Collaborative Policy-Based Autonomic Management

In a Hierarchical Model

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

Michael A. Bauer
Department of Computer Science
University of Western Ontario
London, ON, Canada
bauer@csd.uwo.ca

Abstract—**In this paper we address some of the main challenges in collaboration between different autonomic managers in a policy-based management environment. We focus on three main questions: how should managers collaborate with each other, when do they need to collaborate and what information do they need to exchange in communication. In our initial investigation, we explore a hierarchical model for organizing the autonomic managers. We evaluate the approach via a prototype inspired by cloud providers' virtualized infrastructure and show how collaboration between managers in a hierarchy can improve the response time of a web server and avoid SLA violations.**

Keywords: Autonomic Management; Collaboration; Policy-Based Management; Autonomic Manager Communication

I. Introduction

In recent years there has been a lot of research into "Autonomic Computing" [1], especially about how to build autonomic elements and managers [3]. In the broader vision of autonomic computing, large complex systems will consist of numerous autonomic managers handling systems, applications and collections of services [2]. Some of the systems and applications will come bundled with their own autonomic mangers, designed to ensure the self-* properties of particular components. Other mangers will be part of the general management of the computing environment.

In this paper, we address some of the challenges in enabling collaboration between autonomic managers (AMs) and show how this collaboration can help managing the overall system more efficiently. Our focus is to understand how such managers can communicate, what should be communicated and how a collection of managers should be organized. Our focus is on policy-based managers and so we are also interested in how policies are used in this kind of environment.

The ultimate goal is to automatically monitor and manage a larger system by a collective of collaborating local AMs. In such an environment we assume that each local AM has its own set of policies and is trying to optimize the behavior of its local elements by responding to the changes in the systems or environment.

II. Related work

Some research has already undertaken studies of how the cooperation of local autonomic managers can be done in order to achieve a global goal. This work has looked at hierarchical organization of managers for cooperation, agent-based approaches and registry-based techniques [9, 12, 13, 14].

Famaey and Latre [9] 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 identified the importance of of context in the hierarchy, but do not describe in detail what this context should be and how it should be communicated. 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. [10] also describe 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.

Mukherjee, et. al. [11] 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. 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 is used in [6, 7] to show the collaboration between a power and a performance manager (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 interactions between managers.

Schaeffer-Filho, et. al. [4,5] 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 get into details of the interactions, but in this paper we will address what does the policies look like in a communication and what specific information needs to be exchanged.

III. Approach

If we assume that the problem of managing a large system entails a number of autonomic managers where each one is dealing with smaller or more localized components, then each manager's job is to focus on managing that component efficiently. In such a system, the relationship between different managers will be very important, because each of the AMs has only a local view of the system.

We assume all AMs are working according to a set of policies. An overview of policy-based management along with relevant standards and implementation techniques can be found in Boutaba, et. al. [8]; our policies are of the following form:

"if (Set of Conditions) then { Set of Actions }"

As a first step, we consider a hierarchical organization of AMs. A hierarchy provides a simple, yet useful, structure and has several advantages over a flat structure (e.g. improved scalability by reducing communication overhead that only happens between parent and child). By having a hierarchical organization of AMs then our main focus, is to answer these three questions:

A. How do AMs communicate?

If we think of AMs as separate entities then we need a mechanism (protocol) for communication between these separate entities. Just as humans need a common language to understand each other, we need a common protocol that all AMs understand. AMs could run on different platforms or even be implemented in different languages. So, the communication protocol should have the ability to operate across *heterogeneous* platforms.

AMs should not be dependent on a direct communication with each other, since they should act autonomously. Therefore, we need a way of communication that is *reliable, asynchronous and loosely coupled*. In a hierarchical architecture, each AM can only communicate with its father or its children. So, we need a *point-to-point* or *publish/subscribe model* of communication.

Point-to-point is useful when parent wants to communicate with a specific child and publish/subscribe model is useful when children want to communicate with parent in a loosely coupled way. Considering all these facts we propose a "*message-based"* means of communication between different AMs. To have this capability we need a message broker to accept messages and send them to the requested destination to make sure that AMs are loosely coupled and failure of one would not affect the other.

For small environments with a small number of managers, one message broker is probably sufficient for the entire system. But as the system grows, it may be better to have multiple brokers and domains of communication where the AMs in a particular domain only communicate with a single broker. It is also possible to have a backup broker to ensure fault tolerance, in case the primary broker is not available.

This approach helps avoid a single point of failure and also builds the notion of local loops and outer loops associated with an autonomic manager (see Figure 1).

B. When to communicate?

Communication between AMs is an extra overhead on the systems that needs to be minimized, and so, we have to avoid sending messages when there is no need for it to make sure that the communication part of the system will not turn into a bottleneck itself. Considering a hierarchical organization of AMs, each AM needs to send messages only to its parent or children. So the rest of the system is not affected and depending on how the AMs are networked together, the communication overhead of a specific manager could only affect the local network which is another reason why the hierarchical approach is useful.

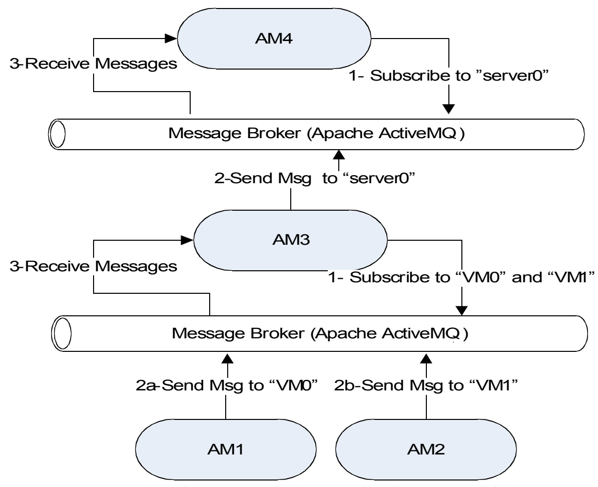
AM4
3-Receive Messages
1- Subscribe to "server0"
Message Broker (Apache ActiveMQ)
2-Send Msg to "server0"
AM3
3-Receive Messages
1- Subscribe to "VM0" and "VM1"
Message Broker (Apache ActiveMQ)
2a-Send Msg to "VM0"
2b-Send Msg to "VM1"
AM1
AM2

Figure 1. Hierarchy of message brokers to implement pub/sub and peer-to-peer model of communication.

With this in mind, as long as an AM is working well, that is ensuring that its policies are being met and without reaching any internal limits, there is no need for further communication. The exception to this is where the higher level manager needs to have more information and ask for it. Assuming that our managers are policy-based, we propose that communication should happen only in these three cases: 1) When a policy is violated, the local AM should inform the higher level manager that a violation occurred and update it with the new information (we expand on what this information is in the next section). 2) When there are no further local adjustments possible, it requires the help of a higher level manager (This can be implemented as a separate policy). 3) When a higher level manager needs to have the latest updated information about its children or the elements that they manage in order to make a decision.

C. What information to exchange?

The most important question in the AMs' relationship is what information should be exchanged between the AMs. As part of our initial investigation on this issue, we suggest that a message should have this format:

Msg = <Type, Info>

Type = HELP_REQ | INFO | UPDATE_REQ | EVENT

Info = Metrics | Event

Metrics = {<m,v> | m is the metric name, v is the metric value}

Event = <Event_Type, Event_Info>

We propose to have the following messages passed from lower level managers to the higher level managers:

M1 = <HELP_REQ, Metrics>: A message requesting help when the local manager is unable to make further adjustments in order to meet the defined policies. Local manager can optionally send the latest values of the metrics available along with the message to the higher manager.

M2 = <EVENT, Event>: When some event happens in the local manager and it has to be passed up, it can be encapsulated inside an event message. The type (*Event_Type*) and content (*Event_Info*) of the event is very system specific and can both be declared in the body of *Event* information. Possible events would be "policy violation event", "system restart event", "value update Event", etc. We will show an example of this message in the next section.

M3 = <INFO, Metrics>: A message that provides information about metric values, which can help the process of decision making in the higher level manager. This message is usually sent in response to the UPDATE_REQ message from a higher level manager (e.g. M4 explained below). The content (*Metrics*) is very dependent on the nature of the system and can be different from one system or application to another. Examples of such information include CPU utilization, memory utilization, number of requests/second, number of transactions, available buffer space, packets per second, etc.

On the other hand, higher level managers can ask their children to provide updates on specific information. To do this, they send a message asking for some information and get the INFO message back in response.

M4 = <UPDATE_REQ, Metrics>: A message asking for the status of the metrics declared in *Metrics*. This message can request for the latest metric values.

Policies play an important role in that they include what metrics should be added to the message and sent to another AM. In the next section we will discuss how to use these concepts in an actual scenario and how we can implement them in a prototype system.

IV. Prototype

To test our ideas, we take an example motivated by cloud computing providers, particularly, IaaS providers. We assume that each server is configured to operate several virtual machines (VMs) that are running separately. All VMs within a server can be monitored and managed from a privileged VM.

Figure 2 shows the layout of the physical machine and guest VMs on top of KVM virtualization [15]. The manager running inside Domain0 has the authority to change the configuration of other VMs such as allocated memory, allocated CPU cores, etc.

We implemented the autonomic manager in JAVA and used simple policy language (SPL) [16] to describe policies. We chose Java Messaging Service (JMS) [17] for our implementation and used the Apache ActiveMQ [18] as the message broker.

Each AM has its own set of policies and tries to optimize the performance of the local system. AM3 manages physical server "0", trying to optimize its performance and behavior based on the policies given to it. This includes monitoring the other VMs (VM0 and VM1) in order to help them when they are in need.

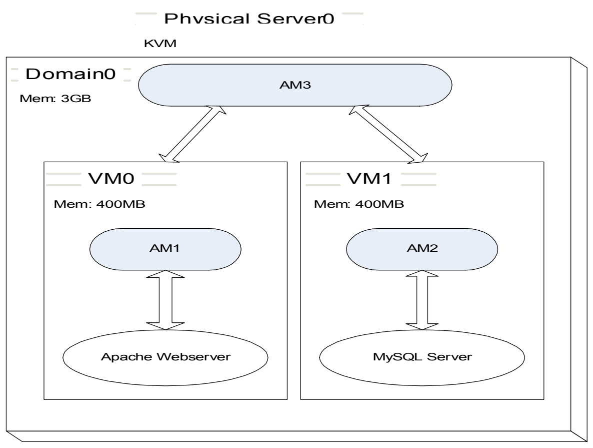

Figure 2. Layout of one physical server in the hierarchy. The server is using KVM virtualization.

A. Policies

Our work focuses on self-optimization with the use of expectation policies; an example of an expectation policy that looks after the Apache response time is like this:

```
if(apache.responseTime > 1500 & apache.maxClients <= 200)
{           apache.setMaxClient(+25);
            apache.generateSLAViolationEvent(cpuUtilization,
memoryUtilization);          }
```

The maximum adjustment that this AM can do to control the response time is to set the MaxClient parameter to 200. Each time this policy is executed it checks the Apache response time and if it is larger than 1500ms and there are enough resources available to allocate (I.e. MaxClient <= 200) then it increments MaxClient by 25. After setting the new value for MaxClient, it will generate a SLA violation event and attach the latest metric values of the local system (e.g. CPU utilization, memory utilization) to it to inform the higher level manager of the latest situation in this local.

Communication policies are just used to communicate with other managers. A sample communication policy would be::

```
if(apache.maxClients == 200 & apache.responseTime>1500){
sendHelpRequest(cpuUtilization, memoryUtilization); }
```

This policy checks to see if the AM has reached its limits and there is a critical situation (i.e. maxClients = 200 and apache.responseTime > 1500) then it will ask for help from a higher level manager by sending the latest information through a help request message.

One important part of the system is how the higher level manager cdecides what to do in a certain situation using policies (e.g. when a help request comes in). In our system a sample policy of the higher level manager (e.g. AM3) is:

```
if(vm0.memoryUtil > 80 & vm0.allocatedMemory <= 500 &
vm0.helpReq == true) { vm0.setMemory(+25); }
```

This policy belongs to AM3 which is running in Domain0 with privileged access to other virtual machines. It checks the condition of VM0 which AM1 is running on, and if its condition get critical (e.g. vm0.memoryUtil > 80) and there is enough resources to be allocated to (e.g. vm0.allocatedMemory

<= 500) and there is a help request from the lower level manager (i.e. vm0.helpReq = true) then it will respond by increasing the allocated memory of the whole virtual machine by 25 more mega bytes and continue monitoring the condition.

V. Experiments and Results

We used an open source online store called "Virtuemart" [19] running on VM0 and JMeter [20] to generate loads on this store and measured the response time of Apache in three scenarios. The ultimate goal of the whole system is to keep the response time under a certain threshold (e.g. 1500ms) that is defined in an SLA.

A. *Scenario 1: No Expectation Policy-No Collaboration*

In the first scenario we disabled all expectation policies in the autonomic managers and disabled all collaborations with higher level manager. Figure 3 shows the response time of Apache web server in this case. As expected, with increasing the load on server the response time of the Apache server gets worse and since there is no adjustment to the environment it will stay above the SLA threshold.

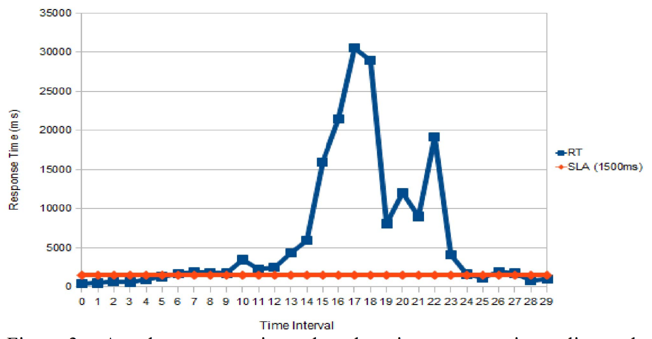

Figure 3. Apache response time when there is no expectation policy and no collaboration between managers.

B. *Scenario 2: Active Policies-No Collaboration*

In the second scenario, we have all policies in place and the local AM is doing its job, but there is no collaboration between managers. Figure 4 shows the Apache response time in this case. As the result, when the load increases the local manager try to adjust the web server by allocating more resources. For example at points A, B, C and D in the diagram an SLA violation occurred. In response, the AM increased MaxClients by 25 until at point D it reaches to the limit for MaxClient.

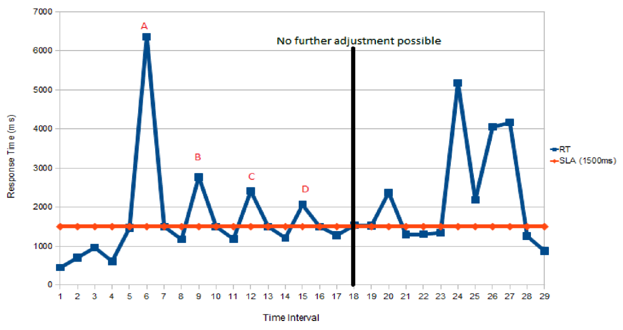

Figure 4. Apache response time when expectation policies are active but there is no collaboration between managers.

C. *Scenario3: Active Policies and Collaboration*

In the final scenario, when the local manager can no longer make adjustments to the system, it requests help from the higher level manager. This is specified in the policies of AM1 as it is mentioned in the previous section. Figure 5 shows the Apache response time in this case. Like the previous scenario, the local manager (AM1) tries to adjust the web server to handle the increasing load at points A, B, C and D.

Eventually, there are no more local adjustments possible (After D) and AM1 asks for help when the next violation happens (point E). In response, AM3 allocates more memory to VM0. At this point, the response time starts decreasing, but since the load is still high AM1 detects another violation at point F and asks for more help, and AM3 allocates 25 more megabytes to VM0.

After point F, the response time stays below the threshold. The results show that there is huge improvement in terms of avoiding violations if AMs can collaborate. In this scenario, total number of messages passed from AM1 to AM3 for collaboration is 8; 6 violation messages are sent at points A, B, C, D, E and F and 2 help requests are sent at points E and F.

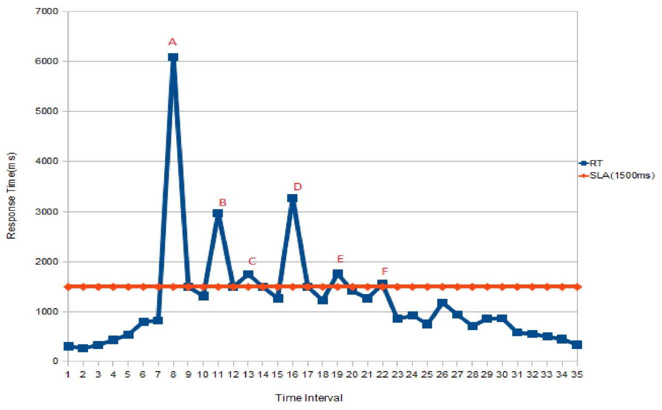

Figure 5. Apache response time when expectation policies are active and there is collaboration between managers.

VI. Conclusion

In this paper we described some of the primary steps towards the details of autonomic manager collaboration. As an initial step, we proposed the hierarchical model for organizing managers and suggested the use of a messaging technique for communication between managers.

We showed how we can use policies as a way to facilitate collaboration among managers and what information was necessary to be exchanged in these relationships. We evaluated these ideas in a prototype and showed how this collaboration can be useful to preserve the response time of a web server under a certain threshold (defined in SLA). For future work, we plan to investigate how this communication can be inferred automatically from policies and how can policies decompose from higher level manager to lower level ones.

References

[1] M.C. Huebscher and J. a McCann, “A survey of autonomic computing—degrees, models, and applications,” *ACM Computing Surveys*, vol. 40, Aug. 2008, pp. 1-28.

[2] J.O. Kephart and D.M. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, Jan. 2003, pp. 41-50.

[3] J.O. Kephart, “Research challenges of autonomic computing,” *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005, pp. 15-22.

[4] A. Schaeffer-Filho et al., “Towards Supporting Interactions between Self-Managed Cells,” in First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007), 2007, no. Saso, pp. 224-236.

[5] 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, 2009, pp. 1–8.

[6] 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*, 2006, pp. 145-154.

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

[8] R. Boutaba and I. Aib, “Policy-based Management: A Historical Perspective,” *Journal of Network and Systems Management*, vol. 15, Nov. 2007, pp. 447-480.

[9] 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*, 2010, pp. 225-232.

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

[11] T. Mukherjee, A. Banerjee, G. Varsamopoulos, and S.K.S. Gupta, “Model-driven coordinated management of data centers,” *Computer Networks*, vol. 54, Nov. 2010, pp. 2869-2886.

[12] G. Tesauro, D.M. Chess, W.E. Walsh, R. Das, A. Segal, I. Whalley, J.O. Kephart, and S.R. White, “A multi-agent systems approach to autonomic computing,” *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, IEEE Computer Society, 2004, p. 464–471.

[13] W. Chainbi, H. Mezni, and K. Ghedira, “An Autonomic Computing Architecture for Self-* Web Services,” *Springer.*, vol. 23, 2010, pp. 252-267.

[14] M. Jarrett and R. Seviora, “Constructing an Autonomic Computing Infrastructure Using Cougaar,” *Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE’06)*, 2006, pp. 119-128.

[15] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux virtual machine monitor,” *Proceedings of the Linux Symposium*, 2007, p. 225–230.

[16] D. Agrawal, S. Calo, K.-W. Lee, and J. Lobo, “Issues in Designing a Policy Language for Distributed Management of IT Infrastructures,” *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, IEEE, 2007, pp. 30-39.

[17] M. Richards, R. Monson-Haefel and D. A. Chappell, Java Message Service (Second Edition), O’Reilly Media Inc., Sebastopol, California, 2009.

[18] Apache ActiveMQ, Available at http://activemq.apache.org/

[19] Virtuemart, Available at http://virtuemart.net/

[20] JMeter Load Generator, Available at http://jakarta.apache.org/jmeter/