Performance Analysis of Web Service Replica Selection in an Extranet

Partheeban Chandrasekaran[1], Shikharesh Majumdar[1], Chung-Horng Lung[1] and Laura Serghi[2]
[1]Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa. [2]Alcatel, Ottawa.
partheeban@yahoo.com, majumdar@sce.carleton.ca, chlung@sce.carleton.ca, laura.serghi@alcatel-lucent.com

Abstract

Providing web service replicas improves the overall system performance and redundancy for hardware failures. In Business-to-Business, this may be particularly interesting for organizations in an extranet. In our previous work, we proposed a broker to perform the web service replica selection. Furthermore we investigated a replica selection algorithm called minU and performed a preliminary analysis comparing random replica selection algorithm in a simple 4-node ring topology. In this paper we present a detailed performance analysis of minU and the factors affecting its performance under a larger 11-node network. The factors are weight given to most recent utility value (p), weight given to communication delay (a) and replica placement to name a few. Results show that the response time achieved using the minU replica selection policy greatly depends on these factors and also on the topology used. For instance, as the weight given to the most recent utility value is increased, the response time of the web services decreased. In our simulation experiments, choosing a to be 0.3 results in minimum response time.

1. Introduction

Web Services have become a common standard for complex software systems to interact with each other in a distributed network. Due to their ability to provide platform and language independence, they are being increasingly used in providing Business-to-Business (B2B) transactions. An extranet takes advantage of web services to expose a part of a company's intranet to its business partners. These business partners are authorized users of the extranet and can thus access any available web service of the participating company. In order to increase availability and to reduce the response time for the deployed web services, a company often decides to replicate key web services. This introduces the need for web service replica selection in the extranet.

In our previous work [1], we introduced a web service replica selection framework that can be used in an extranet. The main component of this framework is a broker that is responsible for replica selection. This broker can be placed in an extranet gateway which lies in between a web service provider (WSP) and a web service requester (WSR). The broker is also responsible for web service discovery which decreases the workload at the WSR. An extranet gateway is a regular gateway which has the capability of performing web service replica selection. The workload of the WSP and the communication delay between the WSR and WSP are the key parameters used in replica selections.

Using a replica selection strategy called minU, the broker first polls the WSPs for their load information and then determines the communication delay by calculating the round trip time from the WSP to itself. It then calculates the utility values (see Section 3) of each WSP and chooses the one with the minimum utility value. The performance of minU was compared with that of Random policy which randomly selects a replica. The simulation results based on our preliminary study using simple parameters on a small 4-node topology showed that the minU replica selection strategy performs better under high load situations. This shows that it is important to use the knowledge of load information of the WSP and communication delay while selecting a replica in an extranet.

In this paper, we present a detailed performance analysis of the framework proposed in [1] by simulating a larger topology to represent a more realistic extranet topology with 11 nodes. In addition, several factors affecting the minU replica selection policy are investigated in detail. Should our framework be deployed in a system, it is important to study the effect of these factors. This paper helps one to understand the minU replica selection strategy and its performance in a sample topology.

The rest of the paper is as follows: Section 2 provides a background on the web service replica selection and the framework we introduced in [1] and its deployment in an extranet. Section 3 explains the working of the minU replica selection strategy. The simulation model we used in our experiments is described in Section 4. The results are summarized in Section 5. Finally, Section 6 concludes this paper.

2. Web Service Replica selection framework and its deployment

Since a web service-based extranet extends from the traditional web service architecture, it consists of WSPs, WSRs and a registry implemented using the Universal Description Discovery and Integration (UDDI) protocol. The WSP could reside in a server located inside a company's private network. The WSR is a client requesting services. It allows applications to find binding information such as the Web Service Description Language (WSDL) documents for web services [*2*]. These three components interact with each other. However this interaction needs to be modified to meet our requirements. Thus in an extranet, the traditional web service architecture is extended to accommodate the broker to perform replica selection. Figure 1 shows the components of this extended web service architecture. They are WSP, WSR, UDDI registry and a broker.

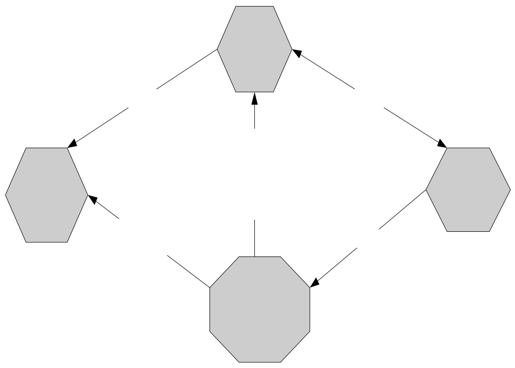


Figure 1: Extended Web Service Architecture [1].

In a typical private network, a gateway connects the WSP and the WSR to the outside world. Thus in this case, it is logical to put the broker in the gateway. In [3], the web service replica selection is done at the WSR. However this introduces extra overhead of the system-level or implementation modification at the WSR to incorporate the web service replica selection. Figure 2 shows a sample extranet architecture. Company A's site 1 consists of a gateway, a WSP and a WSR. The gateways can interconnect company A's other sites to this one and its extranet partner C as shown in Figure 2. One of the functionality of a gateway is that of a router. Using our framework, one can deploy the broker in the gateway to intercept all web service related packets.

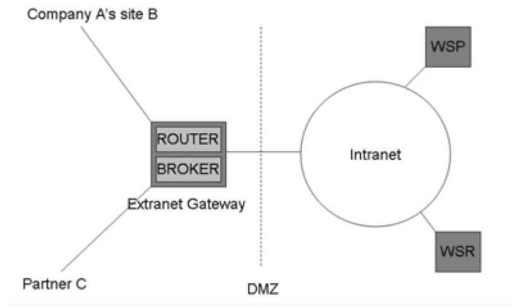


Figure 2: Sample extranet architecture [4].

The broker intercepts any web service requests from a WSR. It retrieves all the Uniform Resource Identifiers (URIs) of the WSPs providing that particular web service using the web service discovery policy. It then uses them to obtain their load information and the communication delays to the WSPs.

Figure 3 shows an extranet gateway with the broker module and its policies. The method of obtaining the load information is based on the information policy. Load information can be obtained by pull or push (i.e. polling or using a centralized load information repository). The broker then uses this information to estimate the load of the WSP and the communication delay using the load estimation policy. Then based on the implementation of the replica selection policy, the broker selects a web service replica and forwards the request to that WSP.

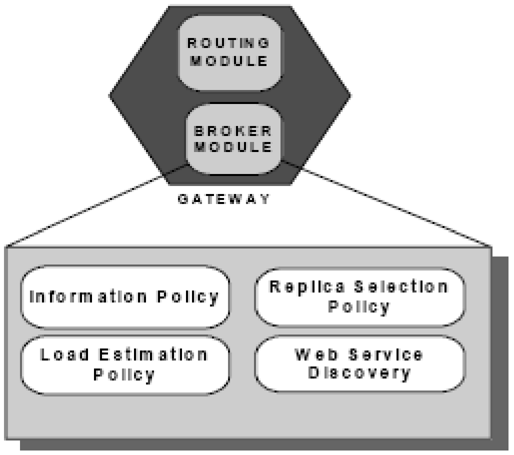


Figure 3: Broker and its plug-in components [1].

3. The minU replica selection strategy

The following sub sections summarize the policies used (see Figure 3) and the equations used in each policy by the minU strategy.

3.1 Web Service Discovery policy

To find out all the available replicas for a web service, the broker contacts the UDDI registry and obtains the URIs of the WSPs. These URIs are then used to obtain the load information of the WSPs as explained in the following section.

3.2 Information policy

The WSPs are polled to obtain their load information (*WSPLoad*). In our simulation, we have considered the number of requests queued for execution (*LQ*) multiplied by the average service time (*s*) for a web service request as the *WSPLoad*, as depicted in Eq. 1. On a real system, measurements need to be made to determine average

Web Service Provider

1. PUBLISH

5. IN

4. FORWARD REQUEST TO SELECTED

service time. In our experiments, the service time of a web service is normally distributed (explained further in Section 4). Thus the service time of each web service can vary. The average value of s is an input parameter in our simulation experiments. The *WSPLoad* is obtained by sending a load information request probe. When the probe returns, it contains *WSPLoad* and the broker calculates the round trip time which models the communication delay between the WSP and the broker (*CommDelay*). Once it has these values, it calculates a utility value (U) of the WSP by using Eq.2

$$WSPLoad = LQ * s \qquad (1)$$

$$U=a*WSPLoad + (1-a)*CommDelay \qquad (2)$$

a = weight given to the *WSPLoad* $(0 \leq a \leq 1)$
LQ = queue length of the WSP
s = average service time for a web service request

3.3 Load Estimation Policy

In [5], the authors investigate the importance of latency in obtaining load information in a distributed database environment. This latency may be caused by communication delays in receiving load information from a node. If the load information is obsolete, then choosing a node based on this information may result in an inefficient load sharing. Some nodes may be highly utilized while others may be idle. This results in load imbalance. One way to improve the load information accuracy is to increase the frequency of polling. Although this may solve the problem, it is often not feasible due to resource limitations of polling. To compensate for the effect of inaccurate input of load information, one can add an extra term to Eq.2 which is the age of the load information. Another way to estimate the load information given the fact these information may be obsolete is through exponential averaging or exponential moving average [6].

In this paper, we experiment with an exponential averaging approach to compensate for the effect of age of the load information which is obtained through polling.

In a typical polling scenario, the broker would wait for the probe to come back. This adds extra time to the overall response time of the web service. To mitigate the problem, exponential averaging was adapted. By using exponential averaging, the broker uses the previous utility values of the WSP to estimate the current value rather than polling the WSPs on demand. This is done using Eq.3. Exponential averaging is particularly efficient because it weighs old data according to its age. This can be seen in Eq.3 where *p* is called the smoothing constant. This equation shows that the old data should be given less weight while predicting the utility value (EU). However if the communication delays are very high and the requests are being generated in a non-uniform fashion, the effect of the age of the utility values may be too high. This makes exponential averaging inefficient. In this case, we can wait for the load information probe to come back and then make a replica selection.

$$EU(n) = p * U(n-1) + (1-p) * EU(n-1) \qquad (3)$$

EU = Estimated Utility value

3.4 Replica Selection Policy

Once the broker has estimated all the utility values of the WSPs, it chooses the one with minimum estimated utility value (EU) and forwards the request to the appropriate WSP.

4. Simulation model

By applying discrete event simulation techniques and using C++ as the programming language, we simulated the extranet components. The WSP was modelled as a queue with a very large queue size. The service time of a web service was normally distributed with a mean equal to 2000 and a standard deviation of 250 time units. The coefficient of variation for all the experiments was set to 0.125 except for Section 5.3. To investigate the effect of the communication delay in the replica selection, the ratio of service time of a WSP to the service time of a link should be significant. We chose this ratio to be 0.125. The links were modelled as an infinite queue with service time exponentially distributed with a mean of 250 time units. That is, each packet takes 250 time units on average to get serviced by a link. The links receive packets from a gateway at one end and transmit it to another gateway at the other end. The overall model is given in Figure 4. The WSR was implemented as a request generator and as a response receiver.

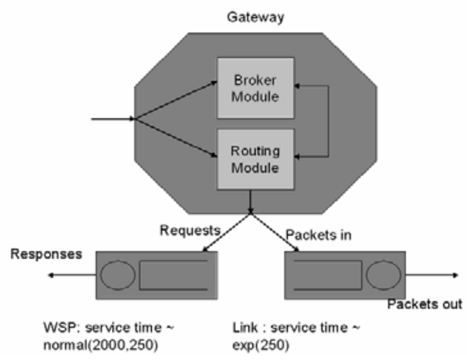


Figure 4: Simulation model of a gateway, a WSP and a link.

An 11-node with 25 links topology (see Figure 5) was simulated to investigate the performance of the minU replica selection strategy. Each node is considered as a company's site and it contains a gateway, a WSP and a WSR. Thus a total of 11 WSRs generate web service requests. The communication delay between a WSR and a WSP within the same intranet was considered negligible. The routing inside the topology was based on Shortest Path First algorithm [7].

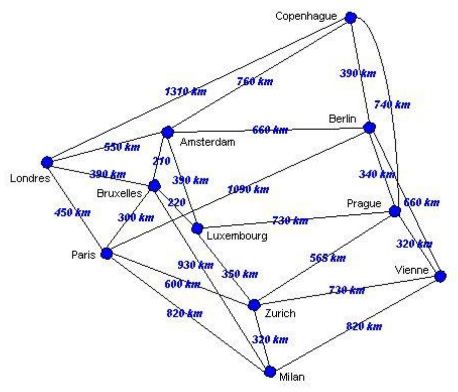


Figure 5: Topology used in the simulation [8].

5. Results

A number of simulation experiments were performed and the factors affecting the performance of minU and thus the framework are investigated. This section summarizes the results. Note that all the time metrics are expressed in terms of simulation clock ticks.

5.1 Effect of *p*

By increasing the value of *p* used in Eq. 3, we give more importance to the most recent utility value while estimating the current utility value of a WSP. When the value of *p* was varied from 0.1 to 0.9, we can investigate the impact of the weight given to the utility values in Eq. 3 on system performance.

Figure 6 shows the measured mean response time as a function of the mean inter-arrival time. As the mean inter-arrival time decreases, there is more contention for system and mean response time increases.

As parameter *p* is increased, the mean response time decreases. The impact of *p* on mean response time is more at low mean inter-arrival time. When mean inter-arrival time is 3000, there is a sharp decrease in mean response time when *p* increases from 0.1 to 0.3. Further increasing *p* produces a smaller improvement in performance. Results captured in Figure 6 show that *p* has a strong impact on the performance. Moreover a large value of *p* needs to be used for achieving a small response time.

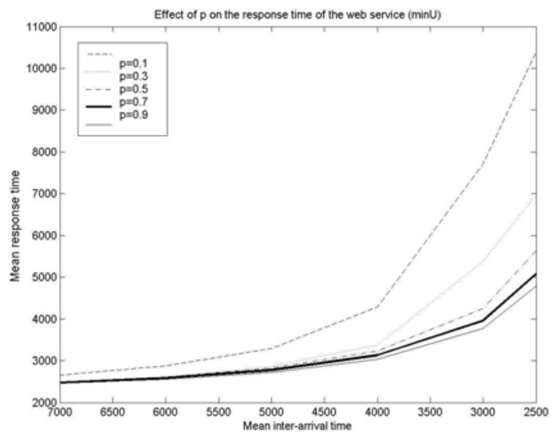


Figure 6: Effect of *p* on the mean response time

5.2 Effect of *a*

By changing the value of *a* in Eq. 2, we can investigate which factor among the *WSPLoad* and the *CommDelay* should be considered more in selecting a web service replica.

In this paper, due to the complexity of the system we did not derive a mathematical function to obtain the response time in terms of parameter *a*. However we assumed that the response time is indeed a function of *a* and used a trial and error method. Parameter *a* was varied from 0.1 to 0.9 to obtain a value close to the best value of *a*.

Figure 7 shows that the mean response time as a function of the mean inter-arrival time of the requests. Note that the units of axes in this graph and in the rest are in simulation clock ticks. The Each curve in the figure was obtained for different values of *a*. As it can be seen from the graph, the best value of *a* is 0.3. This means that more weight needs to be given to the communication delay while using Eq.2. This is because with the workload experimented with, the communication delays have a much higher impact on the response times of the web services compared to the load at the WSPs. Note that this was observed when the link service time was exponentially distributed with an average of 250 time units (*exp(250)*) and a routing algorithm based on SPF (shortest path first) was used. Thus, for a topology with different parameters, the best value of *a* can be different than what is reported in this paper.

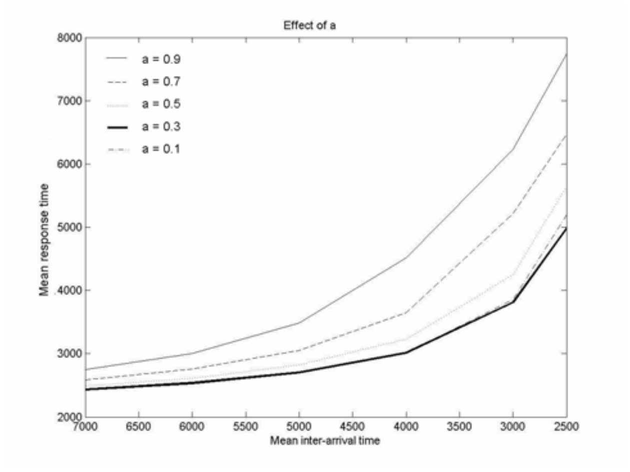


Figure 7: Effect of *a* on the mean response time

5.3 Coefficient of Variance of the web service

The coefficient of variation (CoV) of a web service allows us to investigate the randomness in the requirement of the execution time of a web service.

As this ratio increases with the mean service time kept constant, a small number of very large requests and a large number of smaller requests are generated. These large requests introduce large queuing delays for the smaller requests. Overall, this leads to significant delays for the web service requests. As a result, the mean response time of the web services increase. Figure 8 shows that as the coefficient of variation is increased from 0.125 to 0.5, the mean response time increases. The difference in mean response time when CoV is 0.5 and CoV is 0.25 is higher than when CoV is 0.25 and CoV is 0.125. This suggests that to achieve minimum response time, the value of CoV should be kept below 0.25.

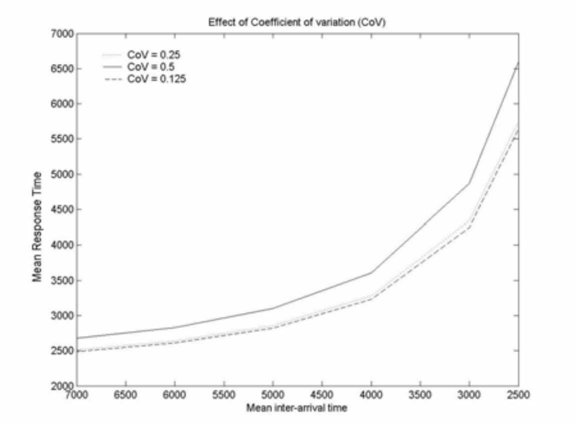


Figure 8: Effect of Coefficient of variation (CoV)

5.4 Multiple web service classes

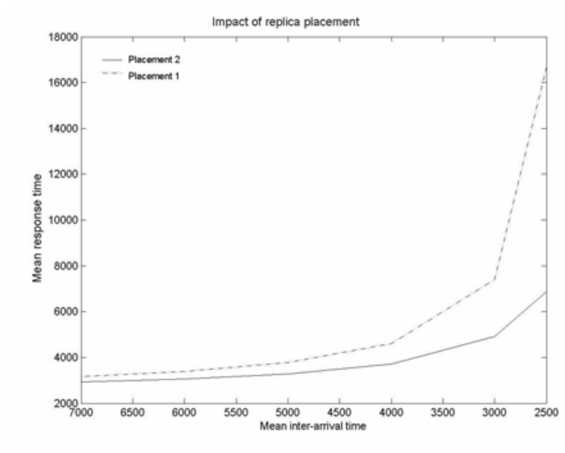
So far we considered all web services with same execution time. In reality, different web services may require different execution times. To investigate such systems, we experimented with three classes of web services. Web services in the same class are statistically identical. The CoV is kept constant at 0.125 for the three classes. The mean service time of a web service in class I is less than that of class II and the mean service time of a web service in class II is less than that of class III by the same difference. In our experiment, we chose this difference to be 1000. The probability of generating a request of any class is the same. This ensures that there is no bias towards any particular class. The parameters characterizing the three web service classes are presented in Table 1.

Table 1: Types of requests and their mean service time and standard deviation

Class	Resource requirement	Mean Service Time	Std Dev
I	LOW	1000	125
II	MEDIUM	2000	250
III	HIGH	3000	375

Figure 9 shows that for any given arrival rate the mean response time for the single class system is lower to that achieved with the multi-class system. Although the average mean service demand is the same in both the cases, the multi-class system workload inhibits a higher variability in service demands. This introduces a small number of very large requests which tend increase the queuing delays experienced by smaller requests and the mean response time deteriorates.

Figure 9: Impact of multiple class web services



5.5 Effect of replica placement

To see the impact of replica placement on the performance of minU, we considered two placements. Table 2 shows the number of replicas for each web service type across the nodes. For example in placement 1, web service 4 is replicated at 5 WSPs.

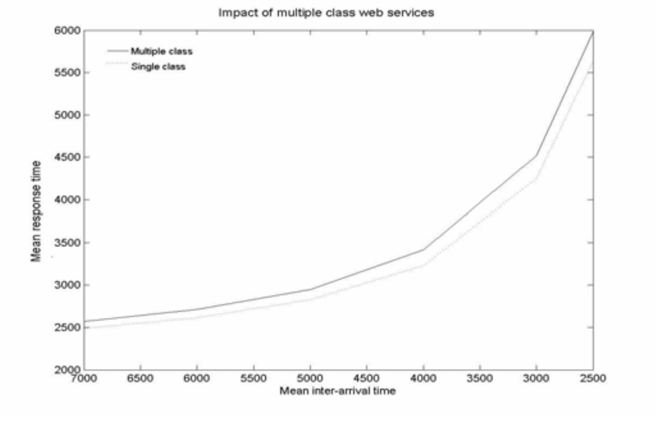


Table 2: Placement of replica

WS type	Placement 1	Placement 2
WS 1	6	7
WS 2	6	8
WS 3	6	6
WS 4	5	3
WS 5	5	4
WS 6	5	4

Figure 10 shows the response time observed for web services when the two placements were used. As seen from the graph, there is significant performance degradation when Placement 1 is changed to Placement 2.

In placement 2, the number of WSPs that can service WS 2 is 8 whereas for WS 4 only 3 WSPs can service this request. As a result, for WS 2 there are more resources available than for WS 4. We observed that for WSPs that can service WS 4, the utilizations were very high. These WSPs were in fact the bottlenecks in the system. This gave rise to a higher response time than in placement 1.

Figure 10: Impact of replica placement on the performance on minU

5.6 Impact of the number of the WSPs polled

Polling can be resource intensive in terms of communication link utilizations and WSP utilizations. One method to reduce the overhead is to poll a lesser number of WSPs for a particular web service request. In [9, 10], the authors investigate the impact of polling in a distributed system. In their research, the nodes are probed for load information and a request is sent to the least loaded node among these polled nodes. The authors in [10] claim that a small poll size could result in almost the same performance as could have been gained by probing all nodes. But in [9], a small poll size was found to perform better than larger poll size. In this paper, we study the overhead of polling in terms of link utilizations since a wide area network is used for extranet. To investigate the impact of the number of the WSPs polled, we used the default parameters but changed the percentage of the WSPs to be polled. Figure 11 shows the impact of the number of WSPs polled. At 10% of the total number of WSPs available, the minU replica selection policy behaves like a random replica selection policy in the sense that only one of 11 WSPs is polled. As the percentage of the WSPs polled is increased, the response time is observed to improve. This is contrary to what was reported in [9]. In our experiments, we assumed that the service time for a load information request in the WSP to be negligible in comparison with a WS service time. Thus there is no CPU cost involved due to polling the WSPs.

It is possible that as the number of nodes in the extranet increases, polling 90% of the nodes may yield to a worse performance than polling a lesser percentage of the nodes. In our future work, we plan to investigate the impact of polling overhead on the WSPs and on the system performance.

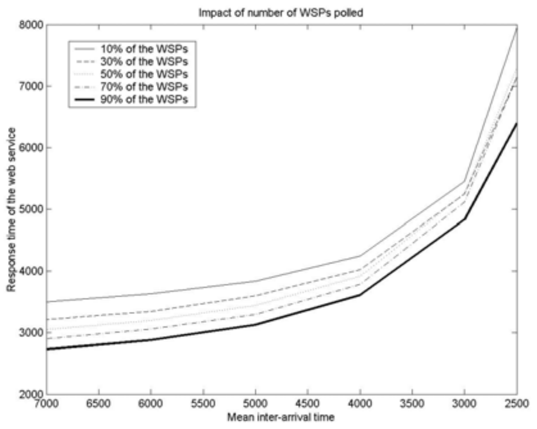


Figure 11: Impact of number of WSPs polled

Although a high system performance is desired, there are tradeoffs which need to be addressed. Aggressive polling may not be feasible due to resource limitations. The load information probes introduce queuing delays at the links and consume CPU resource in the WSP. Figure 12 shows that as the number of WSPs polled increases, the link utilizations increase. Thus for one to use our proposed framework, such resource limitations must be considered before setting up the parameters for number of WSPs polled.

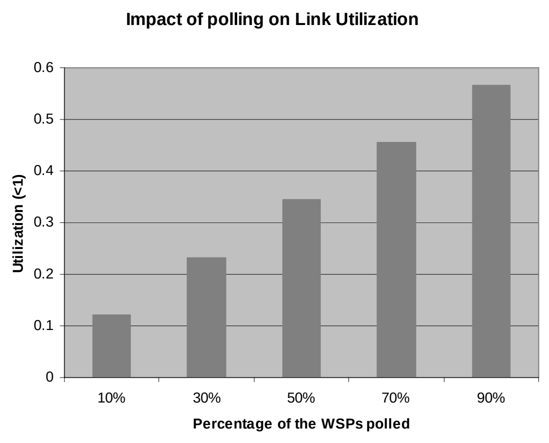


Figure 12: Impact of number of WSPs polled on the link utilizations

6. Conclusions

In a web service-based extranet a broker can be used to perform the replica selection. This broker is implemented as a software module in a gateway such that it intercepts all web service requests from a WSR. We devised a replica selection framework that is described in [1]. There are different parameters of the minU replica selection strategy that affect the system performance. In this paper we have described simulation experiments that were conducted to investigate the impact of these parameters on system performance. Insights gained from these simulation experiments are presented in the following paragraphs.

Parameter p is the weight given to the most recent utility value while trying to estimate the current utility value of a WSP. As p increases, the difference between the estimated and actual utility value decreases and thereby decreasing the response time of the web service requests.

Parameter a is the weight given to the WSP workload while $(1-a)$ is given to the communication delay. The best value of a is 0.3 based on our simulation environment and parameters. This implies that the communication delay in the links impacts the system performance more than the load of the WSPs.

In some systems, web services may not exhibit the same execution time. This affects the response time of the web service requests. To investigate this, the coefficient of variation was varied and the response time of the web services was measured. As the coefficient of variation was increased, the response time increased significantly.

Web services may have different requirements in terms of execution time. To investigate this impact on minU's performance, we considered three classes of web services with high, medium and low resource requirement. Simulation results show that the performance of minU was not affected significantly.

To study the impact of polling on the utilization of the communication links, we varied the number of WSPs polled and measured the link utilization. Results show that as the number of WSPs polled increases, the response time decreases; however, the link utilization increases. Thus a system architect needs to consider the resource constraints.

As our future work and to validate our simulation results, we plan to implement a prototype using our web service replica selection framework.

7. References

[1] K. Frounchi, P. Chandrasekaran, J. Ibrahimi, S. Majumdar, C.-H. Lung, and L. Serghi, "A QoS Aware Service Replica Selection Framework for an Extranet", *Proc. of IEEE Canadian Conference on Electrical and Computer Eng.,* Ottawa, May 2006, pp.1380-1384.

[2] UDDI Spec Technical Committee, "UDDI Version 3.0.2," October 2004, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm.

[3] J. A. F. da Silva, N. das Chagas Mendonca, "Dynamic Invocation of Replicated Web Services," in *Proceedings of WebMedia and LA-Web 2004*, Rebeirao Preto-SP, Brazil, October 2004, pp. 22-29.

[4] S. Prakash, "Extending the corporate intranet," *First IEEE Enterprise Networking Mini-Conference*, Montreal, June 1997, pp. 21-26.

[5] A. Leff and P. S. Yu, "A Performance Study of Robust Distributed Load Sharing Strategies", *IEEE Transactions on Parallel and Distributed Systems,* vol. 5, no. 12, December 1994, pp.1286-1301.

[6] A. Silberschatz, P. B. Galvin, G. Gagne, *Operating Systems Concepts*, John Wiley and Sons, Inc, USA, 2004.

[7] Shay, W. A, *Understanding Data Communications and Networks,* Thomson and Brooks/Cole, USA, 2004.

[8] Planification et Optimisation des Réseaux de Transport Optiques, l'INRIA, le CNRS and l'Université de Nice-Sophia Antipolis, "Dimensionnement de réseaux optiques," January 2006, http://www-sop.inria.fr/mascotte/porto/.

[9] K. Shen, T. Yang and L. Chu, "Cluster Load Balancing for Fine-grain Network Services", *Proc. of the 16th Int'l Parallel and Distributed Processing Symposium*, Florida, April 2002, pp.51-58.

[10] R. Muntz, J.R. Santos and S. Berson, “A Parallel Disk Storage System for Real-Time Multimedia Applications,” *Int’l Journal for Intelligent Systems,* vol. 13, no.12, 1998, pp. 1137-1174.