initial Deployment
res	Resources
a,b,c	Weighting Factors
n	Predifined amount of time

Based on the above, the following paragraphs present the scheduling policies taken into account in the current work: (i) Passive, (ii) Neutral, and (iii) Aggressive. For each one, a mathematical equation describes how the new value for the resources (*f(x)*) is calculated.

A. Passive Scheduling Policy

This model ensures that users do not experience violation of the maximum response time but allows response times to be close to the maximum ones. To achieve that the difference between current response time and minimum response time plus a predefined amount of time, referred as n, is examined and if current response time is higher more resources are requested. Bursting (i.e., extreme growth in user numbers) is also taken into consideration as current response time is contradicted to maximum response time. Spin-up time is not taken into consideration at this model. Finally, to avoid overprovisioning current response time is compared to minimum response time and if it is less then all acquired resources are released. This model is described in (1) and its flowchart is illustrated in Figure 2.

$$f(x) = \begin{cases} a * res, & RT > t^L + n \\ b * res, & RT \geq t^U \\ initDep, & RT < t^L \end{cases} \quad (1)$$

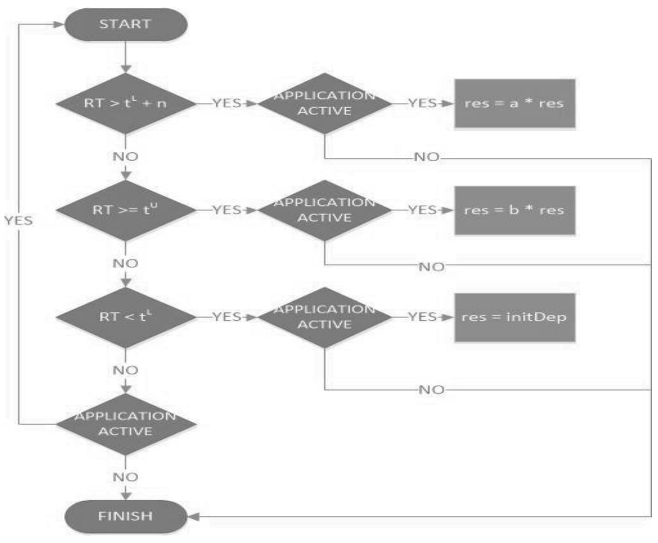

Figure 2. Passive Scheduling Policy

B. Neutral Scheduling Policy

This model's target is to prevent users, even in bursting, to come too close to maximum response time. It follows same logic as the above model comparing current response time with minimum and maximum. In contrast with passive model, this one takes into consideration spin-up time in cases of bursting. Spin-up time is subtracted from maximum response time, ensuring that the requested resources will be available before maximum response time is reached. Furthermore, if the number of target users is reached more resources are acquired as a bonus to users. Overprovisioning is avoided the same way as in passive model. This model is described in (2) and its flowchart is illustrated in Figure 3.

$$f(x) = \begin{cases} a * res, & RT > t^L + n \\ a * res, & Users = GoalUsers \\ b * res, & RT \geq t^U - t^{SU} \\ initDep, & RT < t^L \end{cases} \tag{2}$$

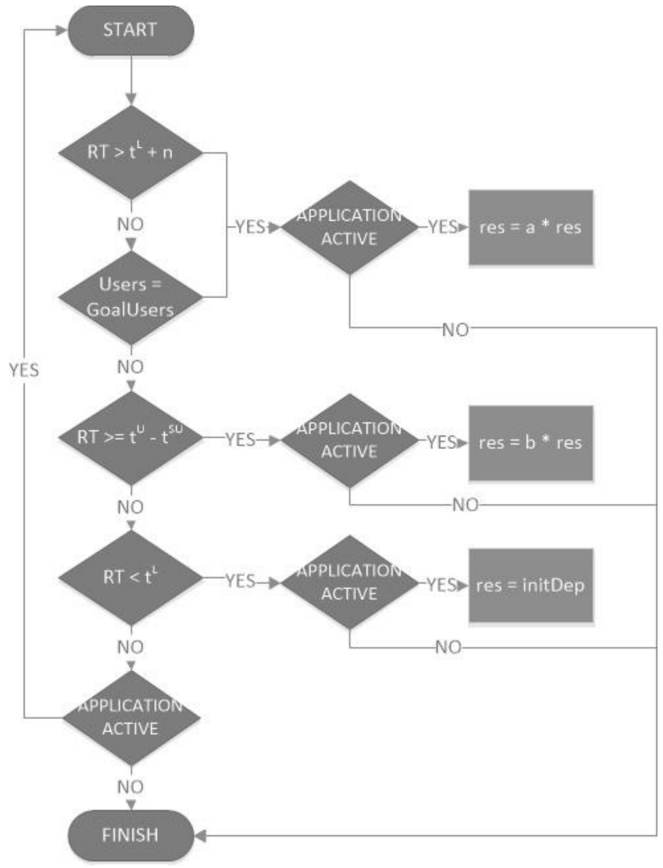

Figure 3. Neutral Scheduling Policy

C. Aggressive Scheduling Policy

This model makes sure users are as close as possible to minimum response time. It is similar to Neutral model although it's weighting factors are bigger. Furthermore, a predefined amount of time (n) is added to spin-up time, ensuring that even in heavy bursts users will not reach close to the maximum response time. Overprovisioning is again avoided by comparing application's response time with minimum response time. This model is described in (3) and its flowchart is illustrated in Figure 4.

$$f(x) = \begin{cases} a * res, & Users = GoalUsers \\ b * res, & RT \geq t^L + n \\ c * res, & RT \geq t^U - (t^{SU} + n) \\ initDep, & RT < t^L \end{cases} \tag{3}$$

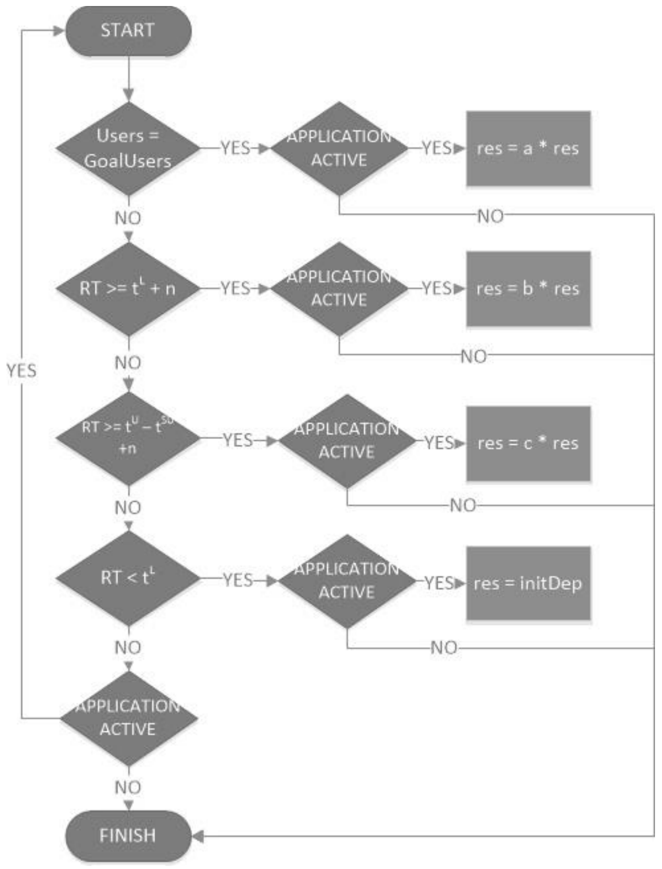

Figure 4. Aggressive Scheduling Policy

The aforementioned policies are supposed to be part of an application's provider / owner business model. According to what QoS application provider / owner promises on his business model the corresponding policy should be followed. Furthermore, weighting factors and the predefined amount of time (n) are different in each policy. That means that weighting factors and predefined amount of time of passive scheduling policy are the lowest while aggressive one's are the bigger.

IV. RESOURCE MANAGEMENT MECHANISM

The goal of the mechanism is to make it completely independent to the process of starting the application, thus making it possible to start an application and join the mechanism later if needed. Secondly, the mechanism should allow application providers /owners to change / switch between scheduling policies in a dynamic way during runtime.

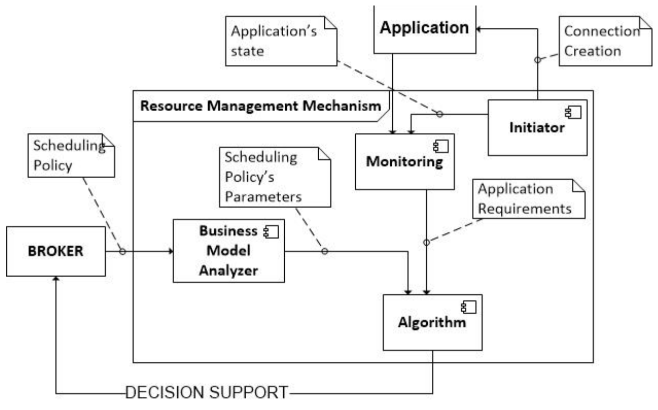

Figure 5. Mechanism's Components

As depicted in Figure 5, the proposed modular architecture consists of four main building blocks / services:

1. **Initiator**: This service creates a connection with the application and obtains its current requirements and current usage (e.g., number of users, number of requests, etc.). After relaying this information to the monitoring component it pauses.
2. **Monitoring**: The goal of this service is to monitor the application's state in terms of both application-level (e.g., number of current users and current response time) and resource-level (e.g., CPU usage) metrics. All the information is passed to the algorithm.
3. **Business Model Analyzer**: The analyzer obtains the current scheduling policy, contained in the business model used by the provider and relays the information to the algorithm. Furthermore, number of target users (referred as goal users in the above section) is also relied to the algorithm.
4. **Algorithm**: Mechanism's logic which takes into consideration both the forecasted elasticity needs of the service and the scheduling policy which the service owner follows. Based on the collected information from the aforementioned components it estimates response time for the acquired users. Since business models, thus scheduling policies, can be changed by application provider / owner during runtime number of target users can also be changed. Each time number of target users is reached application provider / owner can set a new goal number and the mechanism will estimate the response time.

V. EVALUATION

The aim of experimentation is to evaluate the proposed mechanism in a real environment. To this end, the experiment was distributed across three different locations in Europe: EPCC (Edinburgh), HLRS (Stuttgart) and PSNC (Poznan) provided by the BonFire cloud infrastructure [15]. The connection between individual sites was over best effort Internet (Cloud over Internet) besides the connection between UK-EPCC and PL-PSNC sites. The latter was established with GEANT Bandwidth on Demand (BoD) system (AutoBAHN BoD version 2.1.1[16]), which is a service for dynamic bandwidth provisioning across multiple networks (guaranteed bandwidth).

The experimentation infrastructure that had been used consisted of in total of 30 VMs acting as servers (10 VMs have been deployed in each of the following sites: Edinburg, Stuttgart and Poznan). Given that the goal of the epxerimenation was to obtain information with respect to response times, the clients have been deployed in 60 VMs in different sites so as to obtain information for cross-site response times. The response time was measured through Apache JMeter.

For the purposes of the experimentation, a simple Java servlet-based Service Oriented Architecture (SOA) service was developed. Upon receiving an HTTP GET request, the service calculates a million random integers ranging from zero to one thousand. Application does not store user state, thus making easy the service requests to be spread among servers running the same code. The only purpose of the service is to represent highly parallelizable computation task. Requests arrive at a load balancer node, which is included to the deployment, to allocate them to servers. Application was deployed in each client (side).

A. *Evaluation Results*

The aim of our evaluation was to validate the operation and efficiency of the algorithm in different cases and for different metrics. In the experiment presented below, a total of 200 users were used, while policies of neutral scheduling policy where applied with minimum response time set to 200 and maximum set to 15000 milliseconds.

The experimentation data that have been collected are depicted in Figure 6. The information includes active users during the experiment period, response time, CPU and memory utilization. It can be observed that number of users was increased linearly. Response time though was not increasing with the same pattern since it is also dependent on CPU and memory usage. The increase on number of users entailed increase on resources; thus, the response time was maintained steady. As number of users kept on increasing response time started to increase leading to more CPU usage. One should take into consideration application's nature: It requires computation power but not memory usage since it requires no cashing. As observed in Figure 6 memory has an increase at the beginning of the experiment, but then remains steady, while CPU resources are increased, proving that resource management was correct. Response time contradicted, only with the number of users is illustrated in Figure 7.

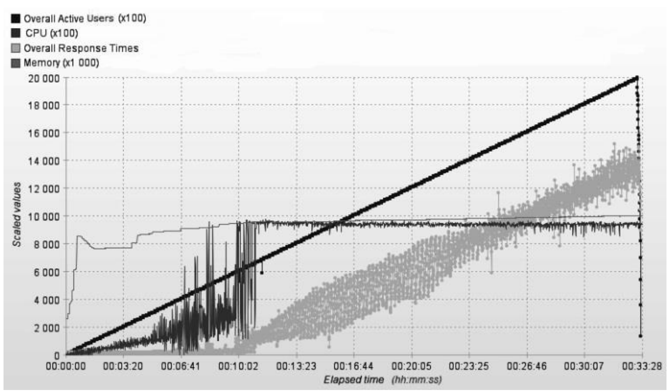

Figure 6. Collected Experimentation Data

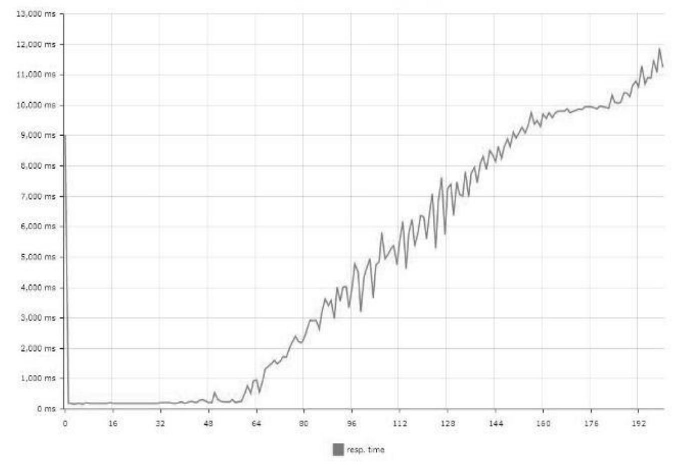

Figure 7. Response Time VS users

The distribution of response times during the evaluation experiment is illustrated in Figure 8. With a closer observation, one can notice that, regardless the linear growth of users, the response time was not also growing linearly but was maintained steady for an amount of time. Another important observation is that the distribution of lower response times (i.e., 300-900ms) is enough smaller. That is easily explained: Since neutral's model policies were applied as long as users were closer to minimum response time no resources were acquired so with number of users growing response time also grew.

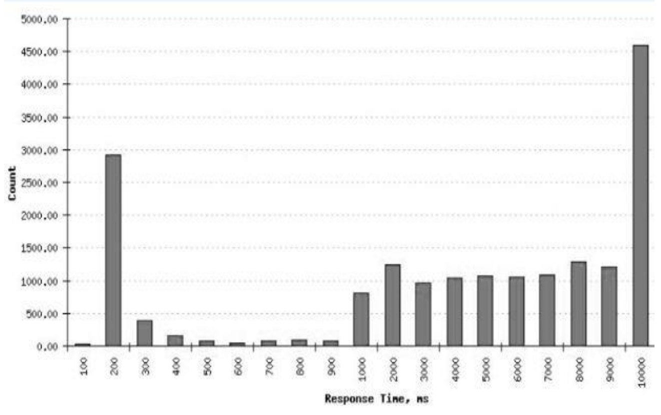

Figure 8. Response Time Distribution

Figure 9 presents a comparison between estimated and actual response time. As illustrated in the figure, the proposed mechanism delivers results very close to reality. As a result, the mechanism identifies the forecasted number of users in an accurate way as required to state the resources required for the number of users.

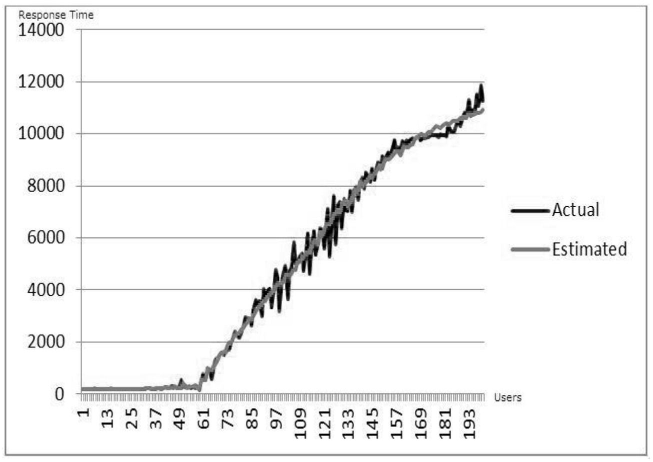

Figure 9. Actual VS Estimated Repsonse Time

In order to illustrate more clearly whether the actual coincides with the estimated response time in Figure 10, a histogram of residuals resulting from the comparison between them is illustrated.

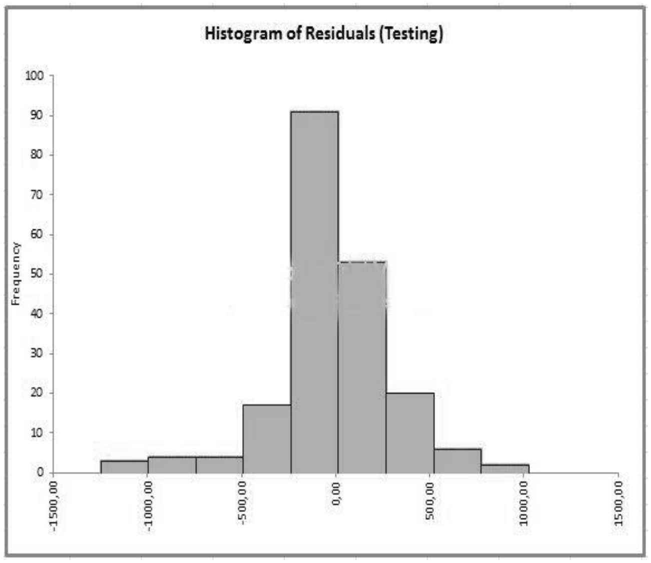

Figure 10. Histogram of Residuals

As shown, the highest residues occur around zero, thus proving the correctness of the prediction results. It is, therefore evident that the mechanism determines the number of users compared to the required resources modeled with precision.

VI. CONCLUSION AND FUTURE WORK

The objective of this paper was to present a new mechanism for the optimum cloud resources management based on dynamic scheduling policies and elasticity needs. Initially, dynamic scheduling policies regarding both current users and expected ones compared to elasticity needs were proposed. Application's response time was compared to the maximum and minimum accepted response time defined in the QoS. Furthermore, a mechanism that maps current and expected users with resource needs while taking into consideration the scheduling policies was introduced. The proposed mechanism is completely independent from the application and can be deployed while an application is active. Service provider can change his scheduling policy during runtime without affecting neither application's nor mechanism's performance.

ACKNOWLEDGMENT

The research leading to the results presented in this paper has received funding from the European Union and the Greek National Strategic Reference Framework Programme (NSRF 2007-2013), Project PinCloud under grant agreement number "11SYN_6_1013_TPE".

REFERENCES

[1] Grance, P.M.a.T., NIST Definition of Cloud Computing, Version 15. 2011, NIST: http://csrc.nist.gov/groups/SNS/cloud-computing. [retrieved: January 2015]

[2] Suleiman, B., Sakr, S., Venugopal, S., and Sadiq, W. (2012), "Trade-off analysis of elasticity approaches for cloud-based business applications", In Web Information Systems Engineering-WISE 2012 (pp. 468-482). Springer Berlin Heidelberg.

[3] Rimal, B. P., Choi, E., and Lumb, I. (2009, August), "A taxonomy and survey of cloud computing systems", In INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on (pp. 44-51). Ieee.

[4] Herbst, N. R., Kounev, S., and Reussner, R. (2013, June), "Elasticity in Cloud Computing: What It Is, and What It Is Not", In ICAC (pp. 23-27).

[5] PinCloud Project. Available from: http://pincloud.med.auth.gr/en. [retrieved: February 2015]

[6] Amazon. Amazon EC2. Available from: http://aws.amazon.com/ec2/. [retrieved: January 2015]

[7] Microsoft Windows Azure. Available from: http://azure.microsoft.com/enus/documentation/. [retrieved: January 2015]

[8] Google App Engine. Available from: https://developers.google.com/appengine/?csw=1. [retrieved: January 2015]

[9] Amazon EC2 Spot Instances. Available from: http://aws.amazon.com/ec2/purchasing-options/spot-instances/. [retrieved: January 2015]

[10] RightScale. Available from: http://www.rightscale.com/. [retrieved: January 2015]

[11] onApp. Available from: http://onapp.com/. [retrieved: January 2015]

[12] Lim, H. C., Babu, S., Chase, J. S., and Parekh, S. S. (2009, June), "Automated control in cloud computing: challenges and opportunities", In Proceedings of the 1st workshop on Automated control for datacenters and clouds (pp. 13-18). ACM.

[13] Meng, S., Liu, L., and Soundararajan, V. (2010, November), "Tide: achieving self-scaling in virtualized datacenter management middleware", In Proceedings of the 11th International Middleware Conference Industrial track (pp. 17-22). ACM.

[14] Brebner, P. C. (2012, April), "Is your cloud elastic enough?: performance modelling the elasticity of infrastructure as a service (iaas) cloud applications", In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (pp. 263-266). ACM.

[15] Kavoussanakis, K., et al., "Bonfire: The clouds and services testbed", In Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on (Vol. 2, pp. 321-326).

[16] Bonfire Documentation – Controlled Bandwidth with AutoBAHN. Available from: http://doc.bonfire-project.eu/R3.1/networking/autobahn.html. [retrieved: January 2015]