An Adaptive Load-balancing Approach to XML-based Network Management using JPVM

Mohammed H. Sqalli and Shaik Sirajuddin
Computer Engineering Department
King Fahd University of Petroleum & Minerals
Dhahran, 31261, Saudi Arabia
E-mail: {sqalli, siraj}@ccse.kfupm.edu.sa

***Abstract* — SNMP is the most widely used network management protocol. Since SNMP is based on a centralized approach, it is confronted with scalability and efficiency problems when the network expands. XML-based network management is a new paradigm developed to overcome these limitations. In this paper, we propose a framework for adaptive load-balancing using JPVM for XML-based network management. The goal is to increase the efficiency of processing the management data, and decrease the communication time by distributing the management load across multiple XML/SNMP gateways. The load distribution among multiple gateways is adapted to achieve better results.**

***Index Term* — Network Management, XML, and JPVM.**

I. INTRODUCTION

The Simple Network Management Protocol (SNMP) is currently the most widely used protocol to manage network devices on the Internet. Nonetheless, today's network has incompatible infrastructure including different information models, information access methods, and management protocols. The main goal of network management systems 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 administrator has no choice but to use separate and incompatible management tools to manage the current heterogeneous network. Currently available management tools and framework, such as SNMP, are based on a centralized approach, and confronted with scalability and efficiency problems when the network expands. 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 [1] [2] consists of an XML-based manager (XBM), an SNMP/XML gateway and SNMP agents. We proposed in [4] 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 [5], we proposed a static weighted load-balancing approach for XML-based network management using JPVM, which we have shown provides better results than other load balancing and single gateways approaches.

In this paper, we present another JPVM-based approach to the proposed extended XNM, which is an adaptive load balancing approach that makes use of JPVM in XNM. We show that this approach improves even further on the results obtained in [5].

The rest of the paper is organized as follows; first we will introduce XML-based network management, and discuss the current related work. Then, we will give a general overview of the JPVM environment. The adaptive load balancing approach with JPVM will then be presented. The section that follows will include the experimental setup and results of comparing this approach to the static one. The paper ends with a conclusion.

II. 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 [6]. We can define our own Structure of Management Information in a flexible form using either Document Type Definition (DTD) or XML Schema [7][8][9]. 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 XSL 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. XML supports the transfer of large amount of data in a single document. All these advantages make XML a good candidate to solve the problems of scalability and efficiency of existing SNMP based NMS.

Figure 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. This, however, requires the development of an SNMP/XML gateway to exchange the messages between the XML-based network manager and SNMP agents.

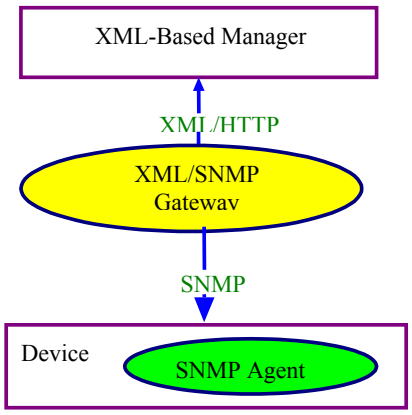

Figure 1. An XML-based Network Management Architecture

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.

III. Related Work

Martin-Flatin proposed using XML for network management in his research work on Web-based integrated network management architecture (WIMA) [3]. 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.

Strauss [10] 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 called "smidump".

Network devices developed by the Juniper Network are equipped with the JUNOS Operating system, which supports JUNOScript [11]. 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 network 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 network is working on implementation of XML document adapter for SNMP MIB modules using Net-SNMP and XML-RPC libraries.

Muller implemented an SNMP/XML gateway as Java Servlet that allows fetching of XML documents on the fly through HTTP [10]. MIB portions can be addressed through XPath expressions encoded in the URLs to be retrieved. The gateway works as follows. When an 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 these 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 validation of the algorithm, POSTECH implemented an XML-based SNMP MIB browser using this SNMP MIB to the 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 [4], a framework for extensions to an existing XML-based network management was proposed, which can reduce the processing time between the XML-based manager and the

SNMP agents. The extensions consist of new types of messages, including 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.

In [5], we proposed a static weighted load-balancing approach for XML-based network management using JPVM. This approach outperformed other load balancing approaches as well as the different single gateways options.

IV. System Architecture

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

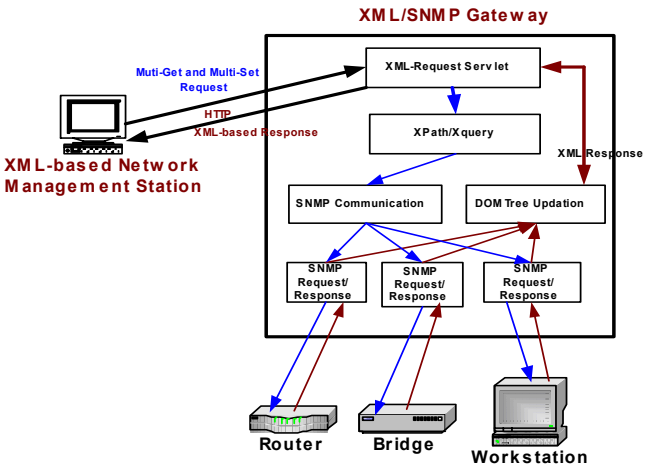

Figure 2. Single-DOM Tree based Framework.

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.

A. Single DOM Tree-based Approach

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

B. JPVM Background

Ferrari introduced JPVM [12] (Java Parallel Virtual Machine) 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 PVM (Parallel Virtual Machine) 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.

C. 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.

D. JPVM Architecture

The proposed JPVM architecture is shown in [5]. 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 of one another's existence and this is called 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.

E. 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 a 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 also communicate with the SNMP agents after dividing the tasks.

The JPVM master gateway algorithm is presented in Algorithm 1. When the master JPVM joins the responses into one XML response document, it will apply XSL to the this document before transmitting the response over HTTP protocol to the XML-based manager.

```
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 among Slave gateways
    Send/Dispatch the work to each Slave JPVM gateways
    Get the result from all the Slave JPVM gateways
    Join the responses into an XML document
  Termination:
    Send to each Slave the Stop command
    Exit from the JPVM Environment
End Master JPVM
```

Algorithm 1: Master JPVM gateway Algorithm

The slave JPVM algorithm is presented in Algorithm 2. 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 send an 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.

```
Algorithm JPVM Slave Gateway
Begin
  Start the JPVM Environment
  Parse the RFC-1213 MIB objects
  While (true)
    Wait to receive 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
```

Algorithm 2: Slave JPVM gateway Algorithm

F. Load Balancing Approaches

Load balancing involves assignment of tasks to each processor in proportion to its performance. The goal of load balancing is to assign the work proportional to the performance of the node or processor thereby minimizing the execution time of the application.

In the equal work non-weighted load balancing approach, the work is equally divided and assigned to all slave JPVM gateways (i.e., the work is divided 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 the work is divided based on the processing speed, i.e., the CPU rate, of the workstations. In this approach, we assign a weight to the workstations depending on their processing speed, and during the work assignment it will be given 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. More details about these results can be found in [5].

G. Adaptive Load Balancing Approach

The CPU rate of a particular processor cannot be directly translated to the workstation performance on a particular application, and so is not usually a correct measure of performance. However, for our purposes, the clock rate does provide a way to make a general approximation of the relative

performance of the workstation. Hence, in the static weighted load-balancing approach, where the weight of each processor is solely in function of the CPU rate, the response time obtained is not necessarily the best we can get.

In the adaptive load balancing approach, the work is initially allocated as that of the static weighted load balancing approach. Then, the weights of the processors are adapted so that all the processors are utilized optimally to minimize the overall response time. First, the response times are recorded for each processor for every one hour for loads ranging from 1, 10, 20 ... to 200 agents, with increments of 10 agents. The master JPVM algorithm for the adaptive load balancing approach is presented in Algorithm 3.

When a request with *n* agents arrives at the XML/SNMP gateway, the gateway initially divides the work based on the static weight of each processor, i.e., processor's speed. Based on this information, the work will be assigned to each slave JPVM. The response time is then collected for each slave JPVM gateway with the statically assigned number of agents. The average of all response times collected from all slave JPVM gateways is then computed. According to the average response time value, we compute the new number of agents to be assigned to each slave JPVM. For each processor, this number is determined by consulting the previously recorded values of response times and interpolating to the closest values. The new weights will then be assigned to each slave JPVM gateway.

```
Adaptive Load Balancing Algorithm - Master JPVM
Begin
  Initialization:
    Start the JPVM Environment
    Get Pool of JPVM Slave gateways
    Initialize the JPVM _Spawn for each Slave (Start of JPVM)
    Calculate Initial Weights based on the Processor Speed.
    Send the Initial work to each Slave JPVM gateway
    Do until Weights adapts
        Get the results From the Slave JPVM gateways.
        Get the response time from Slave JPVM gateways.
        Adapt the response time
        Send the work to Slave JPVM gateway
    End Do
  Termination:
      Send to Each Slave the Stop Command
      Exit from the JPVM Environment
End Master JPVM
```

Algorithm 3: Adaptive Load Balancing Algorithm

The algorithm maintains and promotes a fair distribution of the load by evaluating the weights for the slave JPVM gateways, and using this information as a metric for the next load distribution. The response time for load specific intervals are calculated periodically during the course of program execution. The redistribution is determined by the processor's most recent performance.

We will show in the next section that this approach outperforms the approaches discussed in the previous section. In our experiments, we will focus on the case when there are two slave JPVM gateways with different processing speeds.

This approach can be used in monitoring systems where the same tasks, i.e., collecting MIB objects information from SNMP agents, are frequently repeated. This will provide better response times with each new monitoring cycle.

V. Experimental Results

A. Experimental Setup

The master JVPM gateway is connected to a number of slave JVPM gateways. All the JPVM gateways are PCs running Windows 2000. The master JPVM gateway has Apache TOMCAT 5.0 web server running on it. The same experimental setup has been used for equal work, static and adaptive approaches. In all three cases, the slave JPVM gateways are of different processing speed, i.e., a 350 MHz Intel Pentium II processor and a 711 MHz Intel Pentium III processor. 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 experiment is 200.

B. Results and Discussion

Table **1** shows the response time values for the single gateway approaches (i.e., 350-JPVM, and 711-JPVM), equal work assignment, static weighted approach, and adaptive approach as the number of agent increases. For the equal work approach, both slave JPVM gateways are assigned the same number of agents, i.e., half the total number of agents. In the static weighted approach, the 711 MHz Intel Pentium III processor is assigned twice as much the number of agents as the 350 MHz Intel Pentium II processor. In the case of static and adaptive approaches, the number of agents assigned to each gateway is shown in the last four columns of Table **1**.

Table 1. Response Time values for Static Weighted vs. Adaptive

Agents	350-JPVM	711-JPVM	Equal Work	Static	Adaptive	Static (agents assigned)		Adaptive (agents assigned)	
	Response Time					350	711	350	711
1	1221.8	737.0	821.2	786.4	625.0	0	1	0	1
10	2445.5	1636.4	1939.8	1551.1	1087.6	3	7	4	6
20	4534.6	2834.2	2692.9	2021.7	1714.2	7	13	8	12
30	7141.2	4728.8	3734.4	3304.4	2914.2	10	20	11	19
40	11038.0	6499.4	4726.9	4394.9	3697.0	13	27	15	25
50	14061.2	8233.8	5370.6	5692.7	4993.2	17	33	19	31
60	18769.1	10420.6	6566.5	6936.2	5892.2	20	40	23	37
70	21405.7	11770.8	8199.6	8059.5	7058.0	23	47	27	43
80	26023.5	13751.8	10249.8	9445.0	8097.4	27	53	30	50
90	37661.0	20030.8	12238.7	11032.1	9597.4	30	60	34	56
100	45195.8	24419.2	14764.3	12724.9	10974.0	33	67	38	62
110	54004.7	27860.2	17302.0	13195.9	11762.8	37	73	39	71
120	63118.7	32006.0	19338.9	14319.2	12996.9	40	80	45	75
130	71224.4	36608.6	20815.2	20515.7	15583.9	43	87	50	80
140	80195.3	41776.2	21974.6	23042.4	18352.7	47	93	55	85
150	90753.5	46507.0	25396.4	25655.3	19873.4	50	100	60	90
160	100824.0	50656.8	27840.1	28348.6	22127.4	53	107	67	93
170	113200.7	57196.2	30250.5	30553.8	23823.5	57	113	72	98
180	129406.3	66253.4	37756.4	34308.4	26106.5	60	120	77	103
190	147638.4	71603.0	40355.1	37045.5	27705.4	63	127	80	110
200	153398.6	76602.2	44477.9	38905.3	30393.0	67	133	83	117

Figure 3 shows the response time for single gateway approaches with and without JPVM. We can see that the same gateway without JPVM performs little better than the same one with JPVM. This is due to the overhead of JPVM, but it is negligible. However, we can see that the response times of the 711 MHz Intel Pentium III processor is much better than the 350 MHz Intel Pentium II processor. These results were expected.

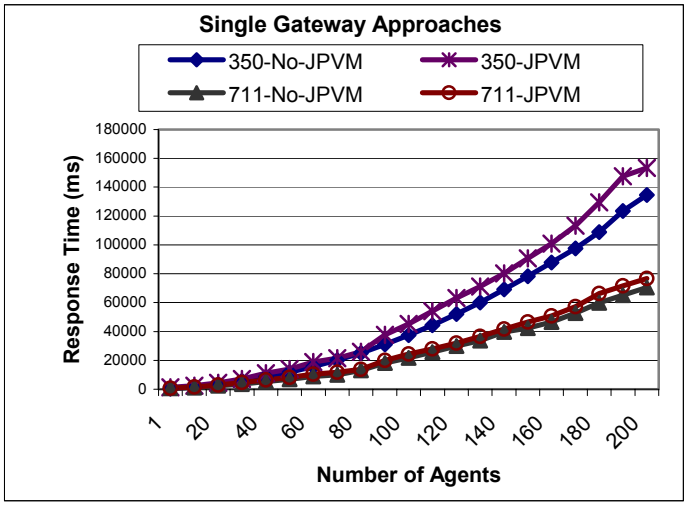

Figure 3: Response Time for Single Gateway Approaches

Figure 4 shows the response time for equal work, static, and adaptive load balancing with two slave JPVM gateways compared to the single 711 MHz slave JPVM gateway. The response time is better with adaptive load balancing compared to equal work and to static weighted load balancing. When the number of agents increases, the difference becomes more apparent. The recommendation is to use adaptive load balancing mainly when the network has a large number of agents.

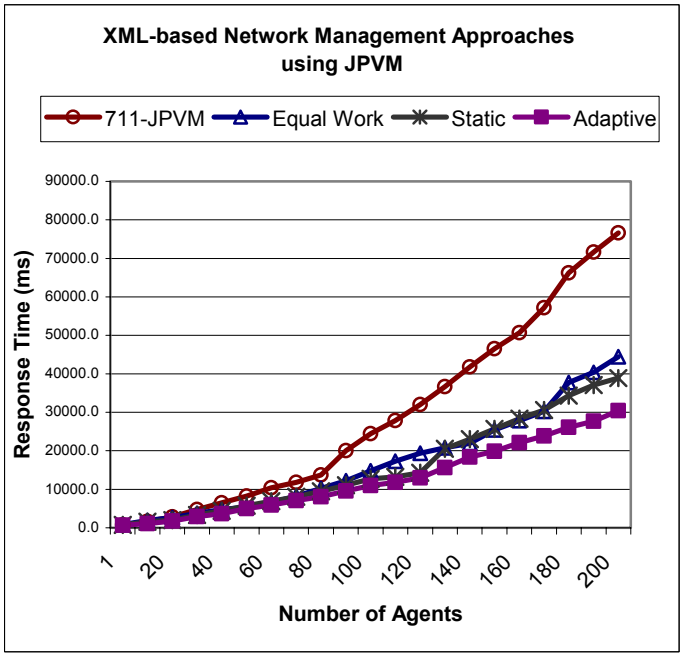

Figure 4: Response Time for Static vs. Adaptive Load Balancing

We can conclude that the response time of the adaptive load balancing approach outperforms the previous load balancing approaches used, i.e., equal work and static weighted. In addition, the adaptive approach will provide better response times as the master gateway learns more about the execution times of each slave gateway. Our goal is to improve these results further by investigating other dynamic and adaptive approaches.

VI. Conclusion

In this paper, we presented an adaptive load balancing approach to XML-based network management, to distribute the load across multiple parallel JPVM gateways. We have shown that adaptive load balancing outperforms other approaches, namely equal work and static weighted. The adaptive load balancing allocates the number of agents dynamically to each slave JPVM gateway to achieve a better efficiency. The weight setting can be further tuned to improve the results obtained, and this will be the subject of future work.

Acknowledgment

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.

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, Saudi Arabia, November 2004.

[5] Sqalli H. M., and Sirajuddin S., "Static Weighted Load-balancing for XML-based Network Management using JPVM", 8th International Conference on Management of Multimedia Networks and Services (MMNS-2005), J. Dalmau and G. Hasegawa (Eds.): LNCS 3754, pp. 228 – 241, Barcelona, Spain, October 24-28, 2005.

[6] W3C, "Extensible Markup Language (XML) 1.0", *W3C Recommendation*, October 2000.

[7] W3C, "XML Schema Part0: Primer", *W3C Recommendation,* May 2001.

[8] W3C, "XML Schema Part1: Structures", *W3C Recommendation,* May 2001.

[9] W3C, "XML Schema Part2: Data Types", *W3C Recommendation,* May 2001.

[10] Straus, F. "A library to access SMI MIB information", http://www.ibr.cs.tubs.de/projects/libsmi/

[11] Phil Shafer "XML-Based Network Management" – White Paper, *Juniper Networks, Inc.*, 2001, http://www.Juniper.net/solutions/literature/white_papers/200017.pdf

[12] Adam J. Ferrari, "JPVM: Network Parallel Computing in Java", Technical Report CS-97-29, December 1997.