Policy-Based Autonomic Collaboration for Cloud Management

Omid Mola
Department of Computer Science
University of Western Ontario
London, Ontario, Canada
omola@uwo.ca

Mike Bauer
Department of Computer Science
University of Western Ontario
London, Ontario, Canada
bauer@uwo.ca

***Abstract*—The management of clouds comprised of hundreds of hosts and virtual machines present challenging problems to administrators in ensuring that performance agreements are met and that resources are efficiently utilized. Automated approaches can help in managing such environments. Autonomic managers using policy-based management can provide a useful approach to such automation. We describe different elements of a cloud system and outline how collections of collaborating autonomic managers in cloud can be a step towards better management of clouds. We also give formal definition of different elements in the managed system and show a summary of implementation results.**

Keywords-Autonomic Management; Collaboration; Policy-Based Management; Cloud Management.

I. INTRODUCTION

Cloud computing environments often depend on virtualization technology where client applications can run on separate operating virtual machines (VMs), particularly for providers of Infrastructure as a Service (IaaS). Such environments can consist of many different host computers each of which might run multiple VMs. As the number of hosts, virtual machines and client applications grow, management of the environment becomes much more complicated. The cloud provider must worry about ensuring that client service level agreements (SLA) are met, must be concerned about minimizing the hosts involved, and minimizing power consumption. Our focus is on how to better manage the virtual machine and system infrastructure of the cloud provider.

In recent years, there has been a lot of research into Autonomic Computing [1], especially about how to build autonomic elements and managers [2]. Autonomic managers try to monitor and manage resources in real time by building systems that are self-configuring, self-optimizing, self-healing and self-protecting. In the broader vision of autonomic computing, large complex systems will consist of numerous autonomic managers handling systems, applications and collections of services [3]. Some of the systems and applications will come bundled with their own autonomic managers, designed to ensure the self-properties of particular components. Other managers will be part of the general management of the computing environment. The complexity of managing a large system will entail a number of different autonomic managers which must cooperate in order to achieve the overall objectives set for the computing environment and its constituents. However, the relationships between these managers and how they cooperate introduce new challenges that need to be addressed.

We consider the use of policy-based managers in addressing this problem. Our initial focus is on a hierarchy of autonomic managers where policies are used at each level to help managers decide when and how to communicate with each other as well as using polices to provide operational requirements. The ultimate goal is to automatically monitor and manage a larger system by a collective of collaborating local autonomic managers (AMs). In such an environment, we assume that each local AM has its own set of policies and is trying to optimize the behaviour of its local elements by responding to the changes in the behaviour of those elements.

We assume some managers will also be expected to monitor multiple systems and directly or indirectly to monitor other local AMs. We also assume that one of the roles of a higher level manager is to aid other AMs when their own actions are not satisfactory.

The focus of this paper is on collaboration and communication between different managers at different levels of the hierarchy based on the active policies. The core issue addressed is how these local managers should communicate with each other and what information they have to exchange to achieve global performance goals. Finally, we will focus on how to automate the collaboration process itself.

In the rest of this paper, Section II explains the related works, Section III focuses on explaining the cloud architecture and challenges that are being addressed, Section IV gives the formal definitions of our approach towards the problem and finally, we conclude with the future works in Section V.

II. RELATED WORK

Some researchers have already begun to study how the collaboration or cooperation among local autonomic managers can be done in order to achieve a global goal.

A hierarchical communication model for autonomic managers has been used by some researchers. Famaey and

 ISBN: 978-1-61208-202-8

Latre [4] used a policy-based hierarchical model to show how it can be mapped to the physical infrastructure of an organization and how this hierarchy can dynamically change by splitting and/or combining nodes to preserve scalability. They also introduced the notion of context that needs to be accessible in the hierarchy, but do not describe in detail what this context should be and how it should be communicated. In this paper, we focus on what this context should be, how it can be transferred from one manager to the other and when this should happen.

Aldinucci, et al. [5] described a hierarchy of managers dealing with a single concern (QoS). They introduce three types of relationship between components but do not explore the details of how and when such components should interact in actual systems. They used a simulator to evaluate the framework and their main focus was on the concept of a behavioral skeleton where they used autonomic management for skeleton-based parallel programs.

Mukherjee, et al. [6] used a flat coordination of three managers working on three different parts of a system (Power Management, Job Management, Cooling Management) to prevent a data center from going to the critical state. They showed how the three managers can cooperate with each other to keep the data center temperature within a certain limit that is suitable for serving the current workload and at the same time not using more power than required. They showed how these three managers should cooperate based on different business policies.

However, these three managers are fixed and adding new managers to this system will be challenging both in terms of collaboration and scalability. The same approach as in [6] is used in [7], [8] to show the collaboration between a power and a performance manager (only two managers) to minimize the power usage as well as maximizing the performance. This method however does not seem to be generalizable to a larger environment with more autonomic managers involved because of the complexity introduced in terms of interacttions between managers.

Schaeffer-Filho, et al. [9], [10] have introduced the interaction between Self-Managed Cells (SMCs) that was used in building pervasive health care systems. They proposed Role based interactions with a Mission that needs to be accomplished during an interaction based on predefined customized interfaces for each role. This approach is very general and does not address the details of the interactions. In the work presented in this paper, we will address what the policies look like and what specific information needs to be exchanged.

Zhu, et al. [11] has introduced an integrated approach for resource management in virtualized data centres. Their approach is similar to the hierarchical approach we used in our work but the relationship between different controllers are tightly coupled whereas we suggest a loosely coupled communication style to better accommodate failures and heterogeneous autonomic managers. The focus of our work is on policies and how they affect the relationship between managers, but it's not clear how they use policies and if there is any effect on controller's communications.

III. Cloud Management Challenges

In order to describe the challenges, we first explain the cloud architecture.

A. Cloud Architecture

The infrastructure of IaaS providers, is typically composed of data centers with thousands of physical machines organized in multiple groups or clusters. Each physical machine runs several virtual machines and the resources of that server are shared among the hosted virtual machines. Therefore, there are a large number of virtual machines that are executing the applications and services of different customers with different service level requirements (via Service Level Agreement (SLA) parameters).

To have a better understanding of cloud provider environment and architecture, we take a closer look at Eucalyptus [12] (an open-source infrastructure for the implementation of cloud computing on computer clusters). In Eucalyptus, there are three main elements that form the cloud infrastructure in a hierarchical fashion:

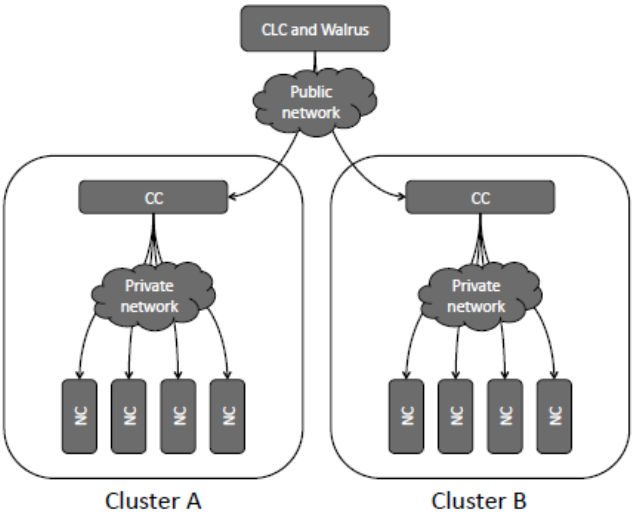

Figure 1. Eucalyptus Hierarchical Design (from [12])

- Cloud Controller (CLC): The CLC is the top level component for interacting with users and getting the requests. The CLC then talks with the Cluster Controllers (CC) and makes the top level choices for allocating new instances of virtual machines.
- Cluster Controller (CC): The CC decides which Node Controller will run the VM instance. This decision is based upon status reports which the Cluster Controller

 ISBN: 978-1-61208-202-8

receives from each of the Node Controllers. CC has three primary functions: schedule incoming instance run requests to specific NCs, control the instance virtual network overlay, and gather/report information about a set of NCs.

- Node Controller (NC): The NC runs on the physical machine responsible for running VMs and the main role of the NC is to interact with the OS and hypervisor running on the node to start, stop, deploy and destroy the VM instances. An NC makes queries to discover the nodes physical resources the number of cores, the size of memory, the available disk space as well as to learn about the state of VM instances on the node. The information thus collected is propagated up to the Cluster Controller in responses to describeResource and describeInstances requests.

B. Challenges

All of the specified elements in the cloud architecture are needed for instantiation of new images or destroying currently deployed VMs and they have some minimal management capabilities. However, the main challenges in managing the cloud environment occur after the VMs start working and receiving loads:

- How should the system respond to the load changes inside one or more virtual machines?
- What should happen to maximize the performance of a specific virtual machine (or an application inside it) according to the agreed SLA?
- How can we scale the system up and down on the fly (change VM parameters)?
- How can one enforce specific operational policies for the entire system?
- How can one make sure that minimum resources are used to perform a task (e.g. minimizing the power usage)?

A deeper look at the cloud architecture and the management needs suggest that providing all these capabilities in real time through a single centralized manager is almost impossible, because of the hierarchical layers in the architecture with different responsibilities at each layer. Also, the dynamics of load change and the need to react to these changes in real time with increasing number of VMs and physical nodes makes it much more difficult to achieve these goals with a traditional centralized manager.

Therefore, a hierarchical approach towards cloud management would be a more efficient way to achieve all of the goals. At the same time, each element in the management hierarchy should act autonomously and manage part of the hierarchy on its own.

IV. Approach and Definitions

Based on the previous discussions, we propose to use a number of different autonomic managers. By using this approach, the problem of managing a large system entails a number of autonomic managers where each one is dealing with smaller or more localized components, and then each manager's job is to focus on managing that component (or small set of components) efficiently based on certain policies.

For example, an AM for an Apache web server should only focus on the behavior of the web server and not the relationship that the webApp might have with a database server or, a Node Controller (NC) AM should only focus on the general performance and the behavior of the VMs inside that specific node.

The hierarchy of autonomic managers might appear as in Figure 2. In the lowest level, the AMs are managing the applications inside the VMs. The AMs at the node controller (NC) level monitor and manage the VMs. Then the AMs at cluster controller (CC) level are responsible for all physical nodes inside that cluster. Similarly the AM at cloud controller (CLC) level monitors and manages all of the clusters.

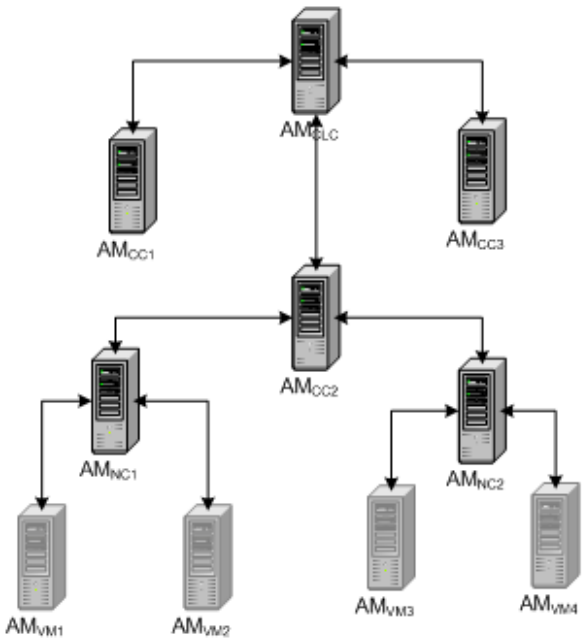

Figure 2. AMs hierarchy based on the cloud architecture

Note that this is a logical organization of autonomic managers and does not necessarily reflect the physical allocation of the AMs, i.e., they do not need to be located on different physical machines. In a large cloud computing provider they could be located on separate machines or some may be located on the same machines. These AMs should then collectively work together to preserve a set of policies for optimizing performance, minimizing resource usage, avoiding SLA violations, etc.

Assuming that the management tasks are specified in

terms of policies, this means that we need policies with different granularity deployed at different levels of the infrastructure and we need to ensure that AMs can communicate properly with each other to enforce those policies.

A. Managed System

Our managed system is composed of a set of elements that can be monitored and managed automatically. Each autonomic manager is typically monitoring and managing one or more managed elements(ME). The managed elements will be equivalent to what is found in ordinary cloud infrastructures such as a virtual machine, a physical node, a software resource, or a cluster.

We can define the charactestic and operations of each ME in a definition document called "ManagedObjectType" which can later be instantiated several times. For example, for modeling a virtual machine ME we can put all properties and metrics of a general VM in the VMManagedObjectType and later we can instantiate two objects of this type called vm1 and vm2.

Other possible managed objects types are: ApacheManagedObjectType, NodeManagedObjectType, ClusterManagedObjectType.

Definition 1. *MOt=$\langle P, M, A\rangle$ A ManagedObjectType is a tuple $\langle P, M, A\rangle$, where:*

- *P is a finite set of properties, P=$\{P_1, ..., P_k\}$,*
- *M is the finite set of metrics, M=$\{M_1, ..., M_l\}$, where: $\forall M_i \in M, M_i = \langle N_i, AC_i\rangle \mid N_i = MetricName, AC_i \in A = RefreshingAction$*
- *A is the finite set of actions, A=$\{A_1, ..., A_m\}$.*

We denote the set of managed object types by MOT=$\{MOt_1, ..., MOt_n\}$

Actions are operations that can be done on that managed object. For example, actions for a VMManagedObjectType could be startVM(), stopVM(), getVMIP(), refreshCPUUtil(), etc.

Properties of a managed object type are set upon instantiating a new managed object. Examples of properties for VMManagedObject are vmName, vmAllocatedMemory and vmOSType.

The metrics associated with a managed object are those properties that change more often and therefore must include actions specifying how they can be updated/refreshed (e.g. by connecting to another AM and sending a message to get the updated values). Examples of these metrics along with their associated actions are CPUUtil, refreshCPUUtil(), or MemoryUtil, refreshMemoryUtil(), etc.

The actions, metrics and properties defined inside managed objects types can later be used in policies to evaluate a specific condition or to perform an action on that managed object.

Therefore, based on this definition, we can now instantiate several managed objects from a single type. For example, vm1 managed object can be instantiated from VMManagedObjectType, etc.

Definition 2. *Given a set of MOT, a ManagedObject (MO) is a tuple $\langle p, m, a\rangle$ where there is a MOt=$\langle P, M, A\rangle$ such that*

- *a=A,*
- *$p = \{\langle P_1, v_1\rangle, ..., \langle P_k, v_k\rangle\} \mid P = \{P_1, ..., P_k\}$ and v_i is value of the property.*
- *$m = \{\langle N_1, V_1, AC_1\rangle, ..., \langle N_l, V_l, AC_l\rangle\} \mid M = \{\langle N_1, AC_1\rangle, ..., \langle N_l, AC_l\rangle\}$ and V_i is the measured value of a metric.*

We denote the set of managed object by MO=$\{MO_1, ..., MO_n\}$

In the rest of this document, whenever we use term managed object, we use this definition.

We assume that inside each AM there is an event handling mechanism for generating events and notifying the interested parties inside the AM. For example, there could be an event bus and different subscribers to certain events (within the AM) and upon raising those events any subscribers will get notified. This event handling mechanism is useful for handling event, condition, action policies and also for communication between managers (both explained later). We assume that for a given system and managed objects, that there are a finite number of event types.

Definition 3. *An event type, Et is a pair $\langle N, M\rangle$ where: N is the name of the event type, M=$\{m_1, ..., m_o\}$, and m_i is the name of a metric from a managed object. We denote the set of event types by ET =$\{Et_1, ..., Et_o\}$.*

Definition 4. *Given a set of ET, an event E is a pair $\langle n, m\rangle$ where there is an event type Et = $\langle N, M\rangle$, n is the name of the event n = N, m =$\{\langle m_1, v_1\rangle, ..., \langle m_o, v_o\rangle\}$, where M =$\{m_1, ..., m_o\}$, and v_i is its value. We denote the set of events by Eve =$\{E_1, ..., E_o\}$.*

For a given set of event types, there may be an infinite number of possible events, depending on the value associated with the metrics of that event type. In this respect, an event is an instantiation of an event type with the associated metrics assigned values.

B. Policies

All of the policies expressed as event, condition, action (ECA) policies. In general, all of our policies are of the form:

On event: E
if (Set of Conditions) then {
    Ordered Set of Actions
}

Upon raising an event inside the autonomic manager, then any policy which matches the event will get evaluated. If the conditions in the policy are met, then the policy actions get

 ISBN: 978-1-61208-202-8

triggered. We provide examples of policies in the following sections.

Definition 5. *A policy is a tuple $\langle N, E, C, A\rangle$ where N is the policy name, E∈ Eve is one of the events defined for the manager, C is a finite set of conditions, and A is an ordered set of actions defined in MO actions. Each condition, is defined by a tuple $\langle MName, Operator, T\rangle$, where MName is the metric name defined in a MO metrics, Operator is a relational operator and T is a constant indicating a threshold value. Therefore, Pl=$\langle N, E, C, A\rangle$, where:*

- *E∈ Eve,*
- *C=$\{C_1, ...C_p\}$ and C_i=$\langle MName, Operator, T\rangle$ or "true",*
- *A=$\{A_1, ..., A_q\}$, $\forall A_i \in A \exists MO_j \in MO \mid A_i = MO_j.AC_k$*

We denote the set of policies by PLS=$\{Pl_1, ..., Pl_r\}$.

A sample expectation policy for monitoring the Apache response time is:

$Pl_1 = \langle$ "apacheRTPolicy", ManagementIntervalEvent, apache.responseTime $>$ 500, apache.increaseMaxClients(+25, 200) $\rangle$

In this policy, ManagedIntervalEvent is an event that gets triggered in a certain time interval (e.g. 1500ms) and it has no metrics associated with it. "apache" is an instance of ApacheManagedObjectType and responseTime is one of the metrics defined in ApacheManagedObjectType. "increaseMaxClients(value, max)" is one of the actions defined in ApacheManagedObjectType and will increase the max client property of the apache web serverby a certain number up to a max (e.g., will not increase it more than 200).

At AM startup there are configuration policies that set up the AM environment, identify the appropriate managed objects and configure them. A sample configuration policy would look like:

$Pl_2 = \langle$ "StartUpConfPolicy", StartUpEvent, true, { system.setFatherIP("192.168.31.1"), system.create(vm1, VMManagedObjectType), vm1.setIP("192.168.31.3") } $\rangle$

This policy happens on AM startup and configures the parents IP of this AM in the hierarchy and also adds one ManagedObject for managing vm1 (This is happening in AM_{NC1} - see Figure 2). This AM will be responsible for managing physical node 1 which hosts vm1 and will communicate with the manager inside vm1 if necessary. The AM hierarchy can be built this way upon system startup but it can change dynamically throughout their lifetime (e.g. by migration of a VM to another machine). In this example, "system" is an instance of SystemManagedObbjectType which is useful for configuration and management of the manager itself.

C. Structural Relationship Between AMs

In order to explain the relationship between AMs in this system we first need to define the AM itself.

Definition 6. *An Autonomic Manager(AM) is a tuple $\langle MO, Eve, Pol, RI, MI\rangle$ where MO is a finite set of managed objects, Eve is a finite set of events, Pol is a finite set of policies, RI is the refresh interval which determine the time interval for updating the managed objects metrics and MI is the management interval, which determine the time interval for enforcing active policies. These two thresholds can be configured for each AM. We denote the set of AMs by AMS=$\{AM_1, ..., AM_t\}$*

Based on the cloud architecture, we assume AMs are organized in a hierarchical manner to reflect different authority levels in cloud. So, the structural relationship between AMs consists a tree.

Definition 7. *The hierarchy of AMs is a tuple $\langle AMS, Edges\rangle$ where AMS is the set of autonomic managers as the nodes of the tree and Edges=$\{(AM_i, AM_j) | AM_i, AM_j \in AMS\}$ is the set of edges connecting two AMs to each other. The following properties exist in this hierarchy:*

- $\exists AM \in AMS | \nexists AM_i \in AMS, (AM_i, AM)$
- $if (AM_i, AM_j) \in Edges \Rightarrow \nexists AM_k | (AM_k, AM_j) \in Edges$
- $if (AM_i, AM_j) \in Edges \Rightarrow (AM_j, AM_i) \notin Edges$

D. Communication Model

Each manager should be able to receive messages from other managers or send messages to other managers. In previous work [13], [14], we suggested the use a message-based type of communication between AMs. Three different types of messages (NOTIFY, UPDATE_REQ, INFO) were proposed as sufficient for communication between managers.

Since we are dealing with a hierarchy of managers then each manager needs to communicate with either its father or its children. However, it is also possible for an AM to send NOTIFY messages to another AM in some other part of the hierarchy based on a request.

The UPDATE_REQ message is sent from higher level managers to lower level ones. INFO messages are sent in response to the UPDATE_REQ message and NOTIFY messages are sent from one manager to another based on the need. In the previous work [14] we have shown how one can use policies to generate these messages for communication among AMs based on demand.

V. Conclusion and Future Works

Based on the previous discussions, we have introduced an automated collaborative approach towards management of a cloud infrastructure. So far, we have implemented the hierarchy of autonomic managers and did some experiments

 ISBN: 978-1-61208-202-8

Table I
RESULTS OF THREE SCENARIOS

Scenario	SLA Violation(%)
1: No collaboration between AMs	72
2: One-Level collaboration in the hierarchy	42
3: Two-levels collaboration in the hierarchy	24

which confirmed the importance of collaboration between AMs at different layers of the cloud. The complete results can be found in [14], but Table I shows the summary of three scenarios with respect to SLA violation rate.

The main contribution of this paper compared to our previous work is to give formal definition of the managed system and autonomic managers which lead to a better understanding of the problem and developing precise algorithms. The ultimate goal is however to design algorithms that can get the system information (e.g., events, policies and ManagedObjects) and generate the required communication messages automatically. Therefore, the collaboration between AMs will become more automated itself. In this work, we assumed that policies are defined and delivered to managers by system administrators, but as a future work we are planning to make this process more automated.

The next step would then be moving towards developing and evaluating these algorithms, enabling more efficient use of the cloud infrastructure as well as meeting SLA requirements while using fewer resources.

REFERENCES

[1] M. C. Huebscher and J. a. McCann, "A survey of autonomic computingdegrees, models, and applications," *ACM Computing Surveys*, vol. 40, no. 3, pp. 1–28, Aug. 2008.

[2] J. Kephart, "Research challenges of autonomic computing," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pp. 15–22, 2005.

[3] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[4] J. Famaey, S. Latrea, J. Strassner, and F. De Turck, "A hierarchical approach to autonomic network management," *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, pp. 225–232, 2010.

[5] M. Aldinucci, M. Danelutto, and P. Kilpatrick, "Towards hierarchical management of autonomic components: a case study," in *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*. IEEE, 2009, pp. 3–10.

[6] T. Mukherjee, A. Banerjee, G. Varsamopoulos, and S. K. Gupta, "Model-driven coordinated management of data centers," *Computer Networks*, vol. 54, no. 16, pp. 2869–2886, Nov. 2010.

[7] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, and F. R. A. C. Lefurgy, "Coordinating multiple autonomic managers to achieve speci ed power-performance tradeoffs," *in IEEE Intl. Conf. on Autonomic Computing, Jun*, pp. 145–154, 2006.

[8] M. Steinder, I. Whalley, J. E. Hanson, and J. O. Kephart, "Coordinated management of power usage and runtime performance," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*. IEEE, 2008, pp. 387–394.

[9] A. Schaeffer-Filho, E. Lupu, N. Dulay, S. L. Keoh, K. Twidle, M. Sloman, S. Heeps, S. Strowes, and J. Sventek, "Towards Supporting Interactions between Self-Managed Cells," in *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, no. Saso. IEEE, Jul. 2007, pp. 224–236.

[10] A. Schaeffer-Filho, E. Lupu, and M. Sloman, "Realising management and composition of self-managed cells in pervasive healthcare," in *Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on*. IEEE, 2009, pp. 1–8.

[11] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 Islands: an Integrated Approach To Resource Management for Virtualized Data Centers," *Cluster Computing*, vol. 12, no. 1, pp. 45–57, Nov. 2008.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 124–131, 2009.

[13] O. Mola and M. Bauer, "Collaborative policy-based autonomic management: In a hierarchical model," *Network and Service Management (CNSM), 2011 7th International Conference on*, pp. 1–5, 2011.

[14] O. Mola and Michael A. Bauer, "Towards Cloud Management by Autonomic Manager Collaboration," *Int'l J. of Communications, Network and System Sciences*, vol. 04, no. 12, pp. 790–802, 2011.

 ISBN: 978-1-61208-202-8