A Framework for Web-Based Load-Adaptive Network Management System

Kamal A. Ahmat  
Department of Information Technology  
City University of New York  
New York, NY 11101  
kamal.ahmat@live.lagcc.cuny.edu

Russell Robinson  
Public Administration Institute  
Fairleigh Dickinson University  
Hackensack, NJ 07666  
russrob@fdu.edu

Abstract—With the advances in enterprise communication and web technologies, web has evolved into a platform for delivering interactive applications. Web-based applications are revolutionizing both the features that can be delivered and the technologies for developing and deploying applications. In this paper, we present a web-based load-adaptive network management system framework. Our system is scalable, extendable, modular and based on open architecture. It is also load-adaptive as it can adapt to the increase in the number of incoming Web Service requests so that the systems performance will not degrade under heavy request loads. To evaluate the proposed architecture, we built a multi-tiered web application testbed with open-source components widely used in industry. Experimental studies conducted on the testbed demonstrated the effectiveness of the proposed approach. By developing the platform and making it available to the research community, we hope to catalyze the development of an open source stack for Web-based network management and monitoring.

I. INTRODUCTION

Although the web was initially conceived as a vehicle for distributing documents, it has gradually evolved into a platform for delivering sophisticated interactive systems. Web-based systems are revolutionizing both the features that can be delivered and the technologies for developing and deploying applications. In the future, it is expected that web-based technologies will continue to rapidly penetrate many business areas, including systems and network management. Web-based network management systems (NMS) are not luxury, but they are rather a necessity to overcome the key problems that have faced traditional network management systems for the past few decades [10]. One of these problems is their very high cost due to the cost of the management platform added to the cost of the management applications. A second and more important problem is that of mobility: the management software is only available through the Graphical User Interface (GUI) on the NMS hardware console, for instance. This lack of user mobility seriously hinders management tasks: network managers want to access management applications from anywhere on the network, not just the NMS console, without compromising any privileges or features. Researcher community on the other hand lacks an open-architecture-based NMS that can be easily used to test their ideas without the need for reinventing the wheel and developing the whole new system infrastructure every time.

Web technology is being used to solve these problems. The use of a web browser as a universal client to access management applications completely eliminates the mobility issue and alleviates the costs incurred in the final solutions. Furthermore, the high level of portability of web-based systems makes it much easier to deploy the system on any hardware infrastructure and concentrate on developing and testing the new ideas.

However, designing web-based network management system is not free of challenges; it has to deal with a wide range of prominent issues that are inherited from the traditional system development and traditional network management systems concepts such as lack of high scalability, maintainability, extendibility, and standardization.

In this paper, we present an open-architecture-based enterprise-grade network management system that can scale to manage very large networks yet at the same time it fully realizes other important features such as the testability, usability and true extensibility. While the idea of using the web in IP network management is not new [7], [15] and [21], we believe our framework has a unique position in the research design space as it was purpose-built to unify the efforts in the network management area and enhance existing network management practices, and has been extensively tested in practice. More specifically, the contribution of this work is threefold:

1) A Web-based multi-tiered distributed system architecture. The system design is based on open architecture approach and different modules can be added or removed from the system without impacting or disturbing the system functionality.

2) A queueing-based self-adaptive event handler model to control the system performance and to handle high-level loads. This model is used to handle high amount of Web Service requests.

3) A computer software system for network management and monitoring. The system can discover the underlying active network status including the current status of devices and links and reflect bandwidth and topology changes in real time based on lightweight Simple Network Management Protocol (SNMP). The system architecture exploits the technical advantages provided by open-architecture, object-orientation, component soft-
ware, object frameworks, containerization, and rapid prototyping to achieve its goals of extendibility, flexibility, re-usability, and generality for network management applications. The architecture consists of layered, highly modular and loosely coupled components where interdependencies are well-defined and minimized. Our open source system is completely composable and permits components and objects to be independently developed and then integrated without disturbing or distressing existing software.

The rest of the paper is organized as follows. The next section presents the background necessary for the reader to be familiar with the rest of our work. Section III is divided into two subsections. The first subsection describes the concrete architecture of the system presented in this paper. The second subsection describes the load-adapting model that is used in our framework. Implementation issues, performance evaluation, and experiential results are described in Section IV. Finally, Section V concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Multi-Tiered Web System Architecture

Modern web-based systems follow the multi-tiered loosely coupled architecture (also referred to as n-tier architecture) where each tier consists of components. These tires in the 3-tiered architecture model are typically the presentation tier, application processing (i.e. business logic) tier, and data management (persistence) tier. This multi-tiered design pattern is a good practice to provide high level of scalability, maintainability, and reliability quality goals [5].

In the three-tiered architecture, the presentation-tier is responsible of user-application interaction and it converts and displays application data into a human-legible form. This tier is typically hosted on a web server and has three key functionalities: (1) receives requests from the clients and service static web requests; (2) forwards requests to the business logic tier; and (3) receives responses from the business logic tier and sends them back to the clients. Examples of typical web servers are Apache and Microsoft Internet Information Server (IIS).

The business logic tier is where all the logic is encapsulated. That is, it is the place where the data objects and business services in the application reside. The data objects contain the data and logic associated with the data while the business services are the algorithms applied to these data objects. Besides executing the algorithms, the business logic tier is also responsible for exchanging data with the persistence tier as needed. The business logic tier is typically hosted on an Application Server. Examples of typical Application servers are Apache Tomcat, Sun Java System Application Server, BEA WebLogic, IBM WebSphere, and JBOSS.

The persistence tier is responsible for encapsulating the access to the database. It exchanges Data Objects with the business logic tier and represents the data in a way that is independent of the underlying persistence technology. Examples of common persistence tier frameworks are Hibernate and iBATIS. The persistence tier is usually hosted on the Application Server along with the Business Logic. However, the database is typically hosted on a physically independent database server. Examples of typical database servers used in web systems are Oracle, Microsoft SQL Server, and MySQL. Figure 1 depicts these concepts.

Fig. 1. Typical three-tiered web system architecture.

B. Open Architecture Systems

The development of software is a long task that stretches beyond the analysis, design, implementation and test phases. While executing the software, new requirements arise, bugs are observed and to meet the new requirements, software updates are needed. One way to ease the software update process is to have an open architecture, which makes it possible to introduce new functionalities or modifications as fast as possible. One of the major design models, for open architectures, is the Component Based Model (CBM) [6]. CBM has two important properties which make it scalable, reliable, and extendable: First, components are implementations and architectural abstractions at the same time, which makes the architectural changes easier. Second, extensions are made (almost) independently of other components because the interaction between components is well defined in component interfaces [4].

III. ARCHITECTURE

In this section, we describe our system’s architecture. The presentation tier of this framework is described in detail in [11].

A. Conceptual and Concrete Architectures

This sub section describes the internal conceptual architectural design and presents the system-to-system relation levels and component distribution. The conceptual architectural design is depicted in Figure 2.

- System Dashboard This is the main graphical user command interface, which handles the user input commands and output reports. The user can perform tasks such as network monitoring, node controlling, data reporting, and all other tasks through the dashboard. This interface is not
• **Web-Based Interface** This interface is similar to dashboard except that it is web-based (i.e. it runs in the browser.) The web-Based interface along with the dashboard represents the presentation tier of this system. The presentation tier is designed as an independent framework so that it can be plugged-in with any NMS by implementing its set of APIs.

• **The Date Manager** The data related to the network infrastructure and performance is stored in the database. The interaction between the system and database is achieved through the persistence tier. This feature ensures the independency of the data repository and the network management platform. This component provides two levels of data caching: memory-based caching and storage-based caching. Two types of storage-based caching are supported: single device caching and distributed caching based on [19].

• **Data Reporting** The main functionality of this sub-component is to provide methods to get category-wise reports for each data category rule declared in the system. Filter is used during the process to get the IP interfaces for that category rule. Using the list of IP interfaces, node reports are created by referencing tables in the database. Finally reports are generated from the report nodes.

• **Event Control Unit** This component is considered as the system kernel (i.e. controller); it is responsible of event-handling and management. In more detail, it receives events from the user, persistence, and different middleware components and takes the appropriate action. This component is also responsible of managing network configurations.

• **Algorithm Repository** This component is the tool box where all business logic (i.e. algorithms) is stored. Each algorithm is defined as a separate module. The framework provides some core algorithms such as topology discovery and infrastructure monitoring. However, users can also develop and add their own components to the repository or modify existing ones and use them for experimental purposes.

• **SNMP Component** SNMP component is the centralized management unit that runs Simple Network Management Protocol and performs node-to-node and node-to-agent management. The SNMP component creates sessions between the managing node and distributed agents. These sessions encapsulate various parameters such as community strings, protocol version, and packet encoding. Once a session is created the manager code can communicate with the remote agent by sending requests and waiting for responses.

• **Scenarios Component** This component is responsible for executing an action based on SNMP events received by the agent. When a notification is received, the Scenario component invokes the corresponding daemon process. Examples of events are new node discovery and notification of monitored system status change. Event-response configurations are implemented through XML mapping.

• **Network Monitoring** This component is responsible for monitoring networks. The component interacts with the networks through pre-defined interfaces such as IPv4, IPv6, IPX, and others.

• **Network Configuring** This component receives its activation commands from the main centralized SNMP unit. By executing some certain management command this component considered the last operation done to configure or manage a network. For example, the configuration management commands could be applied on different levels such as application level for example, accounting, DSL, cable, and access register. Also could be cover networking level for example, IP, MPLS, and Gateways. It could be also applied on the routing level for example, router configuration, switches, and hubs.

• **Network Device Interfaces** Provides all the required interfaces with different network protocols and services such as, TCP/IP, ICMP, FTP, HTTP, DNS, DHCP, MSExchange, IMAP, and POP3. This interface helps the monitoring unit to discover and work with different network protocols.

• **MIB and Plug-Ins Manager**: Some vendors, such as Avaya [3], provide their devices with private MIBs. This use of private MIBs also occurs more widely with VLANs, as many network infrastructure vendors provide their devices with private MIBs that support VLAN
functionality. The purpose of this module is to check the vendor type and load the appropriate MIB structure, if the device doesn’t support standard MIBs, and read the information from the MIB and store it in the database. If the network contains devices that have private MIBs, then the structures of these private MIBs should be placed in the MIB repository in XML format. The structures of private MIBs are usually provided by device vendors. The controller uses macros to find the vendor of each machine and retrieves information using the corresponding MIB architecture if the device uses private MIBs.

B. Load-Adaptive Event Handler

1) Overview: One problem frequently encountered by web-based applications is overloading [17], [18], where the volume of Web Service requests for transactions at a site exceeds the service provider’s capacity. Overload causes longer delays to the clients or even denial of service and is a major reason for performance degradation. In network management systems, overloading can occur, for instance, due to frequent client requests to web services offered by the web-based system, especially in large networks.

To overcome this issue, we have developed a queueing-based web-service model component. The idea behind this model is to provide assure fairness of service provided to the system web services’ clients and to drop the minimum portion of requests when the site is overloaded in order for the accepted requests to meet their response time. Even though using single queues to handle request overloading is not new, to our best knowledge, this is the first time the concept of multi-queue multi-thread based request handlers being introduced and studied. Our model scheme is depicted in Figure 3 and works as follows. A controller instantiates a set of web service interfaces and assigns a thread and queue to each of them. The thread periodically takes performance measurements (measured delay $d$) of the corresponding interface from a monitoring agent, compares it with the desired performance (reference delay $D_{ref}$), and adjusts the admitting probability ($P_a$) to meet the performance goal ($D_{ref}$). The changes to the admitting probability can be actuated through an admission control (AC) module. Through the AC module, a request is being admitted with probability $P_a$ and being dropped with probability $P_d = 1 - P_a$. Requests are associated to queues based on a criterion such as request source IP address.

In our proposed architecture, there is a feed forward loop and a feedback control adaptive control loop working together to output the control command (admitting probability $P_a$) necessary to achieve a specified average delay target ($D_{ref}$). The feed forward loop is composed of a Queueing Model Predictor that takes measurements through a monitor from the computing system (i.e. web service interface) to be controlled, and uses classical results from queueing theory to predict a control command (admitting probability) necessary to achieve the specified average delay target given the currently observed average workload. Let’s call the admitting probability produced by this feed forward queueing model predictor as $P_m$.

Since queueing model used in the predictor serves only as an approximation of the real web site, the performance of the web site (measured average delay $d$) using the queueing model predicted admitting probability $P_m$ may be off from the targeted delay reference $D_{ref}$. To correct this “residual error”, we exploit an adaptive feedback loop. The feedback control loop compares the actual delay achieved to the desired delay reference and adjusts the admitting probability accordingly in an incremental manner to ensure that the desired delay is maintained.

In our scheme, we propose using adaptive control to design the feedback loop. In the adaptive control design, an online estimator will first estimate an appropriate residual error model based on the measurements of inputs (control command adjustments $\Delta P_a$) and outputs (residual errors $\Delta d$). Then the adaptive controller will produce the adjustments of admitting probability $P_a$ based on this online estimated model. The adaptive nature of the feedback loop can help to correct errors due to model inaccuracies and disturbances due to load changes using online measurements. Hence we anticipate the adaptive feedback loop will produce better control performance.

2) Model Design: Our abstraction for the service request handler is a set of M/GI/n Processor Sharing queues (M/GI/n/PS). There is a major reason to use this queueing model in our design. Modeling computing systems by multiple queues is important to handle scalability issues. Using single queue model may result in dropping higher numbers of request and consequently will affect the quality of service. On the other, using multiple-queue-based model can lead to more fair service provided to all clients, if requests are distributed on queues based on appropriate policy. Consequently, this abstraction encapsulates the (bottleneck stage) of the multi-tier web applications and supports higher level of scalability. The control law we use for each queue in our model is obtained by series of reductions and given by Formula 1. Details are omitted due to space constraints and will appear in the full paper.

$$\phi(k)^T \phi(k) = y^*(k+d) \tag{1}$$

where $y^*(k)$ is the reference input at time instance $k$. In our case, since we want to make the “residual errors” to diminish, we set $y^*(k) = 0$. The above algorithm begins with initial condition $P(-1) = p_0 I$ and $p_0 > 0$. 

Fig. 3. Architecture of Queueing-Model-Based Request Handler.
IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this section, we describe our system implementation and evaluate its complexity and its performance. Furthermore, we discuss our practical experiences and outline the limitations of the chosen architecture.

A. Implementation

The architecture of our system follows the current best practices used in the market for developing state-of-the-art systems. Our middle-tier is based on the Spring [18] framework, which is a full-stack Java/JEE application framework, and the persistency is based on Hibernate framework [13]. The reasons why we select this framework combination are: 1) all of them are open source projects and freely available; 2) their performances are among the highest of all individual components; 3) they are widely used on the Internet, even in commercial sites. Hence this combination is quite representative of the current technology.

To illustrate the request-to-response flow, we consider a topology edit request made by a network manager. A network manager interacts with one or more views presented in the browser and submit an HTTP request. For sake of simplicity, we consider a topology edit request as an example. The request goes to the business faade module which is responsible for handling requests through the Request Intercepting Filter component. The business faade first does the necessary security checks such as validating the session then passes the request along with the business object to the corresponding EJB bean, which in turn handles the request object to the to the request handler. The EJB tier consists of an EJB container that handles the data access logic by taking care of loading and unloading algorithms and allocating and deallocating resources. There are two main reasons for using the faade and EJB components in our design: (1) The faade encapsulates and hides the complexity of locating the appropriate EJB on the application server; and (2) The EJB container decouples the business logic (i.e. algorithms) from the system infrastructure. Thus, allowing developing and testing different algorithms distinctly and in parallel. The user needs only to configure the XML file to reflect the algorithms to be executed. The request handler then object invokes the correct action (e.g. by calling the appropriate topology discovery algorithm) The logical flow of topology edit request is depicted via the sequence diagram model in Figure 4.

The request handler has been implemented by modifying Tinyproxy 1.6.1 [14]. Tinyproxy is a lightweight and fast HTTP proxy that consumes less resource than fully equipped proxies such as Squid [9]. In our testbed, all HTTP requests from clients flow through Tinyproxy and are forwarded to Apache web server. Responses from the web server also return back to clients through Tinyproxy. We have implemented a process called monitor within the event handler to monitor both HTTP requests from clients and responses from the web server. The monitor also calculates parameters such as client request rates and other metrics such as average response time of requests. The event handler of two parts: The Queueing Model which takes the measured client request rates from the sensor module and produces the model predicted admitting probability $P_a$ and the Feedback Controller which takes the measured average response time as input and produces the admitting probability adjustment $\Delta P_a$. Then the combined dropping probability $P_d = 1 - (P_a + \Delta P_a)$. In our implementation, we confine $P_a$, $\Delta P_a$ to be within range (0.1; 1.0) and $P_d$ to be within range (0.0; 0.9). The event handler randomly drops a request from clients based on the dropping probability $P_d$.

B. Performance Evaluation

In this subsection, we describe the testbed set-up and the hardware and software environment we used to validate the effectiveness of the proposed architecture. A simpler prototype of this system was tested to discover the topology of large networks [12]. In this paper, we focus on testing the performance of the adaptive request handler. Figure 5 shows our testbed infrastructure. The testbed is composed of four machines. One of them is used as client workload generator and the other three machines are used as Web server, application server and database server respectively. All machines were connected via 100 Mbps Ethernet connections. The client machine was equipped with a 2.8 GHZ Intel Pentium IV processor and 512MB RAM.

TPC-W client emulator [20], an industry standard benchmarking tool, was used as synthetic workload generator on the client. It is a transactional web benchmark specifically designed for evaluating web-based systems. Thus, it implements all functionalities that typical web clients provide, including multiple online browser sessions, dynamic Web page generations, authentications and authorizations, and so forth. The Apache Web server machine has a 1GHZ Intel Pentium III processor and 256MB RAM, which runs Apache 2.0.7 [1].
The application server machine has a 1.5GHz Intel Pentium III processor and 256MB RAM, which runs Tomcat 5.0 [2]. The database server machine has a 1GHz Intel Pentium III processor and 512MB RAM, which runs MySQL 4.1.7 [17].

For testing purposes, we configured the client workload generator to issue topology edit requests. The reason for selecting this scenario specifically is that executing the topology edit module requires running database queries to read the AFTs and other topology related data, then applying the user’s change request by loading and running the topology discovery algorithm. Thus, it tests the efficiency of the system’s full cycle flow. The database was configured to contain 10,000 nodes.

Two sets of workloads were used in our tests. The first set, Workload A is a simple workload which has exponentially distributed inter-arrival time with mean 0.025 sec. The second set, Workload B is a more complicated workload which is changing. From time \( t = 0 \) sec to \( t = 150 \) sec. At time \( t = 250 \) sec, a second workload with the same mean inter-arrival time (0.025 sec) joins in, which makes the total request rate twice as much as that of Workload A. We have created five web service interfaces such that each interface has its own queue and event handler instance. The requests were mapped to a specific queue based on their ranges (e.g. network IP addresses). The number of request generating clients was equal to the number of event handler instances and loads were generated by following Workload B method.

![Deployment diagram](image)

**Fig. 5.** Deployment diagram which is used as testing infrastructure.

### C. Practical Experiences

Interestingly, our experimental evaluation demonstrates the advantages of integrating the adaptive request handler to achieve better performance. As we can see from Figure 6, using the adaptive request handler resulted in very persistent and efficient response time. It is worth to mention that the average percentage of dropped requests was considerably low and didn’t exceed 12 %. However, as shown in Table 1, we observed that the load was not evenly distributed on all event handlers as few event handler instances received high number of requests comparing to others, which was directly dependant on the size of IP ranges (i.e. number of devices in each network).

**Table 1**: Percentage of request distribution on the five request handler’s queues, where \( n \) is the number of IP ranges

<table>
<thead>
<tr>
<th>( n )</th>
<th>( Q_1 )</th>
<th>( Q_2 )</th>
<th>( Q_3 )</th>
<th>( Q_4 )</th>
<th>( Q_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>73</td>
<td>5</td>
<td>3</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>9</td>
<td>70</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>9</td>
<td>70</td>
</tr>
</tbody>
</table>

**Fig. 6.** Performance results for Workload A and Workload B comparing with the delay reference.

### V. Conclusion

In this paper, we presented an open-architecture based network management system that supports high level of scalability, extendability, and reusability. Experimental tests conducted on our framework demonstrated the system’s efficiency and adaptability to high levels of service requests. However, better service-request distribution mechanisms could be developed provide faster response to larger networks. By making this open source framework available to research community, we hope it will catalyze research and development in web-based network management and monitoring areas.

### References