Static Weighted Load-balancing for XML-based Network Management using JPVM

Mohammed H. Sqalli and Shaik Sirajuddin

Computer Engineering Department,
King Fahd University of Petroleum & Minerals,
Dhahran, 31261, KSA
E-mail: {sqalli, siraj}@ccse.kfupm.edu.sa

Abstract. SNMP-based network management is simple but lacks scalability and efficiency of processing the management data as the number of agents increases. XML-based network management is a new paradigm developed to overcome these limitations. One of the main challenges is how to distribute the management tasks to achieve efficiency and scalability. In this paper, we propose a framework using JPVM to distribute the management tasks among multiple gateways. We compare the performance of three approaches, namely the static weighted load balancing approach, the equal work non-weighted load balancing approach, and the single gateway approach. The first approach provides better communication time between the XML-based manager and the SNMP agents. It takes advantage of the XML, DOM, and Java servlets.

Introduction

The main goal of network management systems (NMS) is to ensure the quality of the services that networked elements provide. To achieve this, network managers must monitor, control, and secure the computing assets connected to the network. The Simple Network Management Protocol (SNMP) is currently the most widely used protocol for network management. SNMP is based on a centralized approach and confronted with two main limitations that are scalability and efficiency. A number of approaches have been proposed to overcome these limitations, including XML-based Network Management (XNM). One of the issues for an XNM system is to be able to support legacy SNMP agents, since they constitute the largest base of network management systems.

XML-based network management applies Extensible Markup Language (XML) technologies to network management. In XNM, the management information is defined using XML and the management data is exchanged in the form of an XML document and processed using the standard methods available for XML [1][2][3].

XML-based integrated network management architecture consists of an XML-based manager (XBM), an SNMP/XML gateway and SNMP agents [2]. In [4], we proposed a framework for extensions to an existing XML-based network management system, which can reduce the response time between the XBM and the SNMP agents. The extensions consist of new types of messages, including the multi-get-request and multi-set-request. These new types, for instance, allow a manager to send one or more requests to one or more agents bundled in one message. This framework decreases the overall traffic between the XBM and the XML/SNMP gateway.

In this paper, we present a new DOM-based approach to the proposed extended XNM, namely a static weighted load balancing approach that makes use of JPVM in XNM. We compare results obtained to the single gateway approaches and to the equal work non-weighted load balancing approach. The comparison of these approaches shows that the static weighted load balancing approach outperforms all the others and provides a savings in term of response time as the number of agents in the network increases.

The rest of the paper is organized as follows; first we will give a general overview of the XML-based network management. Then, we will discuss the current related work. We will then introduce the JPVM environment and describe the static weighted load balancing and the equal work non-weighted load balancing approaches with JPVM. The section that follows will include the experimental setup and results of comparing these approaches. The paper ends with a conclusion.

XML-based Network management

Extensible Markup Language (XML) is a Meta markup language, which was standardized by the World Wide Web Consortium (W3C) for document exchange in 1998[5]. We can define our own Structure of Management Information in a flexible form using either Document Type Definition (DTD) or XML Schema [6][7][8]. XML documents can be transmitted on the Internet using HTTP. XML offers many free APIs for accessing and manipulating the XML data. XML separates the contents of a document and the expression methods, i.e., the management data is stored in XML documents and the presentation or format of the management data is stored in Extensible Style Sheet Language (XSL) documents using Extensible Style Sheet Transformations (XSLT) representation. XML supports the exchange of management data over all the hardware and software that supports HTTP. XML needs low development cost, since all the APIs and development kits are freely available.

Fig. 1. shows one of the manager and agent combinations in XML-based network management [2]. It shows the approach that requires a translation from XML to SNMP through a gateway [1][2]. Since most network devices have legacy SNMP agents installed in them, this combination is simpler to implement in the current network environment, and is more appropriate for the current network management framework. In this paper, we only address this combination and we consider non-legacy network elements providing native XML interfaces outside the scope of this work. This combination, however, requires the development of an SNMP/XML gateway to exchange the messages between the XML-based network manager and SNMP agents.

XML-based network management can overcome many limitations of SNMP. For instance, an SNMP request can not exceed a maximum message length limit, but XML supports the transfer of large amount of data in a single document. This allows the transfer of multiple SNMP requests bundled in one message from the manager to the gateway. This message can also be summarized to decrease the amount of traffic to be exchanged between the manager and the gateway. This will result in less traffic at the manager side. The gateway will then expand the message received from the manager into multiple SNMP requests to be sent to multiple agents. With the use of multiple gateways, the processing time of multiple SNMP requests can also be reduced. All these advantages make XML a good candidate to solve the problems of scalability and efficiency of existing SNMP based NMS.

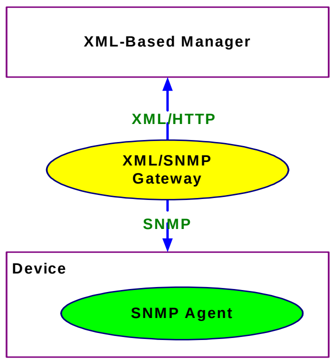

Fig. 1. An XML-based Network Management Architecture

Related Work

J.P.Martin-Flatin [3] proposed using XML for network management in his research work on web-based integrated network management architecture (WIMA). He proposed two SNMP-MIB-to-XML translation models. WIMA provides a way to exchange management information between a manager and an agent through HTTP. HTTP messages are structured with a Multipurpose Internet Mail Extensions (MIME) multipart. Each MIME part can be an XML document, a binary file, BER-encoded SNMP data, etc. By separating the communication and information models, WIMA allows management applications to transfer SNMP, common information model (CIM), or other management data. A WIMA-based research prototype, implemented push-based network management using Java technology.

F. Strauss [9] developed a library called “libsmi”, which can be used to access SMI MIB information. It can even translate SNMP MIB to other languages, like JAVA, C, XML, etc. This library has tools to check, analyze, dump, convert, and compare MIB definitions. The tool used for this is called “smidump”.

Network devices developed by Juniper Networks are equipped with the JUNOS Operating system, which supports JUNOScript [10]. The JUNOSciprt allows the client applications to connect to the Juniper network devices and exchange messages as XML document. The request and response are represented as DTDs and XML Schemas. The communication between the client and network devices is through RPC requests. An XML-based RPC consists of a request and the corresponding response. It is transmitted through a connection-oriented session using any transport protocols like SSH, TELNET, SSL or a serial console connection. Juniper Networks has already implemented a tool for mapping SNMP SMI information modules to the XML Schema. This tool is an extension of a previously implemented tool for converting SNMP SMI to CORBA-IDL. Currently Juniper Networks is working on the implementation of an XML document adapter for SNMP MIB modules using Net-SNMP and XML-RPC libraries.

Jens Muller implemented an SNMP/XML gateway as Java Servlet that allows fetching of XML documents on the fly through HTTP. MIB portions can be addressed through XPath expressions encoded in the URLs to be retrieved. The gateway works as follows: when a MIB module to be dumped is passed to mibdump, an SNMP session is initiated, and then sequences of SNMP GetNext operations are issued to retrieve all objects of the MIB from the agent. Mibdump collects the retrieved data and the contents of this data are dumped in the form of an appropriate XML document with respect to the predefined XML Schema.

Today’s network is equipped with legacy SNMP based agents, and it is difficult to manage legacy SNMP agents through an XML-based manager. Conversion of the XML-based request to an SNMP-based request through an XML/SNMP gateway provides the interaction between the XML-based manager and SNMP-based agents. For a validation of the algorithm, POSTECH implemented an XML-based SNMP MIB browser using this SNMP MIB to XML translator. This gateway is developed by POSTECH at their DPNM laboratory [1][2]. This gateway provides modules to manage networks equipped with SNMP agents [1]. The implementation of the gateway requires two types of translations: specification translations and interaction translations. The specification translation is concerned about the translation of the SNMP MIB to XML. POSTECH uses an automatic translation algorithm for SNMP MIB to XML. The interaction translation methods for XML/SNMP gateway are the process level interaction translation, the message level interaction translation, and the protocol level interaction translation.

In a previous paper [4], we proposed to extend the work of POSTECH & Juniper Networks. The framework proposed allows a manager to send requests to multiple agents using a single message. We defined new types of messages that could be sent by a manager, namely multi-get-request, multi-set-request, and response. These messages can be widely used in configuration management. The implementation for both multi-get-request and multi-set-request can be achieved through an HTTP-based interaction method and a SOAP-based interaction method. We described how a manager can send in one message either one request to multiple agents, multiple requests to one agent, or multiple requests to multiple agents. For the multi-set-request message, if an abnormal condition or an error occurs, some agents may not set the values requested. This will be reported to the gateway and the manager. However, our system does not automatically provide for a rollback mechanism to the previous state. This can be the subject of future work.

In this paper, we will compare the performance of our system using two different JPVM DOM-based approaches, namely a static weighted load balancing and equal work non-weighted load balancing approaches. We will also compare this to a single JPVM gateway approach.

System Architecture

Our framework is based on the XML/SNMP gateway architecture, which is shown in Fig. 2. Communication is between an XML-based Manager, an XML/SNMP Gateway, and SNMP Agents. In this paper, we present a static weighted JPVM-based approach for the implementation of the XML/SNMP gateway.

In this section we present the JPVM-based approach for XML-based Network Management. First, we present the single-DOM tree XML-based Network Management architecture. Then, we give a general background of the JPVM. Finally, we describe the proposed architecture and its implementation. We also present the algorithms for load balancing and our contribution to JPVM.

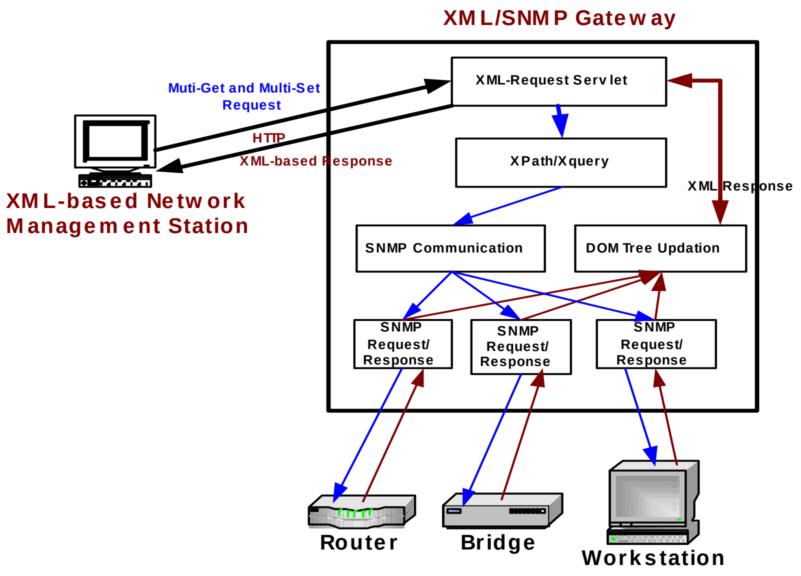

Fig. 2. Single-DOM Tree based Framework

Single DOM Tree-based Approach

The proposed architecture for the single-DOM tree has three main components as shown in Fig. 2.:

- XML-based Network Management Station (XBM).
- XML/SNMP Gateway.
- SNMP agents.

The XML-based request is represented as an XML document. The XBM prepares and sends the XML-based request to the XML/SNMP gateway. The request is received by the XML request servlet, which retrieves the number of target agents present in the request. It extracts the Xpath component of the request and sends it to the Xpath/Xquery module, which parses the XML-based request document. Parsing extracts the target MIB object present in the XML-based request received from the XBM.

Using these target objects and the target hosts, the SNMP communication module will send the SNMP-based requests to the agents and receives the SNMP responses. The DOM tree is updated with the received response values. The updated response DOM tree can be translated into any form according to the user requirements using the XSL style sheets. Here in our approach we apply the XML style sheet to convert the response DOM tree into an HTML format and it is transmitted over the HTTP protocol to the XBM. Another option would be to transmit the XML document to the XBM which will in turn convert it to an HTML document. This will provide more flexibility to the XBM to manipulate the response, at the expense of adding more processing overhead. Since our goal is to minimize the overhead of the manager, we have chosen the first option.

JPVM Background

Adam J. Ferrari introduced the Java Parallel Virtual Machine (JPVM) [11] library. The JPVM library is a software system for explicit message passing based on distributed memory MIMD parallel programming in Java. JPVM supports an interface similar to C and FORTRAN interfaces provided by the Parallel Virtual Machine (PVM) system. The JPVM system is easily accessible to the PVM programmers and has low investment target for migrating parallel applications to a Java platform. JPVM offers new features such as thread safety, and multiple communication end-points per task. JPVM has been implemented in Java and is highly portable among the platforms supporting any version of the Java Virtual Machine.

The JPVM system is quite similar to that of a PVM system. JPVM has an added advantage of the Java as a language for network parallel processing. In the case of PVM, we divide a task into a set of cooperative sequential tasks that are executed on a collection of hosts. Similarly, in the case of JPVM, one has to code the implementation part into Java. The task creation and message passing is provided by means of JPVM.

JPVM Interface

In this section we explore the JPVM interface that provides the task creation, and execution. The most important interface of the JPVM package is the *jpvmEnvironment* class. The instance of this class is used to connect and interact with the JPVM systems and other tasks executing within the system. An Object of this class represents the communication end-points within the system, and each communication point is identified by means of a unique *jpvmTaskId*. In PVM, each task has single a communication end-point (and a single task identifier), but JPVM allows programmer to maintain logically unlimited number of communication connections by allocating multiple instances of *jpvmEnvironment*.

First, we need to set the JPVM environment on all the hosts that we are interested to use for parallel communication. For this, we need to run the *jpvmDaemon* java program on all the hosts. By running *jpvmDaemon* threads, we just initiate the JPVM environment. These threads are not used until all the hosts know about their JPVM environment. Next, we need to start the Console on one of the *jpvmDaemon* running hosts. The console program can be started running the *jpvmConsole* java program. Then, we have to register or add the other *jpvmDaemon* hosts to the host running the console program. We add the hosts by giving the name and the port at which the *jpvmDaemon* started. This port is used during message passing between the JPVM hosts, and is the port through which the JPVM communication takes place.

JPVM Architecture

The proposed JPVM architecture is shown in Fig. 3. It has mainly 3 components, namely an XML-based Manager, JPVM gateways, and SNMP agents. All the JPVM gateways are configured to run daemon processes. There will be one JPVM gateway that will run the *jpvmConsole* in order to notify all the hosts one another's existence and this is called as the master JPVM gateway. The master JPVM gateway will communicate directly with the XML-based manager. The other JPVM gateways are known as slave JPVM gateways. These slave gateways communicate only with the master JPVM gateway. Hence, the JPVM-based network management is based on a master slave paradigm.

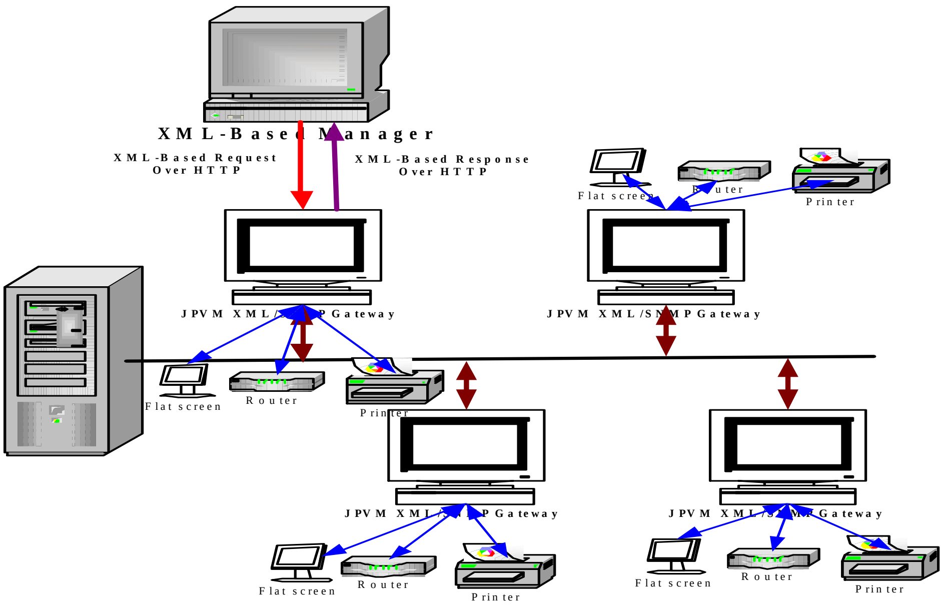

Fig. 3. JPVM Framework for Parallel XML-based Netwrok Management

Implementation of the Proposed Framework

The JPVM-based framework is implemented as a master-slave architecture, where a master JPVM is running at the web server. The master JPVM gateway receives the request from the XML-based manager. A *jpvmDaemon* program will be running on all the JPVM gateways. The master JPVM gateway is connected to a number of slave JPVM gateways, and will run the *jpvmconsole* program. The JPVM slave gateways have only the slave programs running on them for communication with the master JPVM and SNMP agents. The slave JPVM carries out the actual XML to SNMP translation and SNMP communication with the SNMP agents. The master JPVM status can be either working or not working. If the master has a working status, it can communicate with the SNMP agents after dividing the tasks.

JPVM Master Algorithm

The JPVM master gateway algorithm is presented in Fig. 4. The Master JPVM algorithm has three stages: initialization, waiting for the work, and termination. In the initialization stage, the master will start the JPVM environment, and create a pool of slave JPVM gateways. In the wait for request stage, the master will wait for the request from the XBM, and upon receiving the request it divides the work among the available pool of slave JPVM gateways, and dispatches the work to the slave JPVM gateways. It will wait for the response from all the slave JPVM gateways, and after receiving the response, it joins the responses into one response document. Then, it will apply XSL to the XML document before transmitting the response over HTTP protocol to XML-based manager. In the termination stage, the master JPVM will send the *stop* command to the slave JPVMs, and then exit from the JPVM environment.

Algorithm JPVM Master Gateway

```
Begin
  Initialization:
    Start the JPVM Environment
    Create Pool of JPVM Slave Gateways
    Initialize the JPVM _Spawn for each Slave
  Wait For Request:
    Divide the work
    Send the work to each Slave JPVM gateways
    Get the result from all the Slave JPVM gateways
    Join the work
  Termination:
    Send to each Slave the Stop command
    Exit from the JPVM Environment
End Master JPVM
```

Algorithm JPVM Slave Gateway

```
Begin
  Start the JPVM Environment
  Parse the RFC-1213
  While (true)
    Wait to receive the work from the Master
    If (Stop)
      Exit from the JPVM Environment
    If (Work)
      Get the XML-Document
      Do the Work.
  End While
  Exit from the JPVM Environment
End Slave
```

Fig. 4. Master and Slave JPVM Gateway Algorithms

Slave JPVM Algorithm

The slave JPVM algorithm is presented in Fig. 4. The slave JPVM gateway starts the JPVM environment and parses the RFC-1213 MIB objects during the master JPVM initialization stage. The slave JPVM will wait for the work from the master JPVM gateway. Once the work is received from the master, each slave JPVM performs Single DOM tree-based approach (converting the XML-request into SNMP requests, sending SNMP requests, receiving the SNMP responses, and updating the SNMP responses in the DOM tree). All the slave JPVM gateways will pass the XML response document to the master JPVM gateway. Then, all the slaves wait again for work from the master. This repeats until the master sends the terminate command to all the slave JPVM gateways.

Contributions to JPVM

JPVM supports basic data types like integer, long, string, character etc. The communication (message passing) between the different JPVMs is through these data types. XML-based network management requires communication by means of XML documents. JPVM does not support message passing of XML documents among the different JPVM. In order to support message passing of XML documents, we added new data types such as: XML document, NodeList, Node, and SnmpPdu to the current JPVM source code.

Static Weighted Load Balancing

In the equal work non-weighted load balancing approach, we assign equal work to all slave JPVM gateways (i.e., we divide the work based on the number of slave JPVM gateways present in the pool). This approach provides good performance only for a homogeneous network of workstations.

A second approach is the static weighted load-balancing algorithm in which we divide the work based on the processing speed of the workstations. In this approach, we assign a weight to the workstations depending on their processing speed. During the work assignment, a gateway will be assigned work according to its weight. The higher the weight the larger the amount assigned to the slave JPVM gateway.

The weights are assigned based on the base processor’s processing speed as follows: First, each workstation is assigned the same number of agents that it will communicate with. The workstation that takes the longest time to finish the work is taken as the base processor. The weight of this workstation is set to 1, and the weight of any other workstation is obtained by dividing the base processor time by the amount of time taken by this workstation.

The second approach provides better results when we have a heterogeneous network of workstations. Results are shown later that support this statement.

Experiments and Results

Experimental Setup

In the experimental setup for the XML-based network management using JPVM, the master JVPM gateway is connected to a number of slave JVPM gateways. All the JPVM gateways are windows workstation and running on windows 2000 operating system. The master JPVM gateway has a TOMCAT 5.0 web server running on it. The same experimental setup has been used with homogenous and heterogeneous systems. In the case of homogeneous systems the slave JPVM gateways are of equal processing speed and in the other case they are of different processing speed. The experiments were conducted from our University campus, and all the SNMP agents are connected over 100Mbps access network connection and a Gigabit Ethernet backbone. Each experiment was conducted for 25 runs. The maximum number of agents used in our experiments is 200. The request/response messages are for the system group MIB objects from RFC-1213.

The time elapsed between issuing the XML-based request from the XBM to the XML/SNMP gateway and the time the response is received from the XML/SNMP gateway back to the XBM is termed as the response time. We have shown in [12] that most of the response time is consumed during the communication between the XML/SNMP gateway and the SNMP agents, that is the SNMP-STACK communication. For example, more than 90% of the time is consumed by the SNMP-STACK communication when the number of agents exceeds 50. Our goal is then to reduce the SNMP-STACK communication time. This was achieved through the distribution of the work among multiple gateways. In our experiments, we will however compare the overall response time to show the improvements achieved.

Results and Discussion

Table 1 shows the response time values for single gateways (i.e., 350-No-JPVM, 350-JPVM, 711-No-JPVM, and 711-JPVM), homogeneous systems, heterogeneous systems, and static weighted allocation as the number of agent increases.

Table 1. Response Time values for Homogenous, Heterogeneous, and Static Weighted

	350- No-JPVM	350- JPVM	711- No-JPVM	711- JPVM	HOMO EqualWork	HETERO EqualWork	STATIC	STATIC (Agents assigned)	
Agents	**Response Time**							**350**	**711**
1	523.2	1221.8	609.4	737.0	1070.8	821.2	786.4	0	1
10	1528.6	2445.5	1131.5	1636.4	2160.7	1939.8	1551.1	3	7
20	3319.6	4534.6	2369.4	2834.2	2971.6	2692.9	2021.7	7	13
30	5717.9	7141.2	3575.5	4728.8	4256.0	3734.4	3304.4	10	20
50	11678.8	14061.2	6780.9	8233.8	6322.1	5370.6	5692.7	17	33
60	15779.5	18769.1	8949.7	10420.6	7849.9	6566.5	6936.2	20	40
90	31032.2	37661.0	18237.7	20030.8	13364.0	12238.7	11032.1	30	60
100	37481.5	45195.8	21733.8	24419.2	16435.4	14764.3	12724.9	33	67
110	44291.2	54004.7	25692.5	27860.2	18861.3	17302.0	13195.9	37	73
140	69174.2	80195.3	39770.3	41776.2	24561.4	21974.6	23042.4	47	93
150	78327.4	90753.5	42279.3	46507.0	26741.3	25396.4	25655.3	50	100
180	108866.2	129406.3	59818.3	66253.4	41106.2	37756.4	34308.4	60	120
190	123441.5	147638.4	65247.4	71603.0	44417.2	40355.1	37045.5	63	127
200	134577.5	153398.6	70516.2	76602.2	48740.3	44477.9	38905.3	67	133

Fig. 5. shows the response time for the homogeneous vs. heterogeneous systems for the system group MIB objects in the case of equal work assignment. The experiment is conducted with two homogeneous systems and then with two heterogeneous systems. The homogeneous systems are of 350 MHz processing speed Intel Pentium II processors and the heterogeneous systems are a 350 MHz processing speed Intel Pentium II processor and a 711MHz processing speed Intel Pentium III processor. In both cases, the response time is mainly dependent on the slower processor that takes longer to finish the work; since equal work is assigned to the two processors, whether these are of the same speed or not. Since the slowest processor is the same in both cases, i.e., 350MHz, the equal work assignment provides similar response times as shown in Fig. 5. Hence, homogeneous systems are better to use because, in the case of heterogeneous systems, the higher speed processor will be underutilized.

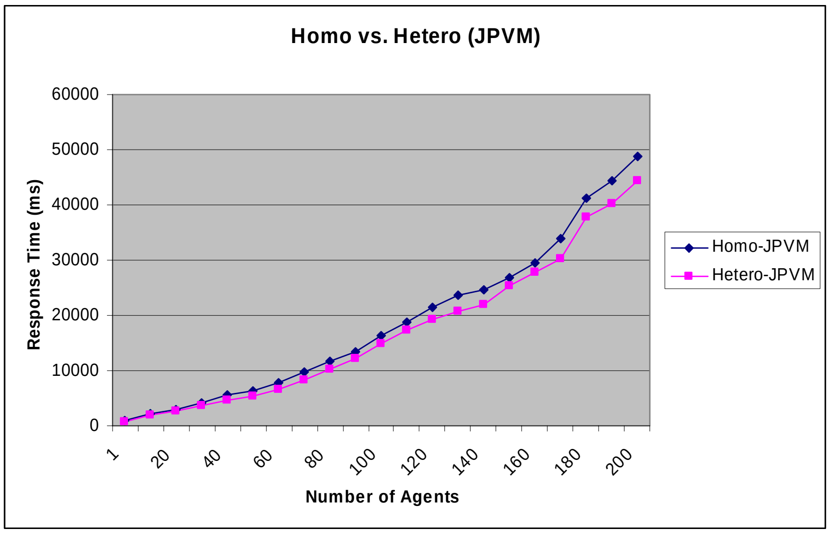

Fig. 5. Response Time for Homogeneous vs. Heterogeneous Systems

Fig. 6. shows the response time for heterogeneous systems vs. static weighted load balancing for the values in Table 1. Let us illustrate the difference between the results of the two approaches through the example of an XML-based request with 30 agents. In the case of heterogeneous systems, the response time is 3,734.4 ms, which is equal to requesting 15 agents by the slowest processor, i.e., 350 MHz processor. In the case of static weighted load balancing approach, the allocation of the work to each JPVM gateway is 10 and 20 respectively for the 350 MHz and the 711 MHz processors. In this case, the response time is 3,304.4 ms, which is equal to requesting 20 agents by the 711 MHz processor, i.e., 2,834.2 ms; in addition to the communication time for data packing and unpacking due to the use of two slave JPVM gateways. We can also observe in this case that the slower processor, i.e., 350 MHz processor, takes less time to request 10 agents, i.e., 2,445.5 ms; compared to the response time of the faster processor requesting 20 agents, i.e., 2,834.2 ms. Hence, the slower processor is underutilized in this case.

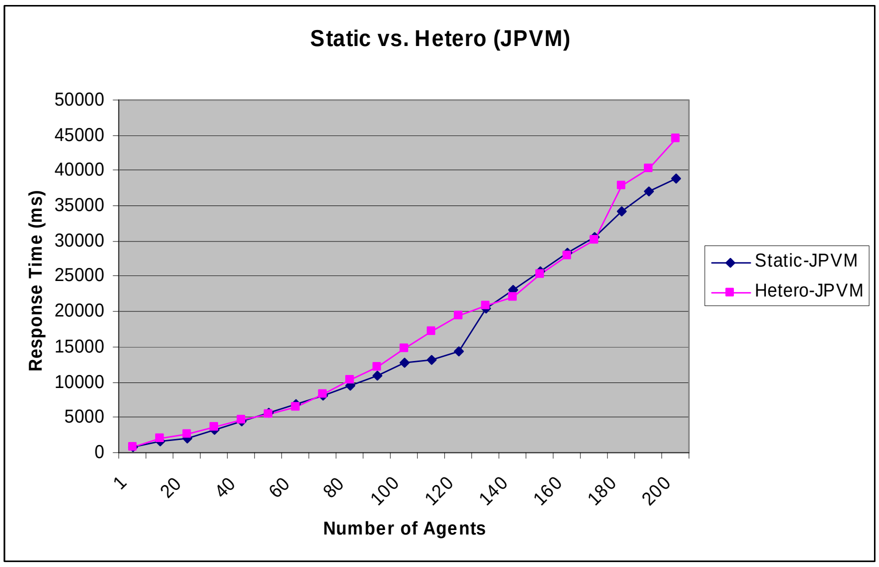

Fig. 6. Response Time for Heterogeneous Systems vs. Static Weighted Load Balancing

The response time in the case of heterogeneous systems with equal work allocation will be almost the same as that of the slower processor. In this case, as the number of agents increases, the faster processor needs lesser time to finish the work and thus is underutilized. There will be better response time with the static weighted load balancing compared to the equal work approach as the number of agents increases.

The choice of weights in this work is solely based on the processing speed of the systems used. We may be able to improve the results obtained by finding a better way to assign weights to avoid as much as possible underutilized gateways. This is the subject of future work.

Fig. 7. shows the comparison between all the experiments that were performed. We can see that the static weighted load balancing outperforms all the others.

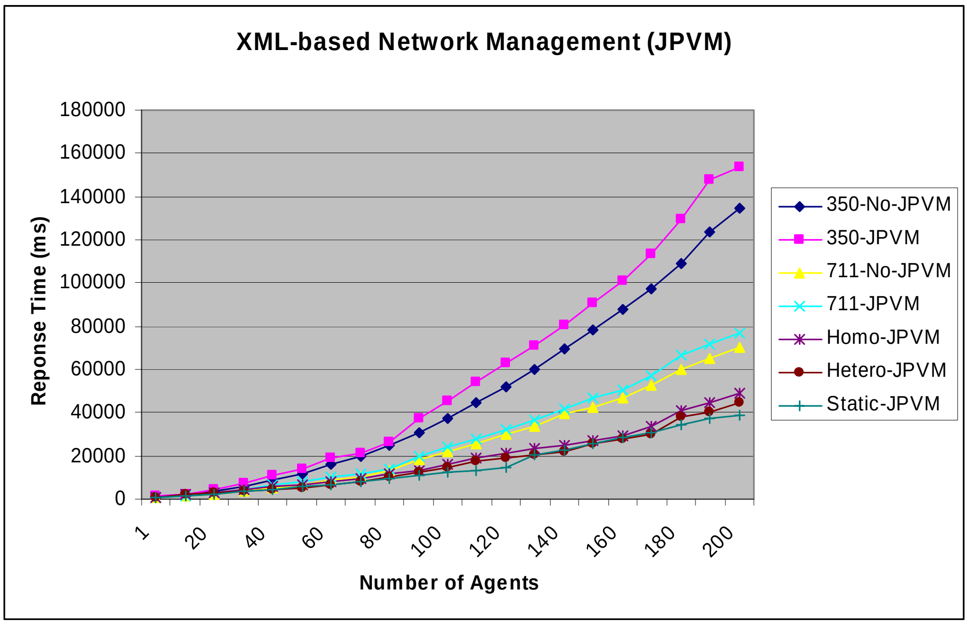

Fig. 7. Response Time for all experiments

Conclusion

In this paper, we presented load balancing approaches to XML-based network management, to distribute the load across multiple parallel JPVM gateways. We have shown that in the case of heterogeneous systems with equal work, the faster processor completes earlier and is underutilized. In addition, the static weighted load balancing approach with heterogeneous slave JPVM gateways provides a better response time than the equal work non-weighted approach, and a much better one than all single gateway approaches. We found as well that in the case of static weighted load balancing approach, the response time is closer to that of the faster processor which takes more time to complete the work. This also led to the fact that the slower processor became underutilized. The weight setting can be further tuned to improve the results obtained, but this will be the subject of future work.

Acknowledgement

The authors acknowledge the support of King Fahd University of Petroleum Minerals (KFUPM) in the development of this work. This material is based in part on work supported by a KFUPM research project under Fast Track Grant No. FT/2004-20. We would like to thank the anonymous reviewers as well for their valuable comments.

References

[1] Jeong-Hyuk yoon, Hong-Taek Ju and James W.Hong, "Development of SNMP-XML translator and Gateway for XML-based integrated network management", *International journal of Network Management*, 2003, 259-276.

[2] Mi-Jung Choi, James W. Hong, and Hong-Taek Ju, "XML-Based Network Management for IP Networks", *ETRI Journal, Volume 25*, November 6, 2003.

[3] J.P.Martin-Flatin, "Web-Based Management of IP Networks and Systems", Wiley series in communications Networking and Distributed Systems, 2003.
[4] Sqalli H.M., and Sirajuddin S., "Extensions to XML based Network Management", International Conference on Information and Computer Sciences (ICICS-2004), Dhahran, KSA, November 2004.
[5] W3C, "Extensible Markup Language (XML) 1.0", *W3C Recommendation*, October 2000.
[6] W3C, "XML Schema Part0: Primer", *W3C Recommendation,* May 2001.
[7] W3C, "XML Schema Part1: Structures", *W3C Recommendation,* May 2001.
[8] W3C, "XML Schema Part2: Data Types", *W3C Recommendation,* May 2001.
[9] Straus, F. "A library to access SMI MIB information", http://www.ibr.cs.tubs.de/projects/libsmi/
[10] Phil Shafer "XML-Based Network Management" – White Paper, *Juniper Networks, Inc.*, 2001, http://www.Juniper.net/solutions/literature/white_papers/200017.pdf
[11] Adam J.Ferrari, "JPVM: The Java Parallel Virtual Machine", http://www.cs.virginia.edu/jpvm/
[12] Sirajuddin S., and Sqalli H.M., "Comparison of CSV and DOM Tree Approaches in XML-based Network Management", 12th International Conference on Telecommunications (ICT-2005), Cape Town, South Africa, May 3-6, 2005.
[13] Hyoun-Mi Choi, Mi-Jung Choi, James W.Hong, "XML-based Configuration Management for Distributed System", Proc. of 2003 Asia-Pacific Network Operations and Management Symposium (APNOMS 2003), Fukuoka, Japan, October 1-3, 2003, pp. 599-600.
[14] Mani Subramanian, "Network Management: Principles and Practice", Addison-Wesley, Hardcover, Published December 1999, 644 pages, ISBN 0201357429.
[15] W3C, "Document Object Model (DOM) Level 2 Core Specification", *W3C Recommendation*, November 2000.
[16] W3C, "Document Object Model (DOM) Level 2 Traversal and Range Specification", *W3C Recommenda-tion*, November 2000.