Improving the Management of Distributed Applications in Cloud Environments

Chrysostomos Antoniadis
Computer Science Department
University of Crete
chranton@csd.uoc.gr

ABSTRACT
This paper presents my findings and the results of my undergraduate thesis, which focused on improving the management of distributed applications in cloud environments with the use of Nagios Core system [1].

1. INTRODUCTION
The size of cloud environments is continuously growing and the distributed applications are getting more and more complex. So it is vital to have an effective way to manage these systems 24x7. Our target is to automate the management procedures and reduce the staff needed for these operations. This has been achieved by using two factors. The Nagios Core monitoring system which provides hardware, software and network monitoring, and the Smart Automated Management application (SAM). SAM uses the results taken from the Nagios Core monitoring procedure, analyzes the data, makes decisions about the system's health and performs the needed actions through its own management sequence. The main role of SAM application is to detect and identify problems and possible threats to the system, take the necessary messures to solve them and keep the system operational.

It must be pointed out that Microsoft has its own automatic management software called Autopilot [2]. We will attempt a comparison between the two systems later on this paper.

For the matters of this thesis an assumptions has been made. The distributed web server we use is very small but what we do can be also done to bigger clusters.

The paper is structured as follows: In section 2 the functionality of Nagios Core system is briefly described. In section 3 we present and describe the cloud infrastructure. In section 4 we are going to describe in detail SAM. In section 5 we compare SAM with Microsoft's Autopilot. Finally, in section 6 our conclusions are presented.

2. NAGIOS MONITORING SYSTEM

2.1 Nagios Core
Nagios Core is an open source computer system monitor, network monitoring and infrastructure monitoring software application. Nagios Core offers monitoring and and the NRPE daemon runs the appropriate Nagios plugin alerting for servers, printers, routers, switches, applications and services. It alerts users when things go wrong and alerts them again when the problem has been resolved. It provides, among others, monitoring of network services such as HTTP, SMTP, FTP and SSH, monitoring of host resources such as processor load and disk usage, and generally monitoring of anything that has the ability to send collected data via a network to specifically written plugins. Remote monitoring is provided via remotely-run scripts via Nagios Remote Plugin Executor (NRPE) [3].

The hosts and the services that Nagios Core monitors, are specified by the user. The data from the various checks is stored in a text file rather than a database. This file is named status.dat and it is the one that SAM uses to collect the data it needs as described in section 4.

By default Nagios Core assumes that all hosts and services that it monitors are in OK state. After a check is performed there are four state possibilities, OK, WARNING, UNKNOWN and CRITICAL. The checks are executed periodically and every check returns some data that accompanies the state. For example the HTTP check returns the HTTP response code, the response time in seconds and etc. In some services checks, the user can define the values for the OK, WARNING and CRITICAL state. This gives the flexibility to Nagios Core to operate in various hardware scales, different systems, and also gives the freedom to the user to adopt the checks in his own targets.

2.2 NRPE
Nagios Remote Plugin Executor (NRPE) is a Nagios agent that allows remote system monitoring, using scripts that are hosted on the remote Linux/Unix systems. This is useful when there is the need to monitor local resources/attributes like disk usage, cpu load, memory usage, etc. on a remote host. The NRPE addon consists of two pieces, the check_nrpe plugin, which resides on the local monitoring machine and the NRPE daemon, which runs on the remote machine.

When Nagios Core needs to monitor a resource of service from a remote Linux/Unix machine, it executes the check_nrpe plugin and informs it which service needs to be checked. The check_nrpe plugin contacts the NRPE daemon on the remote host over an SSL-protected connection

to check the service or resource. The results from the service check are passed from the NRPE daemon back to the check_nrpe plugin, which then returns the check results to the Nagios Core process.

3. THE SETUP

The distributed application that is used in this thesis is a Web Server running on four virtual machines. Each virtual machine runs Crunchbang Linux [4], a lightweight Linux distribution and emulates a server with a single cpu processor running on 1GHz, and 512 MB of memory.

3.1 Cloud Infrastructure

SAM's cloud infrastructure consist of four virtual machines each running Crunchbang Linux (See full setup on Table 1). The virtual machines are running on Windows 7 on VMWare Player [5], connected on a local network created by the player. On the first virtual machine Nagios Core and Apache Tomcat [6] have been installed, and on the other three virtual machines Apache server [7] and NRPE daemon have been installed as well. The three virtual machines will be referred as real servers or hosts or simple servers.

VM Name	VM IP	Nagios Segment	Service
Crunchbang-1	192.168.109.131	Nagios Core	NAT
Crunchbang-2	192.168.109.132	NRPE	Web Server
Crunchbang-3	192.168.109.133	NRPE	Web Server
Crunchbang-4	192.168.109.134	NRPE	Web Server

Table 1.

Nagios Core on Crunchbang-1 uses check_nrpe plugin to monitor the resources and services on the remote servers. NRPE daemon in each server sends the results back to check_nrpe plugin where they are stored in status.dat file as it has been mentioned before. The resources monitored in every host are: CPU load, RAM usage, HTTP and TCP services. Later on is described how the results are used by SAM system to manage the web server.

3.2 Distributed Web Server

Crunchbang-2, 3 and 4 are the main parts of the web server while on Crunchbang-1 a NAT service is responsible to forward the external http requests to one of the real servers (see Table 1).

NAT service has been implemented by a java servlet [8]. Every external request to the web server is redirected by the servlet to one of the available real servers. It reads the available hosts IPs from a file every five minutes and selects one of them in a simple round robin format.

The time period the servlet uses to read the file is not restricted and can be adjusted to the web server needs! This also applies to the round robin format, as any appropriate algorithm can be used.

After the redirection of the request, the selected real server communicates directly with the client, until the end of the session with no further interference by the NAT service servlet.

4. SMART AUTOMATED MANAGEMENT

SAM is the tool for improving the management of distributed applications in cloud environments. It is an application which is responsible to analyze the data taken from Nagios, make decisions about the health of the system and perform the needed actions in order to maintain the web server operational, despite the problems that may occur and threaten the system.

SAM's first job is to read the data returned to Nagios by the check_nrpe plugin through the status.dat file. From this file it reads, beside the results, how many computers are part of the cloud infrastructure (three in our example), their names, their IPs, and finally which service corresponds to which host. In general every time that SAM reads the status.dat file, it also checks if a new real server has been added to the infrastructure. The new server must be registered to Nagios Core first.

After the initialization the first check of the system is performed. A real server is considered healthy if the host state and each of the service states that correspond to the host have OK value. In the beginning every real server is tagged as non healthy and non part of the web server. There are two boolean variables in each host to keep this information. After the check, every host who is healthy is tagged as available to be part of the web server.

Thereafter SAM stores in a file, named info.txt, the names and the IPs of the real servers that are available to be part of the web server. Now the NAT service servlet can read this list of the servers and redirect any http request to one of them.

Then SAM waits for a five minute period. This period is not restricted and it was chosen given the fact that Nagios is set to perform a new check every three minutes. It is in user's freedom to set it in any value that fits his infrastructure.

When the waiting time period has passed SAM reads again status.dat file as before and performs a new health check. Now there are four possibilities. First a host who was previously healthy is still healthy, second an unhealthy host is still unhealthy and third a host who was previously unhealthy is now healthy (apparently the problem he had has been solved), and SAM tags the server as available to the web server. The fourth and more interesting scenario is when a host was healthy at the previous check but now shows some sort of a problem. This means that the host and/or one or more of the services have CRITICAL, WARNING or UNKOWN status.

In this case the real server is not immediately tagged as unhealthy but SAM acts as follows. SAM keeps a counter for every host in the cluster. This counter increases by one for every problem that occurs at each check. It also has a max value. When the counter reaches half of it max value the host is tagged as in critical face. This means that something is going wrong and there is a serious threat

that may cause further problems to the web servers functionality. First SAM determines which service has the problem. If the CPU or the RAM load checks give the faulting response, probably the workload is to high and there are more HTTP requests than the host can handle. In this case SAM adds one more real server to the web server in order to have a better workload distribution. In other cases SAM removes the server from the web server replaces it with another available server and monitors its behavior at the next checks. If the problem insists then SAM attempts to restart the services. After this happens if any of the problems still exist the counter continues to increase and when it reaches its max value, SAM is confident for a problem in the server and remotely reboots it.

In general SAM adds to the web server one host more than the detected threads. In the beginning there are zero threads so only one real server is becoming part of the web server. Obviously in the present system the number of real servers is too small. Usually in distributed applications the amount of the servers is significantly higher.

After every action to deal with a problematic situation, if the threat disappears (the host and all of the services go back to OK state), SAM resets the counter, removes the critical tag, makes the real server available to the web server and a new monitoring cycle begins.

One more case that SAM is trying to predict is when a host is serving a very small number of requests and most of its capabilities are unused and could be eventually used somewhere else. For this reason there is one more check script in Nagios Core which checks if the CPU load is very low. If a WARNING state is returned repeatedly in a number of checks, SAM removes the host from the web server. Obviously there is at least one host part of the web server besides the one SAM removes.

There is always the possibility that a real server is faulty and SAM can not do anything to repair it. For example the network path to the host has been interrupted and the host is unreachable. Another example is in larger scale infrastructures where a group of real servers communicate with SAM through a router and this router is faulty and the whole group has been cut of from the network. In these cases human intervention is required in order to repair the problem. For these reasons SAM, after a number of checks and actions where the initial problem remains unchanged, alerts the administrator through a log file where it records and specifies the problem.

When a check begins to a real server the condition of the server is checked first. It must be noted that SAM always checks the condition to the services even if the host is not in an OK status. This makes SAM act faster because the counter mentioned before increases faster. SAM is also constructed in such a way that in larger infrastructures where the system is composed from smaller systems, it only checks a host if its path is in a healthy condition. This way it avoids unnecessary work. This is the reason why in our web server SAM also monitors Crunchbang-1.

5. COMPARISON WITH MS AUTOPILOT

Autopilot is Microsoft's management software. It is responsible for automating software provisioning and deployment, system monitoring, and carrying out repair actions to deal with faulty software and hardware.

Autopilot's core is consisted by the Device Manager, a Watchdog service, a Deployment service, a Provision service and a Repair service. There are also the Collection service and the Cockpit, a visualization tool.

The Device Manager is a single strongly-consistent state machine that stores the state that the system should be in, without taking any repair actions himself. It uses a number of satelite services with which it exchanges information in order to keep the system consistent. The services report the problem to the Device Manager without trying a repair action. Autopilot then performs cosistensy checks and if there is a problem it tells to repair services to take action. SAM has the responsibility to detect a problem or a threat and repair it as soon as posible. For repair actions it does not use any other application than himself.

The Provision and the Deployment services ensure that each computer is running the correct operating system image and set of application processes. This happens through the supply of a local service and a set of manifests in every computer, in order to specify that the correct services are installed and running. The manifests are used for service configuration. In our system the administrator is responsible to install, configure and ensure that the right operating system and services are running in each computer. During the SAM's management cycle if the needed processes and services are not present in a computer then the machine is not accepted to the cluster and it is considered as in failure.

The Watchdog service detects software or hardware failures and reports them to the Device Manager. The watchdogs perform periodic checks in order to detect problems or possible threats and report back to the Device Manager via the watchdog protocol. Our system checks are performed by the NRPE plugin. The NRPE scripts are the respective watchdogs and they report to Nagios Core via a simple text file.

The Repair service cooperates with the Device Manager to recover from software or hardware failures. It periodically asks the Devices Managers for computers in failure and uses their management consoles to perform their requiret repair action. SAM is resposible to detect a posible threat ignites a repair thread for each machine in failure. We must point out here that the unit of failure in Autopilot is a computer or a network switch rather than a process. There are no application-specific recovery actions. On the other hand SAM's unit of failure is an application or a computer but this is not absolut. It was chosen this way because of the small web server size and for the purpouses of this thesis.

The Collection Service and Cockpit passively gather information about the running components and make it available in real-time for monitoring the health of the service as well as recording statistics for offline analysis.

The Cockpit specifically is a visualization tool that les operators monitor one or more Autopilot clusters using graphs and reports. Nagios Core uses his own graphic environment that shows detailed informations for every machine monitored. There are also graphs to show the relationship between the system's entities.

Autopilot's components use a simple pull model to retrieve information from the Device manager were the components of our system exchange information throught simple text files.

6. CONCLUSION

We have created a simple management tool in order to automate monitoring, fault detection and repair procedures in distributed applications in cloud environments. We use the Nagios Core system with its NRPE plugin in order to monitor a distributed web server. We have also created Smart Automated Management, a software that uses the results from Nagios Core in order to manage actively manage the web server. Finally we compare our system with Microsoft's Autopilot. Autopilot is a commercial, advanced and sophisticated system but there are structure and functional similarities with our system in key components of the management procedure.

7. ACKNOWLEDGMENTS

I would like to thank professor Konstantinos Magoutis, for giving me the opportunity to complete my bachelor degree with this thesis, for his guidance and assistance that helped me complete my project.

I would also like to thank Philip for providing his computer for testing purposes.

A debt of gratitude is owed to Chrysi. Without her nothing would be possible.

8. REFERENCES

[1] Nagios Core http://www.nagios.org/projects/nagioscore

[2] Michael Isard Autopilot: Automatic Data Center Management, April 2007.

[3] Nagios Remote Plugin Executor http://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details

[4] Crunchbang Linux http://crunchbang.org/

[5] VMware Player http://www.vmware.com/products/player

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

[7] Apache Server http://httpd.apache.org/

[8] Jeremy Elson and Jon Howell Handling Flash Crowds from your Garage, USENIX Annual Technical Conference, 2008.