SOAP Request Scheduling for Differentiated Quality of Service

Ching-Ming Tien[1], Cho-Jun Lee[2], Po-Wen Cheng[2], and Ying-Dar Lin[1]

[1] Department of Computer and Information Science,
National Chiao Tung University, 1001, Ta Hsueh Road,
Hsinchu, Taiwan 300
{cmtien, ydlin}@cis.nctu.edu.tw

[2] Computer and Communications Research Laboratories,
Industrial Technology Research Institute, 195 Chung Hsing Road,
Section 4, Chu Tung, Hsinchu, Taiwan 310
{ChoJunLee, sting}@itri.org.tw

Abstract. This paper presents a SOAP request scheduling algorithm for differentiated quality of service. The scheduling algorithm can be deployed on a Web services server or any server that processes SOAP requests. Due to the resource-intensive security processing of SOAP messages, this research implements the scheduling algorithm on a QoS security server. The security server schedules the requests forwarded from the Web services server for the security processing and then sends the valid requests back to the Web services server for executing the Web services. The design of the scheduling algorithm is derived from the traditional deficit round-robin scheduling. However, the scheduling algorithm schedules requests according to the probed CPU resource consumption of requests. In the evaluation, the scheduling algorithm reveals the service differentiation on the throughput and response time and the little scheduling overhead. The resource utilizations are measured to prove the security processing is much more resource-intensive than the Web services execution.

1 Introduction

Web services are self-describing and modular business applications that expose the business logic as services over the Internet through programmable interfaces and standard Internet protocols. A Web services can be invoked by different service requesters; thus, a service provider may wish to offer different Service Level Agreements (SLAs) to different consumers to guarantee different levels of Quality of Service (QoS) [1]. The QoS issues of Web services can be discussed from two perspectives: service consumers and service providers. From the service consumer perspective, a Web services potentially could be provided by many service providers with different SLAs. A service consumer can invoke one or more Web services to accomplish a task after the discovery of the Web service. Many researches have presented QoS brokers and middlewares between service providers and consumers for service selection and composition [2][3][4][5][6]. However, these help a service provider little to guarantee the service levels described in the SLAs. From the service provider perspective, requests for a Web services should be controlled in order to meet the guarantees in the negotiated SLAs. Some researches have proposed request scheduling and resource allocation algorithms to allow a service provider to provide service differentiation to multiple classes of service consumers [7][8][9][10]. Through prioritizing a request or estimating the resource requirement of a request, the throughput or response time can be differentiated among service classes.

In this paper, a SOAP (Simple Object Access Protocol) request scheduling algorithm that manages the system resource for differentiated quality of service is presented. The scheduling algorithm can be deployed on a Web services server or any server that processes SOAP requests. This research chooses to implement the scheduling algorithm on a security server because the security processing of SOAP messages, such as message integrity, message confidentiality, and message authentication, often consume more system resources than the Web services execution. The security server accepts SOAP requests forwarded from the Web services server. Then the request scheduling algorithm schedules the requests to determine the order and time of forwarding requests to the security processing. The Deficit Round Robin (DRR) scheduling [11] is emulated by the scheduling algorithm. However, the presented scheduling algorithm differs from the traditional DRR scheduling in that the former schedules requests but the latter schedules packets. The DRR scheduling requires packet size to be known, whereas the presented scheduling algorithm requires the amount of the server resource consumed by a request to be known. Another difference is that the traditional DRR scheduling is work-conservative, it never idles a link if it has a packet. The presented scheduling algorithm is also work-conservative in order to keep the server busy. However, it is non-work-conservative because it chooses to remain idle when there is no enough server resource, even if it has requests to service.

The presented QoS security server is implemented using the open source packages of the Apache XML project [12]. The security server has a request thread pool and a security thread pool for accepting requests from the Web services server and performing the tasks of the security processing, respectively. In the evaluation, the throughput and response time of each service class are measured to demonstrate the effectiveness of the service differentiation. The CPU and memory resource utilizations of the Web services server and security server are measured during the evaluation to prove the security processing actually consume much more resource than the Web services execution.

The rest of this paper is organized as follows. Section 2 introduces the related work regarding the Web services differentiation. Section 3 presents the architecture of the QoS security server and the design of the request scheduling algorithm. Section 4 describes the implementation and the evaluation of the presented solution. Section 5 finally gives the conclusion and the future work of this research.

2 Related Work of Web Services Differentiation

If a service provider wants to offer differentiated levels of services to multiple classes of service consumers, the SOAP requests destined to a server should be controlled for service differentiation. The general way to achieve this is to deploy a QoS broker or middleware in front of a server to determine the number, order, or time of requests to be forwarded to the server. The following introduces some researches related to Web service differentiation.

A QoS architecture consisting of a broker and proxies to map the QoS requirements from higher layers onto the underlying network layer has been presented in [9]. The proxies mark priorities of requests and responses in the IP packets and let underlying transporting technologies control the QoS. Although this solution can close the gap between the Web services layer and network layer, the prerequisite of success is the underlying transport technology must support QoS. However, the fact is QoS technologies are not widely deployed in the practical networks.

A smartware for according scheduling priorities to requests has been presented in [7]. Its scheduling algorithm adjusts the priorities accorded to requests dynamically to maintain the ratio of the request throughputs, measured by the number of requests per second, with respect to the incoming request traffic. A lower than normal arrival of a request category will penalize the priority, while the greater than expected arrivals will reinforce the priority positively. A concern of this solution is that the overhead of encoding service priority levels in the request header is high. This would affect the performance of the middleware.

A QoS broker that manages the server resource to be allocated to requests has been presented in [10]. The broker employs two resource allocation algorithms, homogenous resource allocation (HQ) and non-homogenous resource allocation(RQ), for legacy and QoS servers respectively. The HQ algorithm sets many threshold points and a step size and calculates the amount of the server resource to be allocated to a client. The RQ algorithm allocates different amounts of resources to different clients according to their requirements. It creates a virtual client to reserve some unused resource to reduce instability. If the reserved resource is not enough, it reconfigures the resource allocation among some existing clients to let the incoming client receive a satisfactory service quality. The purposes of the algorithms is to achieve a high average system utility and avoid making frequent resource reconfigurations. Nevertheless, the nature of the resource reconfiguration is not good for the scheduling. The scheduling algorithms should accurately allocate needed resource to a request rather than correcting the allocation after a reconfiguration.

An architecture and prototype implementation of a performance management for cluster-based Web services has been presented in [8]. The management tasks of the system include resource allocation, load balancing, and server overload protection. Its global resource manager periodically computes the number of concurrent requests to be sent to a server. It uses a simple queuing model to predict the response time of request for different resource allocation values. However, the omitted fact of this research is that different types of requests bring different amounts of resource consumptions. Only counting the number of concurrent requests processed on a server would lead to a bottlenecked or a non-fully-utilized resource.

3 QoS Security Server and Request Scheduling Algorithm

Web services security offers message integrity, message confidentiality, and message authentication in SOAP communications. All XML Web services security functions, such as XML schema validation, XML encryption, XML signature, Web services security and others, require extensive XML processing. If all these security functions are executed on a Web services server, the performance of the Web services would be downgraded seriously. Therefore, the secure XML processing should be offloaded from the Web services server. Here a QoS security server and a SOAP request scheduling algorithm are presented to offload the security processing of SOAP messages and provide service differentiation to Web services consumers.

The architecture of the QoS security server is shown in Fig. 1. The Web services server accepts requests from the Internet. The requests are then forwarded to the security server for the security processing. The request classifier classifies the requests into different service classes according to the pre-defined QoS policies and puts the requests into the corresponding class queues. The request scheduler checks the availability of the CPU resource. If the available resource is enough for the requirement of a request, the request scheduler fetches the request from the queue and determines the order and time the request being sent to the security processor. The security processor performs the security processing for the request. The valid requests will be sent back to the Web services server for executing the Web services, whereas the invalid ones will be dropped. The Web services server returns the response to the requester. The detailed design of the QoS security server is discussed as follows.

Request Profiling. A service provider may assign different security levels to different requesters. Hence, securing SOAP requests could go through different steps. XML schema validation, XML encryption, and XML signature are possible steps. A request could go through one step only, whereas another request could go through two or more steps. This means different requests would lead to different resource consumptions when being processed. In order to manage

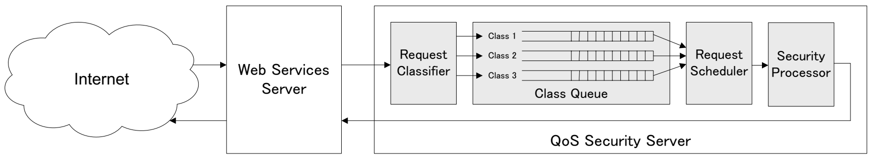


Fig. 1. Architecture of the QoS security server

the resource of the security server for the service differentiation, the resource consumption of the security processing of a request has to be known. The request profiling is a process to profile the resource requirement of a request. The resource consumption of every security function is measured. The amount of the resource requirement of a request is derived from summing all the resource consumptions of the needed security functions. The profiled information is stored in a resource requirement table. The request classifier refers to this table when estimating the resource requirement for a request.

Request Classification. The SLAs of the service provider and its clients define the service treatments the service provider should provide. The service provider therefore defines QoS policies for the request classifier to classify clients into different service classes. The QoS policies are defined in a QoS policy table that describes the rules of classifying requests and the service weights of the service classes. The request classifier accepts a request from the Web services server and inspects the HTTP header and the metadata contained in the SOAP request, such as user id, subscriber id and service name, etc. The header information and metadata are compared with the rules in the QoS policy table. If matched, the request will be classified into a service class; otherwise, it will be dropped. Once a request is classified, the request classifier lookups the resource requirement of the request from the resource requirement table. The request classifier then tags the resource requirement onto the request and puts the request into an appropriate class queue. The requests in the classes queue will wait for being scheduled to the security processor.

Request Scheduling. The key idea of designing the request scheduler is derived from the Deficit Round Robin (DRR) scheduling. A traditional DRR scheduler schedules packets to manage the bandwidth of a link. Whereas in this research, the request scheduler schedules SOAP requests to manage the resource of the security server. The request scheduler determines which request to be fetched next from the class queues and when to forward a request to the security processor for the security processing.

The operation of the request scheduler is shown in Fig. 2. The numbers in the blocks in the queues represent the amounts of resource requirements of the queued requests. The request scheduler uses a deficit counter to record the unused service quantum of a class and a round-robin pointer to point to the class queue to be serviced. It services the request at the head of each non-empty class queue which the value of the deficit counter is greater than the resource requirement of the request. In addition, the request scheduler checks the amount of the available resource for deciding whether to forward a request to the security processor or not. If the available resource is enough for the requirement of the request to be serviced, the request scheduler forwards the request to the security processor; otherwise, the request scheduler stops the scheduling and waits for the resource released from the finish of the security processing of a request. A deficit counter is decremented by the resource requirement of a request being serviced. When the value of the deficit counter is lower than the resource requirement of

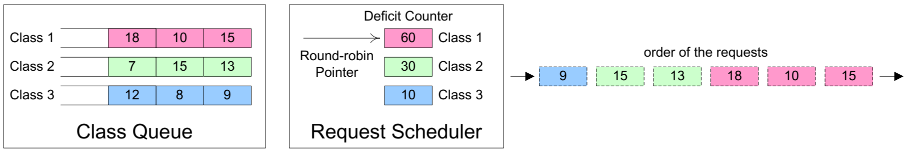


Fig. 2. Operation of the request scheduler

the request that at the head of the queue, this means the service quantum in this round is already not enough for the requirement of this class. The round-robin pointer at this time moves to the next queue and the next deficit counter is incremented by the defined quantum size. Through the request scheduling, the resource of the security server is shared among the classes according to the ratio of the quantum sizes assigned to the classes.

The presented request scheduler and the traditional DRR scheduler are different in several aspects. First, the presented scheduler schedules requests instead of packets. Then, the presented scheduler schedules requests according to the resource requirement of a request, not packet size. Finally, the presented scheduler is work-conservative to the security server; that is, it keeps the security server busy at any time. On the other hand, the presented scheduler is non-work-conservative to the class queues because it may choose to remain idle if there is no enough resource. However, the traditional DRR scheduler is work-conservative. The DRR scheduler keeps scheduling packets if there is any packet in the queues.

4 Implementation and Evaluation

4.1 Implementation

The implementation of the QoS security server is based on the open source packages of the Apache XML project, including XML Security, Xerces, Xalan, Log4j, Jakarta Discovery, and Jakarta HttpClient. The security server has two thread pools, the request thread pool and security thread pool. The threads in the request thread pool perform the tasks of accepting requests from the Web services server; whereas those in the security thread pool perform the tasks of the security processing. Each thread pool is given some fixed number of threads to use. The request thread pool size is larger than the security thread pool size so as to make the class queues accumulate enough requests for the request scheduling. The purpose of using the thread pools is to increase the performance through multi-threading and avoid too much thread switching overhead. When the Web services server forwards a request to the security server, the request thread pool assigns the request to one of its threads. The request classifier classifies the request and puts the request into an appropriate class queue. The request thread is released to accept a new request. Similarly, the security thread pool assigns a security thread to perform the security processing for a scheduled request. Once the security processing of the request is finished, the security thread is released for an upcoming request.

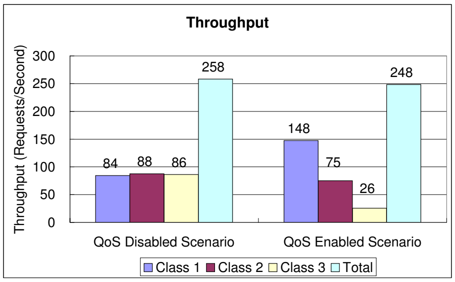
4.2 Evaluation

Evaluation Environment. The evaluation environment consists of three SOAP request generators, a Web services server, a database server, and a QoS security server. The platform of each generator and server is an Intel Pentium 4 2GHz system with 2GBytes of main memory and a 100 Mbps Ethernet network adaptor. Each request generator emulates 50 service requesters to sends a large amount of requests to the Web services server. Therefore the total number of the emulated service requesters is 150. Each request generator sends a new request to the Web services server after the generator has received a response. More request generators put more load on the Web services server. The Web services server forwards the requests to the security server for the security processing and service differentiation. The valid requests are sent back to the Web services server to execute the Web services, whereas the invalid ones are dropped. The Web services server submits queries to the database server and response to the service requesters.

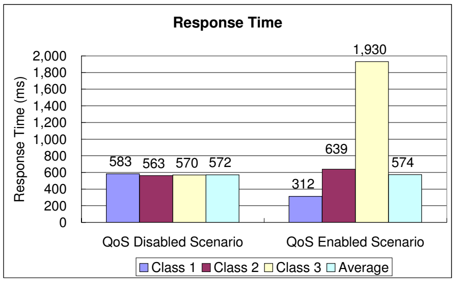
In the evaluation, the QoS disabled and QoS enabled scenarios are compared. The QoS disabled scenario is that the security server processes the requests immediately without any request scheduling mechanism, whereas the QoS enabled scenario is that the security server performs the presented request scheduling algorithm to manage the CPU resource. In the QoS enabled scenario, three services classes are defined and the ratio of the quantum sizes is set to 6:3:1. The throughput, response time, resource utilization of the Web services server and security server of the both scenarios are recorded for analyzing the results.

Service Differentiation. The effectiveness of the service differentiation can be observed mainly from the throughput of the Web services server and the average processing time of requests. Fig. 3(a) shows the throughputs in the QoS disabled and QoS enabled scenarios. In the QoS disabled scenario, it is obvious that there is no service differentiation among the three classes because the throughput of each class is almost the same, i.e. 86 requests per second. The throughputs of the three classes in the QoS enabled scenarios are 148, 75, and 26 requests per second respectively, and the ratio of the throughputs is 5.75:2.92:1, very close to the defined ratio of 6:3:1. The QoS enabled scenario demonstrates the effectiveness of the service differentiation. On the other hand, the total throughput in the QoS enabled scenario (248 requests per second) is a little bit lower than that in the QoS disabled scenario (258 requests per second). This reveals the overhead of the presented scheduling algorithm is very little.

Another result that can demonstrate the effectiveness of the service differentiation is the response time of processing a request. The response time is the time interval between the time a client issues a request and the time the client finishes receiving a response. The duration of the response time may contain the transmission delay, classification delay, queuing delay, and security processing delay, and Web services execution delay. Fig. 3(b) shows the response times in the QoS disabled and QoS enabled scenarios. In the QoS disabled scenario,

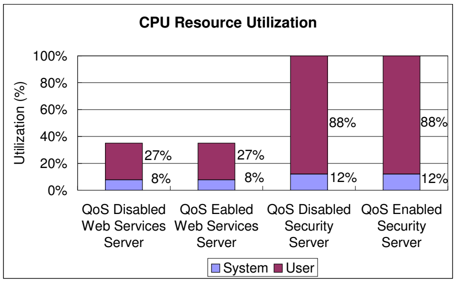


(a) Throughput.

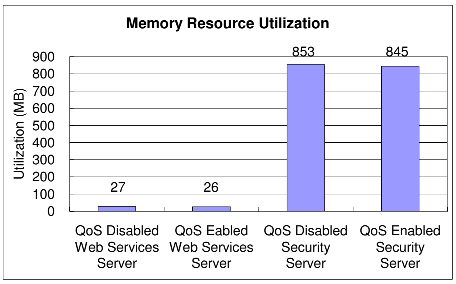


(b) Response time.

Fig. 3. Throughput and response time in the QoS disabled and QoS enabled scenarios



(a) CPU resource utilizations.



(b) Memory resource utilizations.

Fig. 4. Resource utilizations in the QoS disabled and QoS enabled scenarios

the three classes of clients perceive about 572 ms of response time. There is no service differentiation on response time in the QoS disabled scenario. However in the QoS enabled scenario, the three classes of clients perceive 312, 639, and 1930 ms of response time respectively. The response time in the QoS enabled scenario is differentiated among the three classes, even through the ratio of the response times of the three classes is not 6:3:1. Comparing the average response times of the three classes in the QoS enabled scenraio, the response time of the class 1 is rewarded, whereas the response times of the class 2 and class 3 are penalized. The average response time in the QoS enabled scenario (574 ms) is almost the same as that in the QoS disabled scenario (572 ms). This tells the presented scheduling algorithm only make the response time a little bit longer.

Resource Utilization. The need of offloading the security processing from the Web services server is based on the prerequisite that the security processing actually consumes more resource than the Web services execution. Fig. 4 compares the CPU and memory resource utilizations of the Web services server and security server. The majority of the CPU resource utilization is spent in the user space due to the Web services execution and security processing. Both the CPU and memory resource utilizations reveal the security processing is very resource intensive. An interesting finding is that there is no obvious difference on the resource utilization between the QoS disabled and QoS enabled scenarios. However, the QoS enabled scenario has demonstrated its effectiveness of the service differentiation and the little overhead.

5 Conclusion and Future Work

Quality of service of Web services allows a service provider to offer different service level agreements to different consumers. This paper presents a SOAP request scheduling algorithm to manage the system resource for differentiated QoS. The presented scheduling algorithm can be deployed on a Web services server or any server that processes SOAP requests. This research chooses to implement the scheduling algorithm on a QoS security server to offload the security processing of SOAP messages from the Web services server and provide service differentiation to Web services consumers. The security server schedules the requests forwarded from the Web services server for the security processing and then sends the valid requests back to the Web services server for executing the Web services. The design of the scheduling algorithm is derived from the traditional deficit round-robin scheduling. However, it schedules requests according to the probed CPU resource consumption of requests.

The QoS security server is implemented based on the open source packages of the Apache XML project. It uses a request thread pool and a security thread pool to perform the tasks of accepting requests from the Web services server and security processing, respectively. The request thread pool size is larger than the security thread pool size so as to make the class queues accumulate enough requests for the request scheduling. In the evaluation, the presented scheduling algorithm reveals the service differentiation on the throughput and response time and the little scheduling overhead. The CPU and memory resources of the Web services server and security server are measured to prove that the security processing is much more resource-intensive than the Web services execution and it is necessary to offload the security processing from the Web services server.

The presented scheduling algorithm only manages one server resource for the service differentiation. Actually, the requests accessing Web services may consume multiple server resources. Hence, a future direction of this research is on multiple-resource request scheduling. The scheduling algorithm should be capable of managing multiple resources to maximum the resource utilizations and at the same time provide service differentiation. Another future direction is enforcing QoS on compositive Web services. The execution of a Web service may rely on executing several related Web services or tasks on different servers. Therefore, the scheduling algorithm should keep a Web services differentiated when a request is processed sequentially or parallelly on several servers.

References

1. Schmietendorf, A., Dumke, R., Reitz, D.: SLA Management - Challenges in the Context of Web-Service-Based Infrastructures, Proceedings of the 2004 IEEE International Conference on Web Services (2004) 606–613
2. Zeng, L., Benatallah, B., Dumas, M.: Quality Driven Web Services Composition, Proceedings of the 12th International Conference on World Wide Web (2003) 411–421
3. Zeng, L., Benatallah, B., et al.: QoS-Aware Middleware for Web Services Composition, IEEE Transaction of Software Engineering, Vol. 30, No. 5 (2004) 311–327
4. Liu, Y., Ngu, A. H. H., Zeng, L.: QoS Computation and Policing in Dynamic Web Service Selection, Proceedings of the 13th international World Wide Web Conference (2004) 66–73
5. Yu, T., Lin, K. J.: Service Selection Algorithms for Web Services with End-to-End QoS Constraints, Proceedings of the 2004 IEEE International Conference on E-Commerce Technology (2004) 129–136
6. Maximilien, E. M., Singh, M. P.: Toward Autonomic Web Services Trust and Selection, Proceedings of the 2nd International Conference on Service Oriented Computing (2004) 212-221
7. Sharma, A., Adarkar, H., Sengupta, S.: Managing QoS through Prioritization in Web Services, Proceedings of the 4th International conference on Web Information Systems Engineering Workshops (2003) 140–148
8. Levy, R., Nagarajarao, J., Pacifici, G., Spreitzer, M., Tantawi, A. , Youssef, A.: Performance Management for Cluster Based Web Services, Proceedings of the 8th International Symposium on Integrated Network Management (2003)
9. Tian, M., Gramm, A., Naumowicz, T., Ritter, H., Schiller, J.: A Concept for QoS Integration in Web Services, Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (2003) 149–155
10. Yu, T., Lin, K. J.: The Design of QoS Broker Algorithms for QoS-Capable Web Services, Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (2004) 17–24
11. Shreedhar, M., Varghese, G.: Efficient Fair Queuing Using Deficit Round-Robin, IEEE/ACM Transaction on Networking, Vol. 4, Issue 3 (1996) 375–385
12. Apache XML Project, http://xml.apache.org/