Autonomic Management of Elastic Services in the Cloud

Patrick Martin, Andrew Brown, Wendy Powley
School of Computing
Queen's University
Kingston, ON Canada
martin@cs.queensu.ca

Jose Luis Vazquez-Poletti
Facultad de Informatica
Universidad Complutense de Madrid
28040 Madrid, Spain
jlvazquez@fdi.ucm.es

***Abstract*—Cloud computing, with its support for elastic resources that are available on an on-demand, pay-as-you-go basis, is an attractive platform for hosting Web-based services that have variable demand, yet consistent performance requirements. Effective service management is mandatory in order for services running in the cloud, which we call elastic services, to be cost-effective. In this paper we describe a management framework to facilitate elasticity of resource consumption by services in the cloud. We extend our framework for services management with the necessary concepts and properties to support elastic services. A prototype implementation is described.**

Keywords-Management, Cloud Computing, Elastic Services

I. INTRODUCTION

Economic and technological factors have motivated the advent of on-demand computing infrastructures with companies such as Amazon, IBM, Microsoft and Google providing software, platforms and computing resources as services. This approach, framed in the *cloud computing* paradigm, is based on a pay-as-you-go model and its benefits are in the notion of *elasticity*, that is, the ability to scale capacity up or down to match consumer demands. Elastic resources are efficient for service providers since they limit up-front capital expenses and reduce the cost of ownership over time [1]. In this paper we refer to services running on the resources in the cloud as *elastic services*.

Systems management ensures the correct, efficient and secure operation of managed systems and applications. Traditional systems managements are not suited to the dynamic and complex nature of service-oriented architecture (SOA), which allows applications to be constructed of existing services in a dynamic manner. We previously proposed an autonomic, agent-based framework for services management that consists of the following three components [2]:

- A services management model to describe management tasks and goals.
- A method to generate management system components from specifications using the model.
- An infrastructure to allow the integration of management tasks and user interaction with the management system.

Elastic services offer additional management challenges. First, the components and resources involved in an elastic service are dynamic and change as demand for the service changes. This means that the management system must be able to automatically adapt to the changes in the underlying set of managed resources. It is also more difficult for a services management system to ensure predictable performance from the service and to support problem determination since the underlying resource configuration changes. Second, adding or removing resources, such as a virtual machine, to a service can involve significant delays. The services management system must be able to predict when a change in the makeup of the service will be necessary with enough lead time to minimize the impact of the change.

In this paper we adapt our services management framework to support elastic services. We extend the services management model to accommodate elastic services and discuss how our agent-based infrastructure can interface with the resource management systems provided by public and private clouds. We also discuss appropriate performance metrics on which the services management system can base its resource allocation decisions.

The remainder of the paper is structured as follows. Section II provides an overview of our services management framework, which consists of a services management model and supporting infrastructure. Section III gives an example scenario and we use it to describe how the framework is extended to accommodate elastic services. It suggests appropriate metrics for managing elasticity and discusses the status of a prototype implementation of the framework. Section IV examines related work. Section V summarizes the paper and provides suggestions for future work.

II. SERVICES MANAGEMENT FRAMEWORK

We view a management system for services as a collection of agents that interact through the processing and generation of event streams. Each agent performs a relatively simple function and all agents follow the same general behavior pattern. Agents accept new events on one or more event streams and the arrival of an event triggers local processing. The results of the processing can be the generation of a new event on an outgoing event stream and/or changes to the state of the managed system or the management system itself. These simple agents are combined into what we call management goal graphs in order to carry out complex management tasks.

A. Services Management Model

Our services management model consists of a set of constructs to specify management tasks. A complete definition of the model is provided elsewhere [2]. *Managed resources* are the services and components being managed. A managed resource provides a set of metrics to describe its state and performance and a set of configuration parameters that can be adjusted to affect its state and performance.

An *event* is the occurrence of a situation, or incident, within a service or the management system. Events may signify a deviation from typical behaviour or may report expected occurrences within the system such as the completion of a task. *Concrete events* happen in the managed system and are relayed into the management system via sensors. Examples of concrete events include the arrival of a client, the failure of a transaction, or the report of a performance metric such as throughput. *Inferred events*, on the other hand, occur in the management system and are derived from the current state of the managed system and the relevant history of event occurrences. *Event types* describe the common properties of a set of similar *event instances*.

It is assumed that a history of event instances is kept by the management system. The *event context* of an event instance e_i is a description of a condition on the relevant history for e_i. The combination of the arrival of e_i and its context is analogous to a pattern in complex event processing and matching that pattern causes the agent to act.

There are three main types of agents: Sensors, Actors and Effectors. *Sensors* monitor event streams and produce new event streams based on what they observe. *Actors* carry out a management function when triggered by the input of an event or by a user. *Effectors* impose changes for the management system on the managed resources. The behaviour of an agent is defined by its active *policy*, which specifies the input streams, the output stream, the event context(s) and, in the case of actors and effectors, the management action.

An *event stream* represents the flow of events from a source (external or agent) to one or more destination agents. All agents subscribed to a stream see all the event instances published on the stream. Agents are linked together with event streams to form a *management goal graph* that achieves a specific management task or goal.

B. Services Management Infrastructure

Autonomic Web Services Environment (AWSE) is an implementation of our services management infrastructure[3]. AWSE is based on the OASIS standard for Web Services Distributed Management (WSDM) [4]. WSDM provides standard communication protocols and message passing for the agents used in our model. It also employs Web Services Notification (WSN) [5] to support publish/subscribe message exchange, which we use to provide event streams.

We assume that each managed resource employs a WSDM management endpoint through which the management system can obtain performance metrics and adjust configuration parameters. The metrics provided by each managed resource are obtained through the WSDM

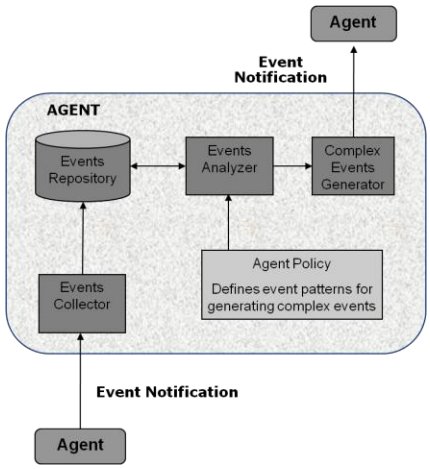

Figure 1: Agent Architecture

management endpoint via notifications that are published periodically by the endpoint to appropriate topics. An agent is defined for each metric produced by a managed resource and this agent subscribes to the topic defined by the WSDM endpoint for the metric of interest.

The management system is a collection of management goal graphs, each composed of a hierarchy of agents (see Figures 3 and 4). Each agent is constructed following the architecture shown in Figure 1. Each agent is implemented as a WSDM entity which enables the agents to communicate using standard protocols and, in particular, provides publish/subscribe messaging capabilities. The defining feature of an agent is the policy that specifies its behaviour. The policy contains the set of input streams, the output stream, a set of patterns, and for an actor, a management function that is triggered when the specified pattern is matched.

Incoming event instances are gathered by the agent's *Events Collector* and saved in the original XML format in a repository, which is implemented by a DB2 database that is unique to each agent. The events repository stores the event history. Insertion of a new event in the repository triggers the *Events Analyzer* which evaluates the new event against the set of patterns specified in the policy. A pattern consists of an event type and a context. The context in our implementation is specified using XQuery [6], a standard language for querying XML data. The XQuery returns a set of "matches" in the event history for the context. If the pattern is matched, an event instance of the appropriate type is generated by the *Events Generator* and published to the appropriate topic, notifying subscribers of the event.

Actors contain a management function as part of their policy. The management functions are implemented as stored procedures and are called using the XQuery specified in the pattern portion of the policy. A stored procedure is an external subroutine (usually written in Java or C) that is available to database applications, in our case, through the XQuery. The stored procedure is used to consolidate, compartmentalize, and externalize the logic for the actors.

Effectors impose change on the managed system. The action of an effector involves communication with the

management endpoint for the managed resource. The effector usually makes a call to adjust one or more configuration parameters or to take some management action on the managed resource via the management capabilities provided by the resource. One effector is created for each manageability capability of the managed resource, that is, each management function provided by a managed resource.

III. SUPPORTING ELASTICITY

Elastic services reside in the cloud and, either singularly or in composition, are offered to customers as *Software-as-a-Service (SaaS)*. The elasticity involves dynamically changing resource allocations to meet the current demands on the service. It is provided through requests to change the resource allocations by the management system to the underlying *Infrastructure-as-a-Service (IaaS)* layer in the cloud, for example the Amazon EC2 service [7].

Elasticity also implies the ability to dynamically change both the composition of the service and the management system in response to changes in the underlying resources. For example, increasing the number of Virtual Machines supporting the service requires another copy of the service software to run on that Virtual Machine. This in turn requires the creation of additional sensors in our Management Goal Graph and potential changes to the policies governing higher level agents in response to these changes.

A. Elastic Service Scenario

We use the following elastic service scenario throughout the remainder of the paper to illustrate our management approach. We consider a publicly available information source, say similar to Wikipedia[8], which is provided as an elastic service hosted on a cloud. The service can experience a widely varying workload of requests for information.

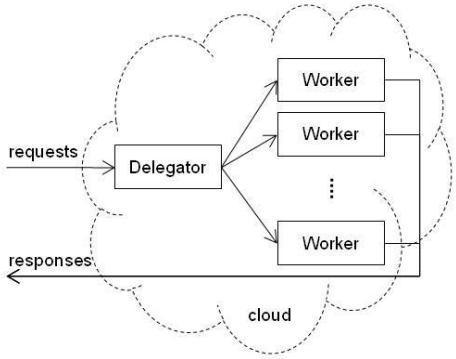

Figure 2: Example Elastic Service Scenario

The service is structured as shown in Figure 2 in order to deal with the varying workload. A single virtual machine, the Delegator, is exposed to the world as the point of access to the service. The service application itself is placed on a Worker VM image that is duplicated to accommodate the current load on the service. The Delegator acts a load balancer redirecting incoming requests across the pool of Workers. In this scenario, elasticity involves increasing and decreasing the number of Worker instances in response to changes in demand.

B. Extending the Framework

Our framework must be extended in several ways to support elasticity in the services. First, the management goal graph, and the network of agents that implements it, must be able to change dynamically to match the changing resources allocated to the service. Second, the model must provide constructs that allow the user to describe how the service should react to changes in demand. Third, the framework infrastructure must provide a mechanism to implement the previously mentioned elastic behaviour.

Figure 3 gives an overview of how we extend our existing services management framework to handle elastic services. The Management Goal Graph and Service sections of the diagram within the box outlined by a dashed line represent the existing framework. The Service, or managed system, consists of one or more managed resources. The Management Goal Graph consists of a set of agents (sensors (S), actors (A) and effectors (E)). The solid directed lines represent event streams. The dashed directed line represents an action by an effector on a managed resource.

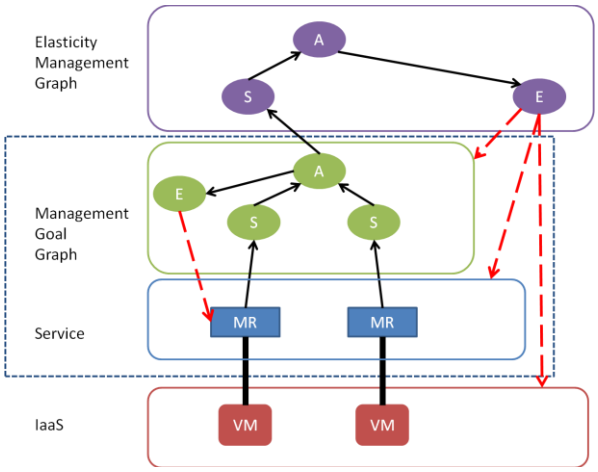

Figure 3: Framework for Management of Elastic Services

Accommodating elastic services requires the extensions represented by the top and bottom sections of Figure 3. Elastic services use the resources provided by the IaaS layer of the cloud, which is shown by the thick solid line between a managed resource (MR) of a service and a IaaS resource (Virtual Machine (VM)). The Elasticity Goal Graph is another set of agents that function in an analogous manner as a Management Goal Graph and that manage the elasticity of the service according to a user-defined policy. As shown in Figure 3, effectors in the Elasticity Goal Graph carry out changes to the Management Goal Graph, the Service, and the underlying resources allocated to the service. We now describe our extensions to the service management model and the management infrastructure in more detail.

The Elasticity Goal Graph, as mentioned above, is analogous to a Management Goal Graph. In the case of an

Elasticity Goal Graph, a Management Goal Graph is the managed resource and there are sensors to receive events from the Management Goal Graph and effectors to take actions on it. The service provider's intentions for how elasticity should be exploited by the service are captured in the policies assigned to the agents in the Elasticity Goal Graph.

Following the logic of the MAPE loop of autonomic computing [9], the sensors monitor and analyze the execution of the Management Goal Graph (and its corresponding service) by receiving events from the graph and creating inferred events for processing by other sensors or actors. The actors plan the appropriate resource allocations in response to the current performance and send the information to the effectors via events. The effectors in an Elasticity Goal Graph execute actions to perform one or more of the following:

- Modify the Management Goal Graph to accommodate changes in the underlying resource allocation. The possible modifications to the graph include adding or removing an agent, adding or removing an event stream between agents and deploying a new policy at an existing agent.
- Modify the service to accommodate changes in the underlying resource allocations.
- Issue requests to the IaaS to modify the resource allocations for the service.

The management infrastructure is service-oriented and so can be naturally extended to support the above behaviour. We encapsulate the management goal graph as a Web service like the other managed resources. The service supports a management interface that delivers events (notifications) to the sensors in the Elasticity Goal Graph and methods to perform the actions to modify the Management Goal Graph and its implementations.

We assume that the IaaS and managed services also provide a service interface for the effectors in the Elasticity Goal Graph. A managed service must provide methods to support the expansion and contraction of the service components. An IaaS typically provides a service interface to manage the allocation of resources [1].

An example of how our extended management framework could be used to manage the size of the Worker pool in our elastic service scenario is shown in Figure 4. Assuming that we start with two instances of the Worker component, the original Management Goal Graph consists of three sensors for each Worker (Response Time, Request Count and Worker Busy), a sensor to indicate if all the Workers are busy (Busy Counter), an actor to decide on actions to balance the requests among the existing workers (Balance Actor) and an effector to maintain the balance (Balance Effector).

The Elasticity Goal graph consists of the three agents at the top of the diagram: a sensor to accept events describing the request rate to the Delegator (Request Rate); an actor to determine the appropriate scaling actions for a given load (Scale Actor), namely add or remove a Worker component, and an effector to implement the required changes in the managed system and the management goal graph (Scale Effector).

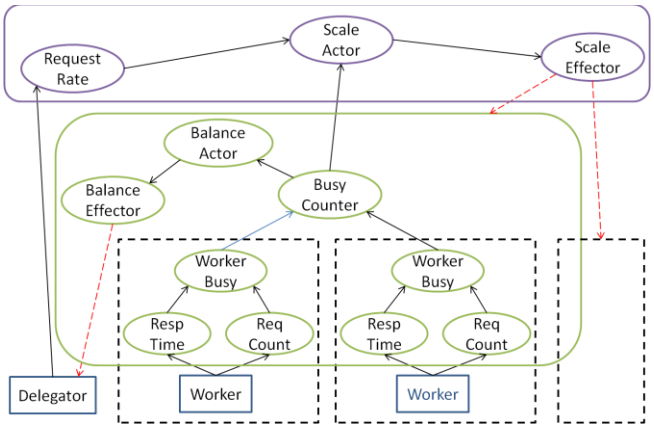

Figure 4: Management Structure for Elastic Service Scenario

On each Worker VM, the Worker produces events reporting the response time and the number of requests currently being processed. The sub-graph of sensors contained on the Worker is responsible for determining a Worker's current activity level. The activity level indicates whether or not a Worker is able to take on new requests. These events are passed to a higher level sensor that generates messages to both the scaling actor to decide if we need to add/remove Workers and to the balancing actor to decide whether or not we should adjust the workload that is going to individual workers.

The Delegator produces an event stream of request rate events. These are produced at regular intervals and report the current request rate. Overall trends of an increasing or decreasing request rate are good indicators that the Worker pool size may need to be adjusted. A sensor monitors the stream of request rate events and produces inferred events to inform the scaling actor of any trends.

The management agents can be split across running VMs with the sensors associated with a Worker placed on its VM and the remaining agents placed on the Delegator's VM. The agents can be pre-installed and configured on the image and added to the operating system's list of programs to run on boot. When a new Worker VM instance is started, its sensors automatically start up and begin monitoring and producing events. At start-up the Scale Effector can pass the management endpoint of the Busy Counter to the new Worker Busy sensor to establish the new event stream. The Balance Actor policy can also be parameterized for the number of Workers and a new value can be passed by the Scale Effector to the Web service encapsulating the management goal graph.

C. *Metrics for Managing Elasticity*

Managing elastic services in public cloud environments bring a new degree of complexity to the resource provisioning problem due to the price to be paid per usage. Amazon EC2, for example, charges per hour and instance type, each with different characteristics as shown in Table 1.

Table 1: VM Characteristics and Costs

Machine Type	Small (Default)	Large	Extra Large	High CPU Medium	High CPU XL
Cores/C.U.	1/1	2/2	4/2	2/2.5	8/2.5
Memory	1.7GB	7.5GB	15GB	1.7GB	7GB
Platform	32 bit	64 bit	64 bit	32 bit	64 bit
Price/Hr	$0.085	$0.34	$0.68	$0.17	$0.68

For this reason, metrics capturing both cost and performance are needed for making provisioning decisions on public cloud infrastructures. Previous performance studies have developed a method for deriving two combined metrics and applied the method to scientific workflows [10], [11], as well as models for certain execution profiles [12], [13].

The first metric, *Cost per Performance (C/P)*, characterizes the system on a task basis and is the result of multiplying the execution time needed for a given group of tasks using a specific infrastructure by the usage cost. The second metric, *Cost per Throughput (C/T)*, offers another way to analyze the request timeline and is based on time intervals. Additionally, both metrics offer a balanced measure of the variables involved but it is possible to change the weight of cost, performance and throughput using multipliers.

The first step in deriving the C/P metrics for an elastic service involves experimentally characterizing the performance of the main tasks on the set of VM types available. For example, if the main tasks in our scenario are *GET* and *PUT* operations, then we can conduct a series of experiments in which series of first GET, and then PUT requests are executed on an instance of each VM type. Given the performance results and the costs, we can compute the C/P values for our sample elastic service on the range of VM types.

In our elastic service scenario, the Delegator sends events to the Request Rate sensor in the elasticity management graph reporting current request rates to the service. When the request rate rises above what can be handled by the current set of Workers the Scale Actor can use the C/P and load information to determine the most appropriate type of VM to handle the additional load. A similar reverse process can be followed to decrease the number of Workers when appropriate.

D. Prototype Implementation

We are implementing a proof-of-concept prototype of our framework for elastic services management on a private cloud running Eucalyptus [14]. To implement the sample scenario described here we create two different virtual machine images: the Delegator and the Worker. The operating systems on the images are configured to automatically start all the software necessary as part of the boot sequence.

The Delegator image contains the Delegator component of the applications and the services encapsulating the management goal graph and the elasticity goal graph. The management goal graph service includes IBM DB2 9.5 [15] and Apache Tomcat 6 [16] to host the Sensors. It performs load-balancing using the Membrane Router [17], which is an open-source SOA router. The modular design of the Membrane Router allowed us to create a customized Balance Actor that uses a weighted round-robin scheduling policy.

The Worker instance uses Apache Tomcat 6 to host the managed elastic service as well as the services implementing the agents that form the Worker's sub-graph. The operating system is configured to start Tomcat once it is booted. When Tomcat starts the sensors are created and immediately begin monitoring the worker component of the application. Finally a message is sent to the Delegator to notify it that the Worker is ready to begin receiving requests.

IV. RELATED WORK

Cloud infrastructures employ virtual infrastructure (VI) management to dynamically orchestrate the deployment of virtual machines, management of storage requirements, and to configure resources to adapt to an organization's changing needs. Four of the most advanced platforms include OpenNebula [18], Enolmaly Elastic Computing Platform (ECP) [19], Eucalyptus [14] and oVirt [20].

Moreno-Vozmediano et al. [21] investigate the use of the OpenNebula VI engine to separate service management from resource provisioning. OpenNebula provides VM management, that is, resource provisioning on a local infrastructure as well as an Amazon EC2 cloud. Although this paper demonstrates the benefits of providing VI management and presents a possible architecture, it does not address the need for predictability of resource provisioning required by elastic services.

In our approach, elastic services are managed by the cloud. Lim et al. [22] believe that the cloud controller structure should leave application control up to the consumer. They explore the use of external controllers for adaptive resource provisioning. The approach assumes that the cloud platform exports a set of sensors and actors to be used by the external controllers and they focus on the challenges of building such controllers given the existing constraints of the cloud infrastructure APIs.

Boniface et al. [23] outline a Platform as a Service (PaaS) architecture that provides tools for management of service oriented applications in the cloud. We envision that our management techniques could be employed within such an environment. Chapman et al. [24] present a service definition language that we plan to investigate to define our management rules and policies.

V. SUMMARY

Cloud computing, with its support for elastic resources that are available on an on-demand, pay-as-you-go basis, is an attractive platform for hosting Web-based services that have variable demand yet consistent performance requirements. Effective service management is mandatory in order for these elastic services to be cost-effective.

Elastic services offer new challenges that cannot be adequately met by traditional management structures. In this

paper we describe a novel services management model and infrastructure for elastic services that builds on previous work in services management. The framework, which is based on autonomic computing principles and techniques, is adaptable and designed to handle the elasticity offered by cloud computing.

Our services management model represents a management task as a management goal graph, which describes the task in terms of event processing by a set of agents playing the roles of sensors, actors and effectors. Sensors accept events and produce new complex events; actors perform decision-making and planning, and effectors enforce changes on the managed system by carrying out the plans defined by the actors.

We extend our previous work on services management with the notion of an elasticity goal graph that is built from our model constructs and that manages the elasticity by coordinating and effecting changes to the cloud infrastructure, the managed service and its management goal graph. A proof-of-concept implementation of our framework is underway and is briefly described.

In continuing our work on elastic services management we plan to examine a number of research questions including the following. First, we plan to define useful elasticity metrics and methods for deriving and exploiting them. These metrics will likely be derived from the C/P and C/T metrics outlined in the paper. Second, we will look at algorithms and predictive models for effectively managing different classes of elastic services. Third, we will specifically examine elasticity for cloud data services, which is further complicated by the costs associated with managing large amounts of data in the cloud.

ACKNOWLEDGEMENTS

The research is supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada, MEDIANET (Comunidad de Madrid S2009/TIC-1468), HPCcloud (MICINN TIN2009-07146) and 4CaaSt (EU Grant Agreement 258862).

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia. "A view of cloud computing". Communications of the ACM 53 (4), 2010, pp. 50-58.

[2] P. Martin, W. Powley, I. Abdallah, J. Li, A. Brown, K. Wilson and C. Craddock. "A model for dynamic and adaptable services management". Proceedings of ICSE International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009), Vancouver BC, May 2009, pp 1 – 9.

[3] Li, J.; Martin, P.; Powley, W.; Wilson, K.; Craddock, C., "A sensor-based approach to symptom recognition for autonomic systems", Fifth International Conference on Autonomic and Autonomous Systems, 2009. ICAS '09. 20-25 April 2009, pp. 45 - 50.

[4] *WSDM v1.1*. 2008, Organization for the Advancement of Structured Information Standards; May 18 2008, http://www.oasis-open.org/specs/index.php#wsdmv1.1.

[5] *WS Notification*. 2004, Organization for the Advancement of Structured Information Standards; July 2 2008; http://www.oasis-open.org/specs/index.php#wsnv1.3.

[6] *XQuery 1*.0: An XML Query Lanaguage, W3C Recommendation, Dec 2010, http://www.w3.org/TR/xquery/.

[7] Amazon. *Amazon Elastic Compute Cloud (Amazon EC2)*. http://aws.amazon.com/ec2/.

[8] *Wikipedia http://www.wikipedia.org/*

[9] J. Kephart and D. Chess. "The vision of autonomic computing", *IEEE Computer*, 36(1), 2003, pp. 41 – 52.

[10] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman, and P. Maechling, "Scientific workflow applications on Amazon EC2", Workshop on Cloud-based Services and Applications in conjunction with 5th IEEE International Conference on e-Science (e-Science 2009), Oxford UK, December 9-11, 2009.

[11] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The Montage example", Proceedings of the 2008 ACM/IEEE Conference on Supercomputer, 2008, pp. 1-12.

[12] J. L. Vazquez-Poletti, G. Barderas, I. M. Llorente, and P. Romero, "A model for efficient onboard actualization of an instrumental cyclogram for the mars MetNet mission on a public cloud infrastructure," *Lecture Notes in Computer Science*, 2011.

[13] J.L. Vazquez-Poletti, J. Perhac, J. Ryan and A.C. Elster: "THOR: A transparent heterogeneous open resource framework", Proceedings of 2010 IEEE International Conference on Cluster Computing Workshops and Posters, Sept 2010, pp. 1-6.

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov. "The eucalyptus open-source cloud-computing system", *Proceedings of* the 9th *IEEE/ACM Int. Symposium. On Cluster Computing and the Grid (CCGRID'09)*, Shanghai, June 2009, pp. 124 – 131.

[15] IBM DB2 V9.5 online documentation. http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp.

[16] Apache Tomcat http://tomcat.apache.org/.

[17] Membrane SOA router http://www.membrane-soa.org/soap-router.htm.

[18] B. Sotomayor, R. Montero, I. Llorente and I. Foster. "Virtual infrastructure management in private and hybrid clouds", *IEEE Internet Computing*, September/October 2009, pp 14 – 22.

[19] Enomaly. Online Document: http://www.enomaly.com

[20] Ovirt. Online Document: http://www.ovirt.org

[21] R. Moreno-Vozmediano, R. Montero and I. Llorente. "Elastic management of cluster-based services in the cloud", *Proceedings of First Workshop on Automated Control for Datacenters and Clouds (ACDC09)*, Barcelona, June 2009, pp. 19 – 24.

[22] H. Lim, S. Babu, J. Chase, S. Parekh, "Automated control in cloud computing: Challenges and opportunities", *First Workshop on Automated Control for Datacenters and Clouds* (ACDC09), June, 209, pp. 13-18.

[23] M. Boniface, B. Nasser, J. Papay, S.C. Phillips, A. Servin, X. Yang, Z. Zlatev, S.V. Gogouvitis, G. Katsaros, K. Konstanteli, G. Kousiouris, A. Menychtas, D. Kyriazis. "Platform-as-a-Service architecture for real-time quality of service management in clouds". *Internet and Web Applications and Services, ICIW 2010 Fiftih International Conference*, May 2010, Barcelona.

[24] C. Chapman, W. Emmerich, F. Galan Marquez, S. Clayman, and A. Galis, "Elastic service management in computational clouds", *CloudMan 2010*, IEEE/IFIP, Osaka, Japan, 19-23 April 2010.