2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing

Autonomic Management of Cloud Neighborhoods through Pulse Monitoring

Trevor Lorimer
IBM United Kingdom Limited
Belfast
Northern Ireland
trevor.lorimer@uk.ibm.com

Roy Sterritt
Computer Science Research Institute
University of Ulster
Northern Ireland
r.sterritt@ulster.ac.uk

***Abstract*—This paper reports on autonomic computing research, including the development of a self-* proof of concept, for a cloud based environment. It monitors administrative boundaries from within an autonomic manager, with each manager operating in a peer-to-peer mode and utilizing a pulse monitor. The prototype was developed in Java utilizing SNMP to demonstrate the manager's self-situation and environment-awareness of the current state of the whole neighborhood and proves the feasibility of communicating the health of the neighborhood to peer managers using an XML pulse concept. Each manager houses the functionality to enact changes to their neighborhood using SNMP based rules. This enables the capability to provide self-healing, self-configuring, self-optimizing and self-protection to network neighborhoods within cloud computing.**

Keywords; Autonomic Computing; Cloud Computing; Pulse Monitoring

I. INTRODUCTION

Autonomic computing as a concept, although still relatively recent, has been researched and applied in conjunction to other areas that include networking, personal computing and grid frameworks [19][23]. The emergence of cloud computing with its unpredictable demands and complexity due to managing large amounts of resources, has led to a requirement of rapid automatic adapting to changing circumstances in the cloud with minimal human intervention [14]. Cloud computing is a fairly new paradigm and seems to be the current industry focus with the main issue being the lack of defined standards, although this has started to be addressed through the input of standards bodies being driven by the large vendor enterprises. Yet to achieve the expected level of heterogeneous automatic adaptive behavior, Cloud Computing really requires self-management such as the common four properties; self-configuration, self-healing, self-optimization and self-protection; of Autonomic Computing. The research challenge of creating a self-managing adaptive Cloud is referred to as the *Autonomic Cloud* with the vision to implement important decisions regarding resource allocation and optimization making the system more robust, adaptable and easier to manage with minimal human intervention [3]. In some ways this research challenge may be seen as the evolution of what was started under Autonomic Grid [23] and Autonomic Business Grid initiatives [24].

The autonomic initiative itself in 2001 had an initial long-term vision of 20+ years; the vision could be interpreted as *extensively localized cooperative peer-to-peer self-management*. The practicalities of implementing a revolution in an evolutionary fashion resulted in most cases sticking with client-server style self-management (watchdog or sentinel servers) since putting the self-* in and taking the human out is such a difficult task in itself. There are concerns that Cloud Computing even with Autonomicity could go the same way. To achieve the truly flexible self-adaptive, self-managing next generation Cloud will require that *extensively localized cooperative peer-to-peer self-management.* This research study aims to investigate the relevant research efforts that have been carried out within comms, personal and grid computing towards achieving this and determine through a proof of concept its applicability to Cloud Computing.

II. AUTONOMIC CLOUD & RELATED WORK

The large data centers of cloud computing providers must be managed in an efficient way. In this sense, the concepts of autonomic computing inspire software technologies for data centre automation, which may perform tasks such as: management of service levels of running applications; management of data centre capacity; proactive disaster recovery; and automation of VM provisioning [20].

Cloud computing is a model for providing on-demand access to a wide variety of software applications, and to configurable computing resources that can be deployed to support whatever applications a user chooses to run. On-demand access and self-provisioning are key characteristics of cloud computing, and are one of the principal features that distinguish cloud computing from other types of hosted service.

Fundamental to the cloud model from the service provider's viewpoint is the pooling of resources to serve many customers. This will mean the use of virtualized computing and storage resources, and multi-tenant support for applications. This is key to achieving efficient use of resources and a low cost base, which means lower costs for end users. The use of virtualized platforms and multi-tenancy also enables another important aspect of cloud computing – the idea of computing resources being scalable or elastic, so that users can take as much or as little as they require and change their requirements at any time. It is from the aspect of flexibility that the authors' believe Autonomic techniques of self-configuration and self-optimization can be applied to Cloud computing [6].

978-0-7695-4862-3/12 $26.00 © 2012 IEEE
DOI 10.1109/UCC.2012.60

A. Administrative Boundaries

Within the area of resource management for cloud computing the network nodes are provisioned statically, yet cloud resources need the ability to be provisioned dynamically on demand. There is also a requirement for network resources to be provisioned away from application resources, as is common in an enterprise environment there are several administrative domains present on divided silos and may operate in isolation. Due to the nature of autonomous clouds, administrative boundaries are formed as a collection of *neighborhood* nodes with the administrative boundary limit being the number of nodes comprising the neighborhood [4].

With an autonomic cloud instance created on demand and forming inherent administrative boundaries with a quick turn around time, collaboration between administrative boundaries becomes much harder, this has the potential to void an original benefit of grid and cloud computing paradigms, that being collaborative resource sharing.

Autonomic cloud isolation principles with regards to managing the different layers of an IaaS infrastructure have been investigated, where devices on the physical layer are grouped together with firewalls filtering desired packets to the region [21].

B. Pulse Monitoring

Pulse Monitoring (PBM) is an evolution from the fault tolerant mechanism of a heartbeat monitor (HBM) where an "I am alive" signal is generated periodically to inform the system that a component is still functioning and when no signal is returned the system realizes there is a fault or problem. Independent process monitoring through HBM allows problem determination to become a proactive rather than reactive endeavor. The concept of a beacon tone lends itself to PBM where the tone of the heartbeat varies to indicate the level of severity of the problem which can be used to prioritize action on the most poorly component if several are in difficulty and also gives greater advanced warning on a potential fault. PBM is a hybrid of HBM and NASA beacon monitor where instead of just checking for the presence of a beat the rate of the beat is measured as well [17][19].

In an Autonomic Element (AE) which is made up of a managed component and Autonomic Manager (AM) the component is actively self-monitored for its current state and that of its external environment, with the information gathered being analyzed with the system knowledge base and adjustments made to the managed component if necessary. The PBM can summarize the state of the managed component and transmit this to all other connected AE's so they become self-aware of the health of the managed component or the external environment as seen by that AM, if there is no signal then there must be an issue with the AM. As the beat of a heart has a double beat (lub-dub) the AE's PBM can have a double beat encoded with a self health/urgency measure and an environment health/urgency measure this maps to the self-awareness and environment awareness properties in the AE.

To provide an element of system knowledge a database is stored on each node and stores the pulse levels and knowledge (which may adapt over time), monitoring logs and the history of neighbor nodes. With a biological system the reflex reaction is wired-in through genetic adaptation over time, the autonomic PBM system requires an ability to pre-configure the reflex reaction with examples being "pulse sending interval" and "change pulse after three failed process" properties.

C. Policy Based Compliance Management

IBM Tivoli provides an application called Policy Based Compliance Management (PBCM) which is a network compliance application and is a component of their Network Automation Solutions [8]. The method of keeping networks devices in compliance involves defining policies which continually manage and validate device configurations and intelligently resolve noncompliant conditions when they are recognized. The complete network compliance life cycle is automated to define policies, validate devices against policies, resolve violations, and report results in a continuous and closed-loop [5].

D. Dynamic Routing with DHT

Distributed Hash Table's (DHT) are adaptive to topology changes and are scalable and self-organizing which compliments the principles of autonomic computing and cloud computing. DHT however assumes the devices are homogeneous but in a large scale network the devices are most likely to be heterogeneous. This has been countered by organizing devices into groups that allows routing control in a DHT [22]. The grouping together of heterogeneous nodes mentioned before allows flexible routing control to choose new routing paths when regions are detected as being under stress.

E. Cloud Bursting

Cloud Bursting integrates a private cloud with a public cloud. Autonomic techniques are employed monitoring network conditions and calibrating the cloud-burst engine's network activities [10]. An Autonomic Cloud virtualized architecture proposed by Pujolle [13] uses agents distributed throughout the network, monitoring their own local neighborhood of devices and cooperating to characterize any problems more precisely which are then reported to the management center. Casalicchio [1] identified functional requirements of an autonomic service provisioning system and found that currently services such as auto-scaling and advanced monitoring are not usually offered by public IaaS (Infrastructure as a Service) providers. So there is a need for the components responsible for the Monitor, Analyze and Plan phases of the autonomic cycle to be implemented independently.

III. DESIGN

The purpose behind the tool is that each node within an administrative boundary contributes its health to the neighborhood autonomic element which uses the collective device input to monitor the health of the region, with many

health indicators available on each device that can be correlated and acted upon as necessary. The heart-beat monitoring of the system is one of the component parts to the tool and one basic indicator of a nodes health is whether or not the device is up and functioning. This can be achieved through pinging and this will return either true or false, if multiple devices are recognized as down by the neighborhood autonomic manager, this will signify poor rudimentary health of the affected administrative boundary and instantly alerts to a current issue.

Most devices also incorporate detailed system and traffic information that will indicate an overborne or stressed node, which points to a region that is experiencing a heavy workload and an autonomic manager is alerted in advance to an immanent danger, this would enable a proactive approach to maintaining network health as active measures can then be taken and creates further potential for autonomic techniques to be employed. The communications channel that enacts the required changes will be the same as the technique used to gather the device information in the first place and will be implemented using SNMP.

The tool operates in a peer-to-peer mode so each manager has responsibility to monitor its neighbors so an autonomic manager is connected to a peer autonomic manager monitoring a separate region by registering with it, because of the peer-to-peer architecture all managers have the same capabilities and responsibilities. When an issue is encountered in a region the corresponding manager will try and fix locally and if the issue persists the manager will notify its registered peers by sending a change of pulse via an external monitor. This pulse information is generated from an internal monitor that correlates the health of the regions hosts and the pulse will take the form of a message that will be interpreted by the main monitor within the receiving peer manager and can then act appropriately by using predefined knowledge rules. The rules that determine a change of pulse will be adaptable and can be re-configured over time. The pulse sent between peers is represented in an XML message and transmitted using TCP as the messages may become too large to be accommodated by UDP as they were in previous pulse monitoring applications [17].

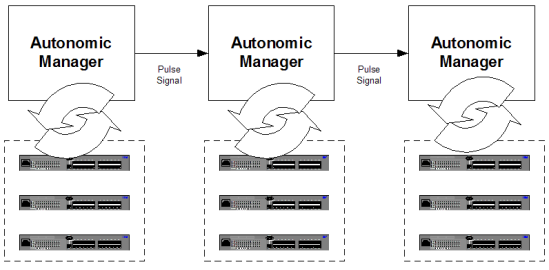

Figure 1. The Autonomic Pulse Monitoring Tool Environment

Information on manager properties, peer-to-peer relationships, knowledge rules and logging will be kept on a central database, but this is only used as a central repository for use by the GUI to display an operational representation of the tool. In a peer-to-peer system there needs to be movement away from a central store of the whole systems data, so all managers download the properties, relationships and rules upon initialization, and keep these in memory throughout their operation, though rules and relationships can be downloaded from the GUI database as a reasonable way to distribute new updates to the locally stored data.

A. SNMP

SNMP which stands for Simple Network Management Protocol was designed by the Internet Engineering Task Force (IETF) in the early 80s as a set of protocols for managing complex networks. It is implemented by sending messages called Protocol Data Units (PDUs) to specified devices in a network. An SNMP compliant device is called an agent and they store information about their configuration and set-up in an arrangement specified by Management Information Bases (MIBs), this data can be returned to the SNMP requester that transmitted the PDU.

The management data for the device is exposed by SNMP using variables called Object Identifiers (OIDs) defined within the MIBs on the managed device [11].

This is a fundamental usage of SNMP and reverberates well with the needs of the tool being developed as essentially SNMP exposes management data that signifies traffic flows and congestion across the device being monitored along with physical and logical data that can help with inventory management of a device. An example being if an interface card is removed without authorization this seriously affects the function and performance of the device and needs to be escalated, an autonomic element will not be able to heal the physical loss but logically it may be possible to recreate the function of that card on any card not in use on the device.

SNMP operates in the application layer at layer 7 of the OSI model and allows active management tasks to be performed, such as modifying and applying a new configuration by modifying OIDs on the managed device, the OIDs are organised in the MIBs using hierarchies that include metadata such as data type (integer/alphanumeric) and description [12]. The SNMP agent receives requests on UDP port 161, the agent response is then sent back on the source port to the manager.

Two of the core PDUs intended to be utilised are:

- GetRequest – A manager-to-agent request that retrieves a value of an OID or list of OIDs which are specified in variable bindings. The response is returned in an atomic operation by the agent. This will be how the tool will gather information during device monitoring.
- SetRequest – Is similar in operation to the GetRequest, instead to change the value of an OID or list of OIDs, the variable bindings need to have the correct data value bound to the OID being updated. This technique will allow the tool to update the device as an actuator to any healing or optimisation [11].

Some issues are connected with SNMP one is that the installations can vary across devices, and the operations available may not perfectly match the blueprint laid down in the MIBs which can make data validation temperamental.

Also processing SNMP queries on certain data sets (large routing tables) may require higher than normal CPU utilisation because of SNMPs structure not being well understood by a platforms internal data structures.

An alternate to using SNMP to monitor and respond to managed device issues would be to upload the entire device CLI as a text file and parse the entire configuration for the relevant information, make the change to the CLI and push the text file back down to the device. The benefits of this would be increased power to implement changes to the device, but the added complexity and the fact there is no common configuration layout between device manufacturers means SNMP is ideal as it is simple and is common on almost all network devices.

B. XML Messages

With prior PBM implementations and research for inter-process communication between autonomic managers, messages have been sent between managers using UDP [17], this is fine for small messages relating to a single device but in the monitoring of a region of devices the messages need to be larger as more content is required to capture the overall state of the neighborhood if multiple devices are experiencing issues. A message in regards to the tool is an idiom for representing information packaged for distribution between autonomic managers, to implement a message that can vary in size and support hierarchical information then XML is a good choice to represent this. XML can provide a standard, agreed upon format for data transfer between applications running on separate Java processes. However one issue is that XML conversion can be very resource intensive with large data sets, other possible options considered where Java RMI and SOAP. SOAP (Simple Object Access Protocol) was dissuaded for use in the tool as it requires a web server and has a verbose XML format that adds to complexity. Java RMI (Remote Method Invocation) was a viable option but the easy layout, simplicity and flexibility of sending XML messages over a socket was the most attractive.

For the tool each manager will be able to connect to multiple managers to broadcast their pulse message, with UDP multicast is an intrinsic part of the protocol, however to implement this functionality using TCP each subscribing manager's IP address will have an accompanying port. With TCP it is a benefit for the tool for messages to arrive in order as the health of the transmitting manager can conceivably change between messages being received by the subscribing manager, which could result in the subscribing manager actuating a non-required remedy that would undo the work already carried out by the transmitting manager.

The format of the XML message will contain the overall status of the neighborhood being monitored and one or more issues that are of interest to the peer group of managers of the transmitting manager. There are three elements with the issue element consisting of three sub elements.

Host – The IP address of the transmitting manager.

Status – The overall health of the manager neighborhood.

Issue – Consists of 3 elements:

Ip – The IP address of the device pertinent to the issue.

Oid – The affected SNMP OID that was not compliant with the predetermined rules supplied to the manager.

Fail – The nature of the non-compliance of the OID.

```
<message>
    <host>9.180.191.77</host>
    <status>0</status>
    <issue>
        <ip>172.25.0.12</ip>
        <oid>1.3.6.1.2.1.6.0</oid>
        <fail>101</fail>
    </issue>
</message>
```

Figure 2. Example XML message

IV. IMPLEMENTATION

A. Initial Setup

The class from where most of the tool behavior is controlled is AutonomicManager, its main purpose is to act as a controller to the monitoring threads and setup data from the database into local memory, this object is instantiated on a server by using a batch startup batch file to call the main method of AutonomicManager. The main method is passed a properties file which contains the database connection properties and the hostname for the AutonomicManager instance. The startup() method of the class then connects to and interrogates the database for all data pertinent to the autonomic manger instance and stores in local memory, this is the only call to the database that is of critical use, all subsequent database interaction is for demonstration purposes linked with the GUI. The database connection uses standard JDBC calls that are encapsulated within the DatabaseAccess class which handles connecting and disconnecting to database, all database queries, updates and inserts, and converts all data retrieved into modeled classes for each database table.

There are three model class views Manager, ManagedDevice and OidRule that map more of less directly to the database tables of the same name, and have separate private methods for retrieving and building the data, this technique helps enable the use of Java generics which moves the responsibility of type safety to the compiler. So there is no need to write code to test the correct data type because it is enforced at compile time, and iterating through a list of generic types using a for loop becomes very clean and simple as there are no casts or primitive-to-object conversions. In addition to type safety generic collection types generally provide better performance for storing and manipulating value types as there is no need to box the value types.

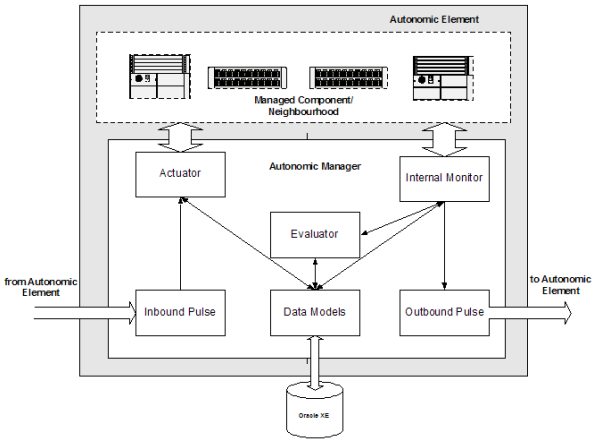

Figure 3. System Architecture

When the local data models are populated the two monitoring threads for external and internal monitoring are started using the implements Runnable interface technique. The AutonomicManager instance is passed to both threads to share the populated model information which requires the accessor methods to have the synchronized keyword to avoid concurrency issues. This applies most relevantly to the logger method. Local memory models can be updated from the database by calling the refresh() method in AutonomicManager this can be performed periodically and uses a refresh flag to indicate the local data has been changed. The shutdown() method closes down the AutonomicManager in a graceful way by finishing up work started by threads and is initiated by the user via a batch file.

B. Monitoring

The monitoring of the neighborhood involves two phases with the first phase being where each device in the neighborhood is pinged to check that it is up. Pinging a small number of devices is not very challenging, but if there are many devices this may cause problems. If the polling interval is too short a problem can emerge where the nth poll hasn't finished before the nth+1 poll begins [11]. Once a device has been confirmed as up the gathering of pertinent network data can be performed through SNMP

With regards to monitoring the region to collect system performance metrics [1] there are two possible techniques to obtain management data stored on a device through SNMP these are by using an SNMP GetRequest or SNMP Trap.

SNMP provides the ability to send traps, or notifications, to advise an administrator when one or more conditions have been met. SNMP traps are alerts generated when a condition has been met, by agents on a managed device, the alerts may contain data that is statistical or status related. These conditions are defined in the MIB provided by the device vendor. The network administrator defines thresholds, or limits to the conditions, that are to generate a trap. Conditions range from preset thresholds to a restart. After the condition has been met the SNMP agent then forms an SNMP packet that is returned to the manager used to monitor the network [2].

Though the use of SNMP traps is beneficial and the alert is generated to the manager as soon as a predefined condition is met, this requires setup on the managed device that would add additional setup and extra domain knowledge for the tool. To bypass those issues the same benefits can be achieved by using an SNMP GetResponse with a list of pertinent OIDs to return the value, this makes the tool device independent and flexible with no additional setup steps. The OIDs relating to a particular device are stored in an OidRule class that contains, the OID to be retrieved, the Rule or condition that the value on the device must comply with, and whether to apply the resolution to by the manager that discovered the issue or send it to its neighbor manager to enact the resolution. The OidRule class also contains a FailureCategory field that specifies the resolution that should be applied on failure of the condition.

The SnmpComms class connects to the device and performs the SNMP GetRequest this is facilitated by the third party software SNMP4J [15] which provides classes and interfaces for creating sending and receiving SNMP messages, it uses a SNMP security that is SNMPv1, SNMPv2c and SNMPv3 compliant though the tool only supports SNMPv1 and SNMPv2c through the building up of a SnmpCredentialSet that is uniquely stored for each device. Results of the GetRequest are returned in a Map, these results are used by the SNMPEvaluator class which evaluates the management data returned with the rules stored in the OidRule objects for each device.

As each OID value is evaluated when an incompliant value is detected it can either be addressed locally or can be broadcast to the managers subscribing neighbors, this means the issue will be added to the pulse message that is transmitted from the manager. The issue is added to an XML document is built up using JDOM where all issues are collated under the manager hostname. The tool uses JDOM as an alternative to DOM (Document Object Model) which is an API that allows programs to store an entire XML document in memory and access it like a tree structure [9]. JDOM only requires one third of the equivalent DOM code to perform the same function and is much cleaner. Another option for parsing XML would have been SAX (Simple API for XML) which is platform independent and is efficient at parsing very large XML files, as the XML messages used should not be extremely large JDOM was chosen as the preferred method for XML manipulation.

The monitoring process from checking the devices availability through to evaluating the SNMP GetRequset is repeated for each managed device in the manager's neighborhood. The JDOM document is then converted to a String and is ready to be broadcast across the socket from the internal monitor to all subscribing peer managers.

The subscribing autonomic manager has an inbound monitor that sits waiting for messages from the transmitting managers by using the traditional client-sever technique of running inside a separate thread from the main thread of execution. When a message is received the concept of an actuator [16] updating the monitored entity is enacted using

SNMPActuator and called to process the message string by cleaning it up and converting it again using JDOM into an XML document. The XML document is then turned into an object using the AutonomicMessage class that makes it simpler in Java to process the issues delivered in the message. When the message object is created, the message receipt and transmitting managers' status is logged, then the issues are examined and using the FailureCategory resolutions are attempted to be applied if the resolution string in prefixed with #set, otherwise a string message is logged.

Within the SNMPActuator object the SnmpComms class has the capability to issue SNMP SetRequest's to resolve specific issues, setting SNMP OIDs in the environment of the tool is best suited to issues where the OID result does not match the expected value in the OIDRule and need to be set back to a specified value. When making an SNMP SetRequest a pair of values is required in the PDU, the OID that is to be changed, and the value to change it to. In the following format:

#set oid_to_update oid_type new_oid_value

The OID needs to be read-write which can be found out from the OID access value in the MIB, also the type of value the OID accepts is required, it is important to send the correct type of value to the agent because if the wrong type of value is sent to perform the Set operation the agent will return a WrongType error in the response PDU.

C. GUI

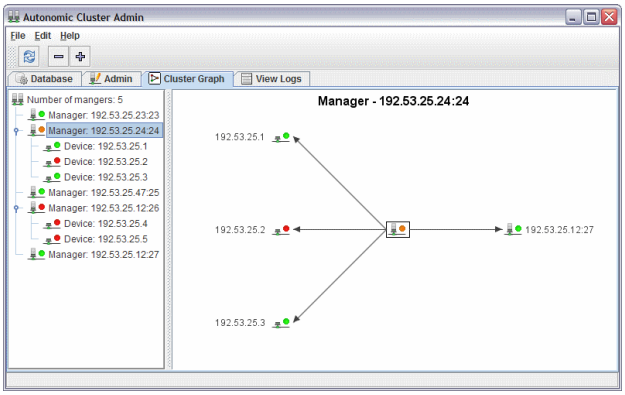

Figure 4. The Autonomic Cluster Graph Visualization

The GUI for the tool is developed using JAVA Swing libraries and the user is able to administer and view the health of the autonomic system through a visual and colour coded environment. The various Autonomic Manager inter-relationship linkages are displayed in a diagram on the first presentation screen. Each node contains the given name for the AM region and the colours of Green, Amber and Red to generally indicate the health of the AM and its region. Green represents a fully functioning region, amber is a state where one or more devices are in difficulty and red means all devices are not responding or a critical issue has been detected. The colour coding is also extended to the tree view down the left hand side.

The GUI is launched through the ACAdmn which extends JFrame, this is a key class and links together various models, views and controllers conforming to the MVC design methodology. Each of the tables in the database populates its contents into a data model which is used within in the GUI to display the relationships between managers and the devices monitored by the managers.

When an autonomic manager in the tree view is selected an additional visual representation of the linkage between the manager and the devices it is responsible for are displayed in a cluster graph, the relationship between other manager's is also presented visually again in a cluster graph. It is on this screen that devices under an autonomic manager's administrative boundary can be added or removed. In addition to the cluster graphs there is a tab present for each of the tables in the database which represent an interface for quickly adding data into the system, using a built in widget.

The GUI is affected by data that is periodically updated into the central monitoring database (Oracle XE) from the information held in memory on each autonomic manager. The data transfer is bi-directional with the GUI and autonomic managers needing to re-sync periodically.

V. EVALUATIONS

A. Self-Healing Scenario

This evaluation follows a scenario where an autonomic manager detects an anomaly from an SNMP GetRequest on one of the managed devices within its administrative boundary, the rule it breaks changes the health of the AM to amber and the fix is required to be administered by a neighbouring AM. This is highlighted in the XML message received by the neighbouring AM and enacted upon but using an SNMP SetRequest.

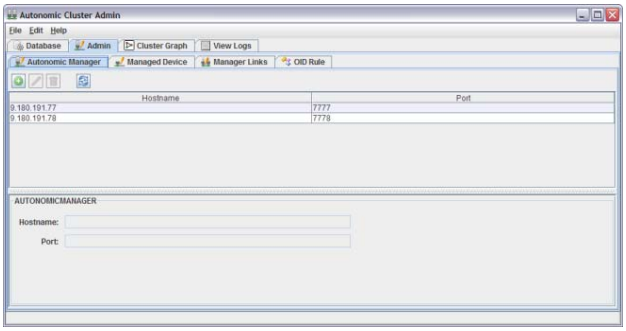

Figure 5. Updating the Database using the Admin Tool

The test system consists of two managers (9.180.191.77 & 9.180.191.78) with two devices (172.25.0.12 & 172.25.0.15) in a region assigned to 9.180.191.77.

The device 172.25.0.12 has an OIDRule assigned to it in relation to sysLocation that it must be equal to "Belfast" however in this instance it happens to be "London". So in the

GUI the device shows up red and the manger is colored amber.

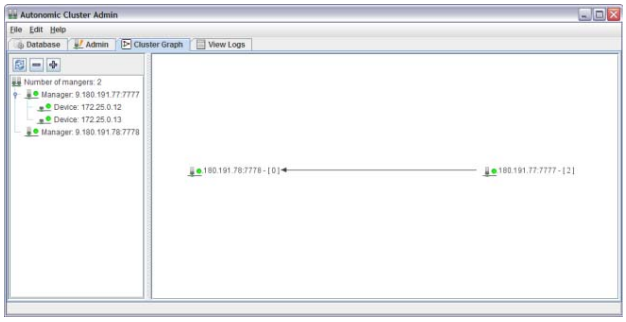

Figure 6. The Manager relationship cluster graph

As the local flag is set to false in the OIDRule therefore the execution of the resolution must be carried out on another autonomic manager other than its neighborhood manager. An xml message is sent to 9.180.191.78 containing the device affected and fail category.

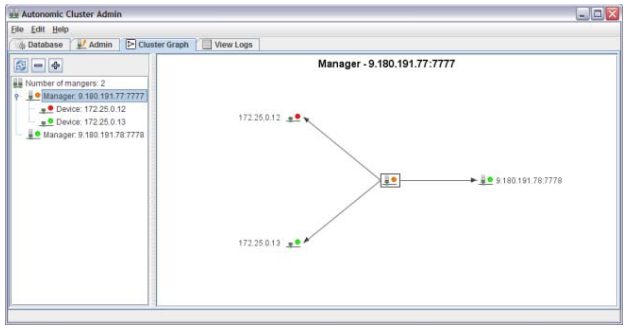

Figure 7. Visualisation of anomaly found on 172.25.0.12

When 9.180.191.78 receives the message it uses an SNMP SetRequest to update the sysLocation on 172.25.0.12 using the FailResolution command in the FailResolution object. It takes the format OID, OID type (in this case OctetString) and value to update.

B. Performance Evaluations

The performance evaluation of the system focused on the implementation surrounding the SNMP scalability relating to the GetRequests and SetRequests. For these tests six different models of device were employed to demonstrate if there is any differentiation in the speeds displayed for the same action performed on the exact same OIDs for each device. The six device VTMOS are displayed in TABLE I. .

GetRequests

The devices were all located in the IBM Titanic Quarter Belfast Lab and were located within a similar distance to the manager server running on RedHat Enterprise Linux version 5.0 administering the tests.

TABLE I. DEVICES USED IN PERFORMANCE TEST

Device #	VTMOS			
	Vendor	***Type***	***Model***	***OS***
10	Cisco	Router	3825	12.4(19b)
11	Cisco	Router	7200	12.3(12a)
12	Cisco	Router	3640	12.3(6a)
13	Cisco	Router	2600	12.1(2)T
14	Cisco	Router	2500	12.2(28)
18	Juniper	Router	M5	JUNOS 10.1

TABLE II. RESULTS FOR GETREQUEST PERFORMANCE TEST

Device Type	SNMP GetRequests (Milliseconds)				
	1	***2***	***5***	***10***	***20***
Cisco 3800	2	3	5	12	22
Cisco 7200	1	2	5	13	22
Cisco 3600	7	12	31	62	124
Cisco 2600	6	12	30	60	118
Cisco 2500	25	49	124	250	497
Juniper M5	2	4	10	19	34

TABLE III. RESULTS FROM SETREQUEST PERFORMANCE TEST

Device Type	SNMP SetRequests (Milliseconds)	
	1	***2***
Cisco 3800	4	5
Cisco 7200	4	6
Cisco 3600	16	30
Cisco 2600	8	15
Cisco 2500	30	60
Juniper M5	4	7

The tests took the form of loading the PDU with OIDs present in the IF-MIB which were ifIndex, ifMtu, ifSpeed, ifInOctets, ifInUcastPkts, ifInNUcastPkts, ifInDiscards, ifInErrors, ifInUnknownProtos, ifOutOctets these are OIDs associated with traffic monitoring and are all numeric and common to each device. There were two values for each OID relating to two different interfaces, bringing the total possible number of OIDs to 20. The experiment involved running a GetRequest for a PDU of OIDs numbering in increments of 1, 2, 5, 10, 20, and for each of these runs an average time was recorded by iterating five times over each increment. The results are shown in TABLE II.

SetRequests

The tests used the same six devices from the GetRequest evaluation and the experiment followed a similar structure only this time the PDU applied an SNMP Set to sysContact and sysLocation which are both strings. Only two OIDs where used during this evaluation because the devices were the property of IBM and could not afford to have the device placed into an unusable state, the updating of sysContact and sysLocation are actions that are somewhat safe-TABLE III.

VI. CONCLUSIONS

This research involved developing a prototype that incorporated peer-to-peer autonomic Pulse Monitoring (PBM) and applied to a cloud based system. PBM has been demonstrated within Personal Autonomic Computing (PAC) [18] and Grid Computing [23] but the relevance to cloud computing had not been sufficiently explored. The system involved moving the Autonomic Manager away from the device on the physical layer and placed it on a virtualized server to monitor several devices in a neighborhood similar to what Pujolle [13] described. Each manager collates and sends the health of its monitored neighborhood to its subscribed neighboring autonomic manager in a message i.e. pulse, using a peer-to-peer structure. Aspects of Policy Based Management were used to evaluate the state of the system against predefined rules and actuate self-healing to return the system to the desired state. The focus of the evaluation was to investigate the capability of the system to recognize a fault and react to it in a positive manner by altering the pulse message in the appropriate way. The approach to evaluating the prototype was achieved using real devices in the IBM lab. Relevant performance metrics were gathered to illustrate the overheads that may come into consideration with regards to scalability. The results of the investigation highlight what features need to be improved or added for a fuller scale version of the prototype, yet demonstrate the promise of extensively localized cooperative peer-to-peer self-management for the Cloud.

ACKNOWLEDGEMENTS

IBM, Tivoli, and Netcool are trademarks of International Business Machines Corporation. Oracle, Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

REFERENCES

[1] Casalicchio, E.; Silvestri, L.;, "Architectures for autonomic service management in cloud-based systems," 2011 IEEE Sym Computers and Communications (ISCC), pp.161-166, Jun 28 -Jul 1

[2] Cisco, Understanding Simple Network Management Protocol (SNMP) Traps [Online] [Accessed 26 April 2012] Available at http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a0080094aa5.shtml

[3] Endo, P.T.; Sadok, D.; Kelner, J.;"Autonomic Cloud Computing: Giving Intelligence to Simpleton Nodes," *2011 IEEE 3rd Int. Conf. Cloud Comp. Technology and Science,* pp.502-505, Nov. 29-Dec. 1

[4] Erdil, D.C.; "Dependable Autonomic Cloud Computing with Information Proxies," *2011 IEEE Int. Sym Parallel and Dist. Processing Workshops and Phd Forum,* pp.1518-1524, 16-20 May.

[5] ETSI ES 282 003: Telecoms and Internet converged Services and Protocols for Advanced Networking (TISPAN); Resource and Admission Control Subsystem (RACS); Functional Architecture, June 2006.

[6] Hall, Peter. "What is Cloud Computing?". The Role for Telcos in Cloud Computing. Ovum Summit. 2010.

[7] International Business Machines Corp., An architectural blueprint for autonomic computing, White Paper Fourth Edition, 2006.

[8] IBM, IBM Tivoli Netcool Configuration Manager - Compliance System Admin Guide [Online] [Accessed 26 April 2012] Available at http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_configurationmgr.doc_6.1.0/ncm_pdf_comp_sysadm_61.pdf

[9] JDOM [Online] Available at http://www.jdom.org/index.html [Accessed 26 April 2012]

[10] Kailasam, S.; Gnanasambandam, N.; Dharanipragada, J.; Sharma, N.; , "Optimizing Service Level Agreements for Autonomic Cloud Bursting Schedulers," 39th Int. Conf. Parallel Processing Workshops (ICPPW), 2010 pp.285-294, 13-16 Sept. 2010

[11] Mauro, D; Schmit, K (2005). *Essential SNMP* (2nd Ed). O'Reilly.

[12] Perkins, D; McGinnis, E (1997). Understanding SNMP MIBs. NJ: Prentice Hall.

[13] Pujolle, G. "An autonomic virtualized architecture for clouds and sky," 2010 IEEE GLOBECOM Workshops. pp.1644-1647, 6-10 Dec.

[14] Russell, D. M.; Maglio, P. P.; Dordick, R.; Neti, C.; , "Dealing with ghosts: Managing the user experience of autonomic computing," IBM Systems Journal , vol.42, no.1, pp.177-188, 2003

[15] SNMP4J, SNMP4J -The Object Oriented SNMP API for Java Managers and Agents [Online] Available at http://www.snmp4j.org/ [Accessed 26 April 2012]

[16] Solomon, B.; Ionescu, D.; Litoiu, M.; Iszlai, G.; , "Designing autonomic management systems for cloud computing," *Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on* , vol., no., pp.631-636, 27-29 May 2010

[17] Sterritt, R.; Chung, S.; , "Personal autonomic computing self-healing tool," *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the* , vol., no., pp. 513- 520, 24-27 May 2004, doi: 10.1109/ECBS.2004.1316741.

[18] Sterritt, R.; Bantz, D.F.; , "PAC-MEN: personal autonomic computing monitoring environment," *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on* , vol., no., pp. 737- 741, 30 Aug.-3 Sept. 2004, doi: 10.1109/DEXA.2004.1333562

[19] Sterritt, R.; Bantz, D.F.; , "Personal autonomic computing reflex reactions and self-healing," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* , vol.36, no.3, pp.304-314, May 2006, doi: 10.1109/TSMCC.2006.871592

[20] R. Tolksdorf and D. Glaubitz, Coordinating web-based systems with documents in xmlspaces, in Proc 6th IFCIS Int. Conf. Cooperative Information Systems, Springer-Verlag, 2001, pp. 356–370.

[21] Wailly, A.; Lacoste, M.; Debar, H.; , "Towards Multi-Layer Autonomic Isolation of Cloud Computing and Networking Resources," *Network and Information Systems Security (SAR-SSI), 2011 Conference on* , vol., no., pp.1-9, 18-21 May 2011

[22] Yiming Zhang; Lei Chen; Xicheng Lu; Dongsheng Li; , "Enabling routing control in a DHT," *IEEE J. Selected Areas in Communications,* vol.28, no.1, pp.28-38, January 2010

[23] Sterritt, R.; , "Pulse monitoring: extending the health-check for the autonomic grid," Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on , vol., no., pp. 433-440, 21-24 Aug. 2003, doi: 10.1109/INDIN.2003.1300375

[24] Franke, Carsten; Theilmann, Wolfgang; Zhang, Yi; Sterritt, Roy; , "Towards the Autonomic Business Grid," Engineering of Autonomic and Autonomous Systems, 2007. EASe '07. Fourth IEEE International Workshop on , vol., no., pp.107-112, 26-29 March 2007, doi: 10.1109/EASE.2007.27