Measuring Overhead for Distributed Web Service Handler

Beytullah YILDIZ
Department of Computer Engineering
TOBB University Economics and Technology
Ankara, Turkey
E-mail: byildiz@etu.edu.tr

Geoffrey C. Fox
Department of Computer Science
Indiana University
Indiana, USA
E-mail: gcf@cs.indiana.edu

***Abstract*—Service Oriented Architecture perfectly manifests itself in Web services, which create seamless and loosely-coupled interactions. Web service utilizes supportive functionalities such as security, reliability and so on. These functionalities are called as handlers, which incrementally add new capabilities. However, adding new handlers into the execution path may cause performance and scalability problems. Distribution of handlers solves these problems by providing abundant computing resources. However, pulling a handler out of its native place and positioning it away from Web service endpoint brings additional costs. Hence, we will investigate the overhead of handler distribution for various environments.**

Keywords-Web Service, Distributed Computing, Handler Structure, Multi-core

I. INTRODUCTION

Web service is defined by World Wide Web Consortium (W3C) as a software system that provides a standard means of interoperating software applications, running in variety of platforms[1]. It utilizes Web Service Description Language (WSDL) to provide a standard way to define services [2]. A Web service can be published to a Universal Description Discovery and Integration (UDDI) registry to be discovered and defined how to be interacted over Internet [3]. In addition, Web service framework uses Simple Object Access Protocol (SOAP) as de-facto standard to exchange structured information [4]. Requests and responses travel within SOAP messages. Hence, Web services strongly employ SOAP processing engines and transport helpers to contribute the interactions. These functionalities are combined in middleware system called Web service container, which essentially hides the complexity of SOAP processing and details of message transportation.

Web service employs additional functionalities, which are called handlers, by utilizing the extensibility feature of SOAP. Depending on the service container, these functionalities can be called as filters too. Generally, a Web service container provides a handler processing pipeline so that many handlers can contribute to a service interaction. In other words, many capabilities can be incrementally added to a service interaction. Even though handlers improve service capabilities, they may complicate a service interaction because of having too many handlers in a single processing chain. This may be inevitable in some situation to offer necessary qualities. On the other hand, handlers can become autonomous processing nodes. Hence, they can be separated away from the service endpoint with the intention of creating more powerful, efficient, scalable and modular service environment. Web service architecture supports this separation without harming correctness of the execution. When handlers are deployed away from a service endpoint, they become individual applications running without knowing each other. Hence, the detached and distributed handlers are needed to be orchestrated and managed so that they can achieve the execution, which was successfully happening before.

There are several reasons to separate a handler from the service endpoint. We may need to benefit from additional resources such as processor, memory and storage space. We may want to have a powerful architecture by offering a more modular and scalable structure. We may need to increase usability. Finally, we may successfully introduce concurrency to the handler execution. However, all these advantages do not come for free. The additional requirements for the orchestration and management of the message execution bring extra burden to the services. Hence, we will investigate overhead for Web service handler distribution.

II. DISTRIBUTING HANDLERS

Handlers are very necessary architectural components of Web service framework. Distribution of the handlers to the individual physical and/or virtual machines provides many advantages and opens doors to the immense computing resources. The computing power of machines almost doubles every year following the projection of Moore's law[5], the network speed also catches up with the same pace. Hence, the obtainable computing power increases steadily. Moreover, many other resources also became accessible such as application software, storage, and sensor and so on. We may hit barrier if we insist to utilize single machine for Web services while we can access many computing resources in the remote places. Therefore, we build architecture to distribute handlers, shown in Figure 1.

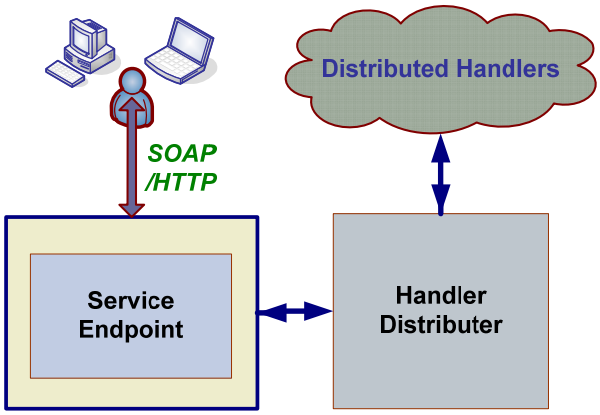

Figure 1 : Distributing Handlers

In general, conventional handler structures does not benefit from handler distribution. JAX-RPC [6], Apache Axis [7] and Web Service Enhancement (WSE) [8] deploy handlers into the computer where the service endpoint resides. On the other hand, there exists a work whose intention is to distribute the handlers. DEN/XSUL targets directly to the Web Service security processing steps without touching the service logic at all [9, 10]. It tears and granulates the security processing node and deploys the sub-tasks as individual services. This approach sets an example to distribute the handlers as Web services.

Utilizing Web service approach for the handlers provides several benefits. The first benefit is to be able to remove bottlenecks from the SOAP processing pipeline with a very well-known style. Additionally, service based-approach improves the interoperability of the deployment. Moreover, this approach is able to utilize the tools that have been already implemented for the Web services.

On the other hand, we follow a different approach. We have created an architecture, Distributed Handler Architecture (DHArch), to provide a scalable, efficient and modular environment for the handlers. It is basically a distributed Web service handler container, a specialized middleware system. It basically removes the bottlenecks from SOAP processing pipeline by using additional resources and providing an environment for the distributed execution.

DHArch efficiently distributes handlers to the various environments. It is able to utilize a cluster containing heterogeneous computers as well as a single computer utilizing multiple processors or cores. It is capable of executing distributed handlers in a single processor machine too. In this scenario, virtual machines provide necessary distributed environment.

DHArch is able utilizes a Message Oriented Middleware (MOM) for the transportation purpose. MOM is a powerful tool to provide asynchronous, reliable, efficient delivery mechanism. In addition to excellent messaging capability, it can provide a queuing mechanism for the handler execution to regulate the message flow [11, 13].

On the top, an orchestration mechanism is introduced to coordinate tasks [14]. Additionally, supportive mechanisms are provided to manage a message execution. Queuing systems, data structures are those entities that we want to mention for the architecture.

III. Measurements And Analysis

A. *Methodology*

In order to calculate the overhead resulting from the handler distribution, we utilize Apache Axis environment. It benefits sequential execution for the handlers. Handlers cannot be distributed to exploit additional resources [15]. On the other hand, DHArch is able to distribute handlers. Additionally, handlers can be executed concurrently by using both pipelining and handler parallelism.

For the sake of the fairness, the results have been gathered by utilizing the same environments. The handlers are completely same. The number of the handler in an execution is also equal in every step. The only difference is the distribution. Measurement starts with 1 handler. The number of the handler is increased by 10 in every step. We continue to add the same handler into the execution path until having 50 handlers. Figure 2 illustrates how the handlers are deployed in Apache Axis.

Every measurement is observed 100 times; a client performs the same requests 100 times in every step. At the end of the measurement, the service elapsed times are collected and an average value is calculated. After gathering the values in both environments, the overhead is calculated with the following formula:

$$Overhead = (Tdharch - Taxis) / N \quad (1)$$

Where, Tdharch is the elapsed time of a service utilizing DHArch. Taxis is the elapsed time of a service utilizing Apache Axis. N is the number of the handlers in the deployment.

Any performance improvement mechanism such as parallel execution is not exploited to find out the pure overhead added over the non-distributed execution. Handlers are running sequentially with the same conditions in the distributed environment too. The same deployment strategy is applied in the distributed environment. Figure 3 illustrates the sequential deployment of the distributed handlers. Handlers are deployed to a separate computing resource such as core, processor or physical machine.

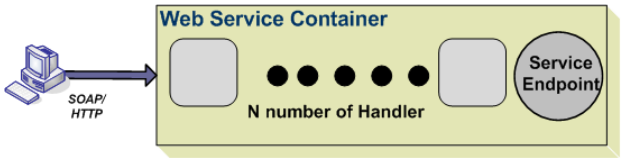

Figure 2 : Apache Axis sequential handler deployment to measure the overhead

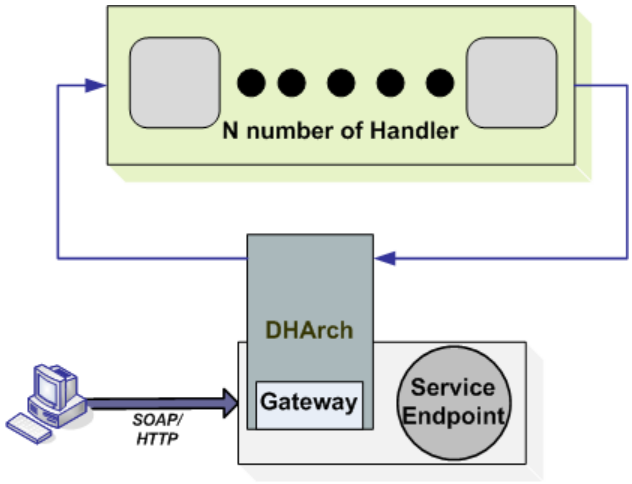

Figure 3: DHArch sequential handler deployment to measure the overhead

We give a special attention to the measurements in multi-core system. The utilized machine in this experiment has 1.2 GHz UltraSPARC T1 processor that contains 8 cores running Solaris Operating System, 4 threads per core, with 8GB physical memory. The second environment is a cluster sharing a Local Area Network. The computers in this cluster have the same hardware features. They utilize Fedora operating system in Intel Xeon CPU running on 2.40GHz and 2GB memory. The last environment is a single computer, utilizing single 2.80GHz processor with 1.5 GB memory. It is running Red Hat Enterprise Linux AS 4 operating system.

B. Measurements

The first experiment is conducted in multi-core system. We initially collected the results in Apache Axis handler structure. Then, the same scenario is repeated in the distributed environment. Figure 4 illustrates the service elapsed time and standard deviation.

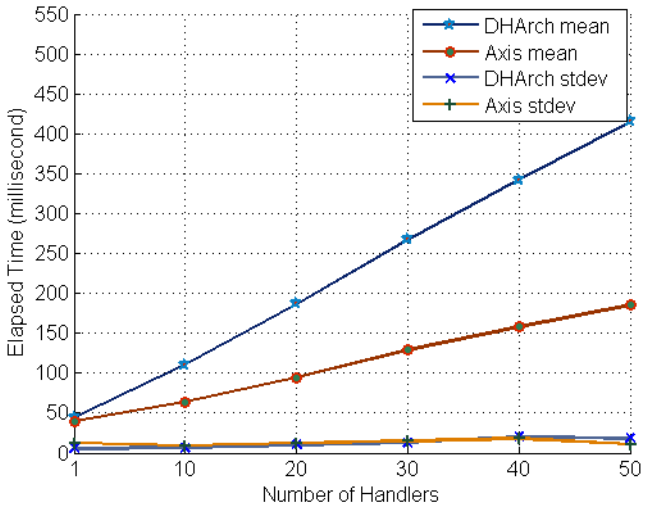

Figure 4: Comparison of handler execution in Apache Axis and DHArch for a multi-core system

Table 1 : Overheads of a handler distribution in the multi-core system for the increasing number of handlers in the execution path

Number of handlers	1	10	20	30	40	50
Overhead (msec)	4.52	4.59	4.63	4.61	4.60	4.59

Adding a new handler into the execution path linearly increases the processing time. This pattern is seen in the distributed environment as well. The formula 1 is applied to calculate the overhead. We observe that the overhead is almost equal for the increasing number of handlers which are added to the execution path. Table 1 shows the numerical values. Adding new handlers does not cause an unreasonable fluctuation. This is an expected outcome from a stable and scalable system. Of course, we should note that the pattern should be expected to change when the system resources start saturating. However, we do not expect that the average number of handlers in a single execution exceeds 50.

For the second environment, a cluster sharing LAN environment is utilized to measure the overhead. Three computers are used for the deployment in the distributed environment. Service endpoint and the messaging broker are deployed to an individual computer. Similarly, handlers are distributed to separate individual virtual machines in a single computer. The gathered results resemble to those collected in the multi-core system. Figure 5 depicts the results. Even though the tasks are carried to/from the distributed handlers by using LAN, the overhead is lower than that in the multi-core system. Table 2 shows the numerical values. This illustrates that the processor speed affects the overhead more than the network speed. Previous configurations do not have network latency. However, we must know that the network speed in this hardware configuration has a minuscule effect due to the usage of the computers sharing a LAN. The results would be different if the computers use a Wide Area Network (WAN).

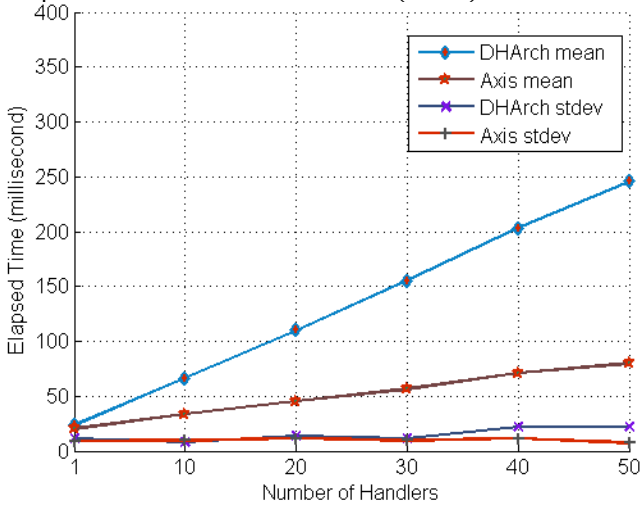

Figure 5 : Comparison of handler execution in Apache Axis and DHArch for the cluster

Table 2 : Overheads of a handler distribution in a cluster utilizing Local Area Network for the increasing number of handlers in the execution path

Number of handlers	1	10	20	30	40	50
Overhead (msec)	3.3	3.31	3.25	3.29	3.30	3.31

Finally, we have conducted the experiment in a single processor system. Figure 6 shows the results gathered for this configuration. Similar to the previous hardware configurations, the time for the execution of a service in the distributed environment is higher than those in Apache Axis environment because of the overhead resulting from the handler distribution. Although this system has a faster processor capability than the previous configurations and there is not a message transferring cost coming from the network usage, the overhead is not the smallest one. This must be due to thread scheduling. In this configuration, handlers are distributed into virtual machines instead of cores, processors or individual physical computers. In other words, handlers, service endpoint and messaging broker share a single processor to execute their tasks. Hence, the thread scheduling causes performance degradation so that the overhead is not the smallest one.

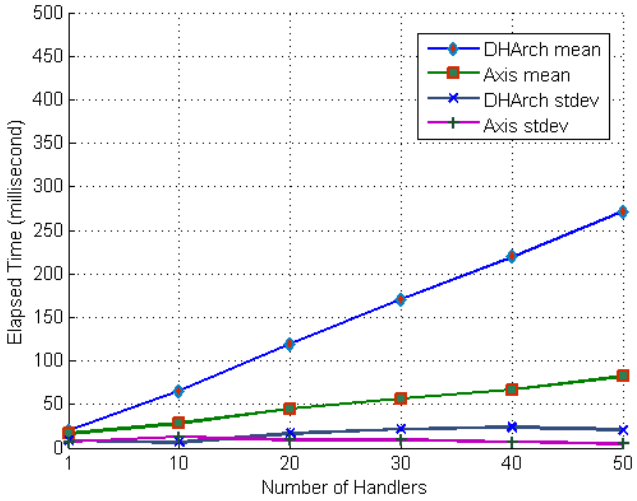

Figure 6 : Comparison of handler execution in Apache Axis and DHArch for a single processor system

Table 3 : Overheads of a handler distribution for a single processor system for the increasing number of handlers in the execution path

Number of handlers	1	10	20	30	40	50
Overhead (msec)	3.74	3.73	3.72	3.80	3.81	3.79

IV. CONCLUSION

Handler is a crucial aspect of Web service architecture because of the key importance in the execution path. However, the way of utilizing handlers and their structures become important when the number of the necessary functionalities increases. The experiments have provided us a clear understanding of the behavior of the distributed environment for the Web service handlers. The overhead value is mostly affected by the computing power and the network speed.

A single processor computer may not be as good as the computer which utilizes additional computing power for the performance wise. It starts to be affected by frequent context switches. By exploiting multi-core system and/or a cluster, we are able to remove the limitation over the computing resources.

Multiple computers can be efficiently utilized for the distributed execution. Each handler may acquire an individual computer within a network to contribute to the execution with the additional computing power. Even though there are overheads and obstacles for the distribution and the management of the execution, the use of the additional computer provides suitable environment for the handlers due to the improvements in the network speed, especially in Local Area Network (LAN).

Since the computers sharing LAN has more powerful processors than that utilizing multi-core processor, the overhead in LAN environment is better in the benchmark results. The effect of the message transferring on the overhead is minuscule because the distance is short and the LAN network provides fast message transferring environment. However, the network latency should be expected to become main factor for the overhead if the distance increases, especially while utilizing Wide Area Network (WAN). On the other hand, having faster processor in multi-core computers may provide better opportunity. Additionally, network latency coming from the transferring messages between computing nodes can be eliminated totally. In this situation, multi-core environments would be best option for the handler distribution.

As a result, we witness that the overhead is affordable. It can be easily compensated by exploiting the advantages, which are originated from the utilization of additional distributed resources. These benefits may even provide substantial gains. One of the ways is to utilize parallel execution for the distributed handlers.

REFERENCES

[1] Web Service Architecture, http://www.w3.org/TR/ws-arch/.
[2] Web Service Description Language (WSDL), http://www.w3.org/TR/wsdl.
[3] Universal Description Discovery and Integration (UDDI), http://www.uddi.org/ .
[4] Simple Object Access Protocol (SOAP), http://www.w3.org/TR/soap12-part1/.
[5] Lundstrom, M., APPLIED PHYSICS: Enhanced: Moore's Law Forever? Science 2003 Vol. 299. no. 5604, pp. 210 - 211.
[6] Java, A.P.I., for XML-Based RPC (JAX-RPC). 2003.
[7] Apache Axis, http://ws.apache.org/axis/.
[8] Microsoft Web Service Enhancements (WSE), http://www.microsoft.com/downloads/details.aspx?FamilyId=FC5F06C5-821F-41D3-A4FE-6C7B56423841&displaylang=en.
[9] Shirasuna, S., et al., Performance comparison of security mechanisms for grid services. p. 360-364.
[10] Slominski, A., et al., Asynchronous Peer-to-Peer Web Services and Firewalls, In 7th International Workshop on Java for Parallel and Distributed Programming (IPDPS 2005), April 2005.
[11] Pallickara, S., Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.
[12]Fox, G., 2004. A Scheme for Reliable Delivery of Events in Distributed Middleware Systems. In Proceedings of the First international Conference on Autonomic Computing (Icac'04) - Volume 00 (May 17 - 18, 2004). ICAC. IEEE Computer Society, Washington, DC, 328-329.
[13]Tai, S., Thomas A. Mikalsen, and Isabelle Rouvellou, Using Message-oriented Middleware for Reliable Web Services Messaging. Lecture notes in computer science (Lect. notes comput. sci.) ISSN 0302-9743 , 2003.
[14] Beytullah Yildiz, Geoffrey Fox, Shrideep Pallickara An Orchestration for Distributed Web Service Handlers The Third International Conference on Internet and Web Applications and Services ICIW 2008 June 8-13, 2008 - Athens, Greece
[15] Yildiz, B., S. Pallickara, and G. Fox, Experiences in Deploying Services within Apache Axis Container, Proceedings of IEEE International Conference on Internet and Web Applications and Services ICIW'06 February 23-25, 2006 Guadeloupe, French Caribbean.