Web Service Pipelining

New Melchizedec, Sriram Krishnamoorthy

Vimal Kumar Vivekananthamoorthy, Arul Siromoney[1]

School of Computer Science and Engineering,

College of Engineering Guindy,

Anna University,

Chennai, India.

[1]asiro@vsnl.com

Abstract

Web servers provide information that can be accessed in the form of web services. Integrating these services often involves getting the result from one web service and sending it to the next. Traditionally, the client gets the result from one web service and sends it to the next. This paper proposes a model called Web Service Pipelining to improve the performance of web service access in such situations.

1 Introduction

Web servers provide information, which often need to be combined to produce the desired result. The user browses the web site, obtains the desired information from the servers and puts them together to get the desired result. The advent of Web Services helps

to remove the human element and allow an automated application to get desired information from various servers and process them to produce the result. Integrating these services often involves getting the result from one web service and sending it to the next. In most cases the output of one service is sent as input to another without any intermediate processing. Traditionally, the client gets the result from one web service and sends it to the next. This is illustrated in Fig1. This paper proposes a model called Web Service Pipelining to improve the performance of web service access in such situations.

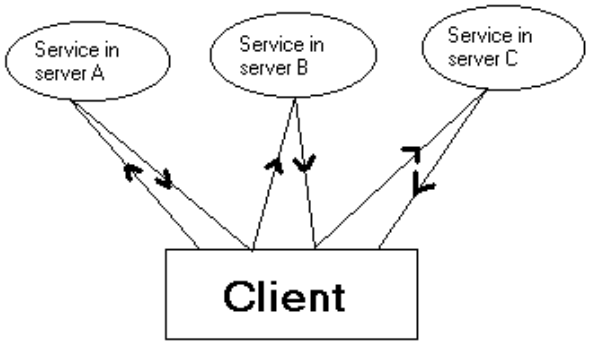


Fig 1. Traditional Web service access

2 Web service pipelining

In our proposed model, the client, instead of getting the result from every intermediate web service, requests the intermediate results to be forwarded from one web service to the next. Each server, which wishes to allow its services to be accessed in such a way, should run an additional service called the Pipe Service (PS). Two models of pipelining are proposed.

2.1 Recursive Access

Here the sequence of web services to be invoked, along with the required parameters and the relation between the result of one web service and parameters of the next and the address of the PS in the server containing the service, is packaged into a Call Sequence (CSeq) and sent to the PS in the server containing the last service to be invoked. The PS in this server determines whether all the parameters required to the service in this server are available. If not, it determines the service whose results are required and invokes the PS in the server containing it. The CSeq sent to the invoked PS consists of CSeq received by the invoking PS excluding the details corresponding the web service whose parameters are sought. On getting the result the PS invokes the web service and returns the result to the caller. The client only accesses one service, while most of the operations are done at the server side. This is especially advantageous when the size of intermediate results is large compared to the input provided by the client. In addition, contiguous services in the Cseq, residing in the same server, are called by the PS in the server without any data flow through the network The flow of requests and responses is as shown in following figure, the numbers representing the order in which they take place.

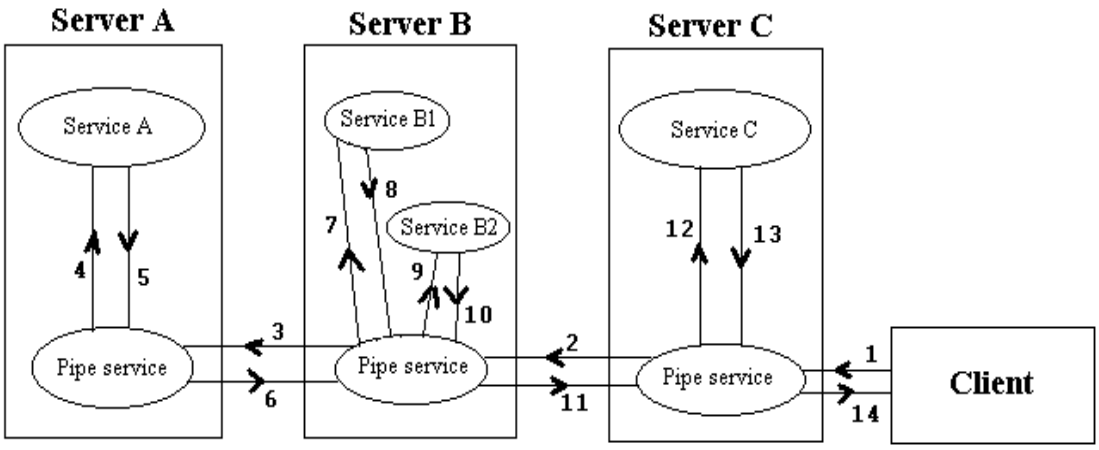


Fig 2. Recursive access

2.2 Linear Access

In this method, the client gives the CSeq to the PS in the server containing the first web service. The PS, which receives the CSeq, recognizes itself as the first pipe service (FPS) if the First Pipe Service Address (FPSA) field is not set in the CSeq and stores its address in the FPSA field. Then the first web service is called, its result obtained and stored in the parameters field of the next service in the CSeq and forwarded to the next PS specified in the CSeq. If the Next Pipe Service Address (NPSA) is not set in the CSeq, the PS identifies itself as a last pipe service (LPS) and sends the result to PS whose address is specified in the FPSA field. Each PS will iterate through the CSeq till the next service to be invoked is not in the same server as the PS.

Before sending the CSeq to the next PS, the FPS prepares an identifier for this CSeq (this instance of the pipeline) and stores it in the CSeq. This identifier should be unique within FPS. It is used by the LPS to return the result to FPS. The FPS sleeps until the result arrives. The LPS reports the result (or error) by awakening the FPS and gives it the result (or error). The instance of FPS to be awakened is determined by the unique ID in the CSeq obtained by the LPS. The client invokes only a single web service to get the result.

It is important to notice that the calls between PS's are asynchronous-they return immediately and do not wait for completion. The returning of result from the LPS to the FPS increases the communication one hop as compared to LPS returning the result directly to the client. But this provides a lot of advantages. Consider LPS returning the result directly to the client. The client will have to make two calls, one to the FPS and one

to the LPS. A unique identifier within the FPS is not sufficient anymore. The call to LPS might arrive before the CSeq has reached it, which would have to block this call until the Cseq arrives, or it might arrive after the request has been serviced in which case results needs to be cached until the client's request for result arrives. Both lead to wastage of server resources and increases complexity.

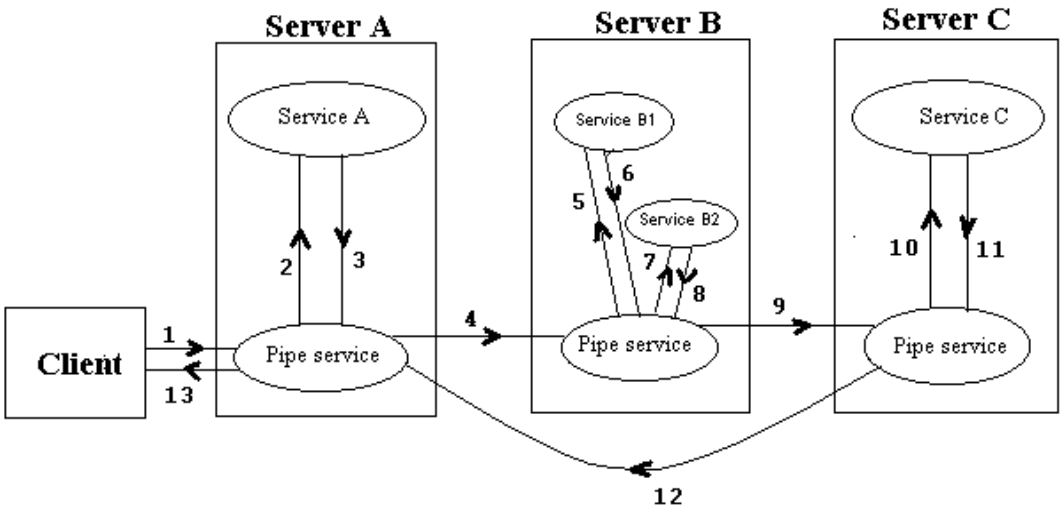
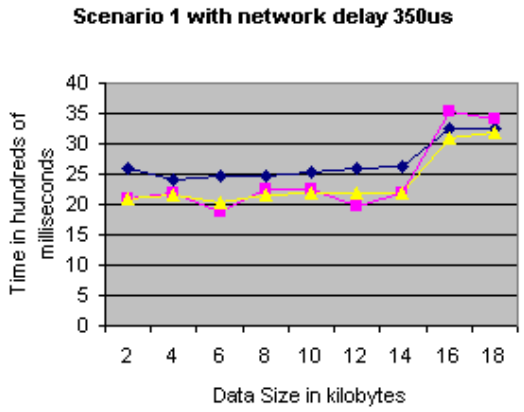


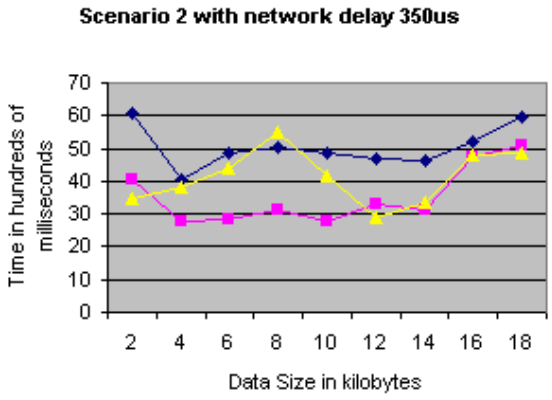
Fig 3. Linear access

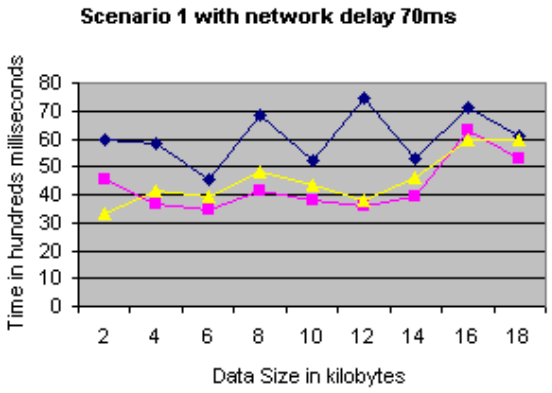
3 Simulation and Results

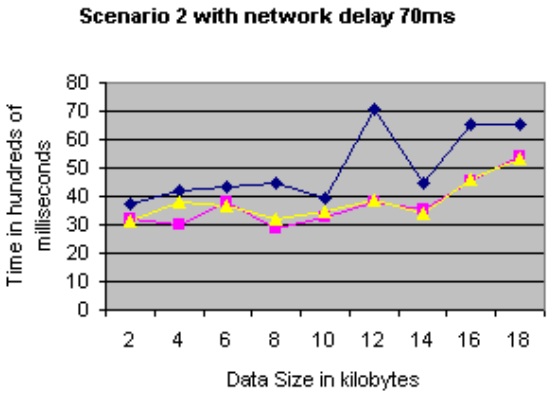
The simulation was done in our computer science department network Two machines were used as web servers with IIS server installed on them. The client was implemented on another machine. The web services and the clients were implemented using the .NET framework. SOAP protocol was used to access the web services. The implementation was tested under two scenarios. In the first scenario, the client sends a request with little data as parameter (an example would be a file name to get the contents of a file). In the second scenario the client sends a large amount of data as parameter (an example would be to

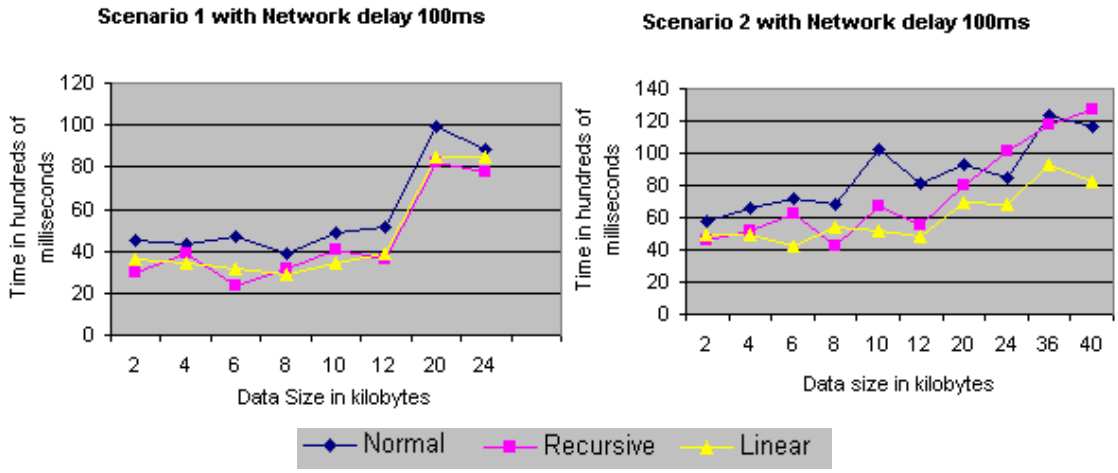
encrypt a file). Six web services on the two web servers were called in an alternate manner to test the performance. In the first scenario, the first service generates data of size given as parameter. The subsequent services do a transformation on the input and return a result of size proportional to the input size. In the second case all web services do a transformation on the input given by the client. The performance was measured in terms of time of execution of the normal, recursive and linear access models. The network load was varied, by flooding it using ping, and the execution times were measured. The results are depicted in the graphs below.











From the graphs it is clear recursive and linear access models perform better than the traditional model of web service access. The two alternate models perform differently in the two scenarios. Let n be the number of services to be accessed. In scenario 1 the client sends a small amount of data as parameters and the bulk of data travels (n+1) hops in case of linear access and n hops in case of recursive access. Thus recursive access is slightly better than linear access. In scenario 2 the client sends a sizeable amount of data as parameters and the data travels (n+2) hops in case of linear access and 2*n hops in case of recursive access. Thus linear access is better in this scenario.

4 Conclusion and Enhancements

In this paper alternative models for web service access were proposed and their performance analyzed. It was found that the models perform better than the traditional model of web service access. Accessing a sequence of services in the same server can be done very efficiently. Greater performance would be obtained if more number of web services are accessed as part of a pipeline. The provision of lightweight web services

(such as for addition, matrix multiplication, etc.) would enable simple processing of intermediate results, thus increasing the scope of web services being part of a pipeline. Further improvement would be obtained for clients with lower computing power and communication bandwidth.

5 References

1. SOAP 1.1 specification http://www.w3.org/TR/2000/NOTE-SOAP-20000508
2. WSDL 1.1 specification http://www.w3.org/TR/2001/NOTE-wsdl-20010315
3. Building Distributed Applications with .NET
   http://msdn.microsoft.com/library/default.asp?URL=/library/techart/Bdadotnetover.htm