Zahir Tari · Ann Khoi Anh Phan
Malith Jayasinghe · Vidura Gamini Abhaya

On the Performance of Web Services

Springer

On the Performance of Web Services

Zahir Tari • Ann Khoi Anh Phan • Malith Jayasinghe • Vidura Gamini Abhaya

On the Performance of Web Services

Springer

Zahir Tari
School of Computer Science and
Information Technology
RMIT University
Melbourne Victoria
Australia
zahir.tari@rmit.edu.au

Ann Khoi Anh Phan
Macquarie University
North Ryde New South Wales
Australia
ann.phan@mq.edu.au

Malith Jayasinghe
School of Computer Science and IT
RMIT University
Melbourne Victoria
Australia
mjayasin@cs.rmit.edu.au

Vidura Gamini Abhaya
School of Computer Science and IT
RMIT University
Melbourne Victoria
Australia
vidura.abhaya@rmit.edu.au

ISBN 978-1-4614-1929-7 e-ISBN 978-1-4614-1930-3
DOI 10.1007/978-1-4614-1930-3
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011940805

I dedicate this book to my lovely son Amazigh Tari who always keeps asking the question Why? and to my wife Astrid Blume-Tari who always was present during during difficult and good times of my life!
Zahir Tari

Contents

Acronyms

BEEP: Blocks Extensible Exchange Protocol
BXSA: Binary XML for Scientific Applications
CBM: Content-based Multicast
CDR: Common Data Representation
CORBA: Common Object Request Broker Architecture
DCOM: Distributed Component Object Model
DDS: Differential Deserialization System
DR: Designated Router
DVMRP: Distance Vector Multicast Routing Protocol
FCFS: First Come First Serve
FTP: File Transfer Protocol
FB: Foreground-Background
FXI: Financial Information eXchange
HHFR: Handheld Flexible Representation
HTML: Hypertext Markup Language
QoS: Quality of Service
LCFS: Last Come First Serve
LLF: List Loaded First
LRW: Least Remaining Work
MEP: Message Exchange Pattern
MLTP: Multi-Level Time sharing Policy
MLMS: Multi-level Multi-server Load Distribution Policy
MLMS-M: Multi-level Multi-server Load Distribution Policy with Task Migration
MLMS-WC-M: Multi- Level-Multi-Server Task Assignment Policy with Work-Conserving Migration
MOSPF: Multicast Extension to OSPF
MQ-DMR: QoS-based Dynamic Multicast Routing
MTOM: Message Transmission Optimisation Mechanism
MTTMELL: Multi-Tier Task Assignment with Minimum Excess Load
OSPF: Open Shortest Path First

PIM: Protocol Inde- pendent Multicast
PIM-SM: PIM-Sparse Mode
RPC: Remote Procedure Call
RMI: Remote Method Invocation
RR: Round Robin
SITA-E: Size Interval Task Assignment with Equal load
SMP: Similarity-based Multicast Protocol
SMPT Simple Mail Transfer Protocol
SOA: Service-Oriented Architecture
TAGS: Task Assignment by Guessing Size
TAGS-WC: TAGS with Work Conserving
TAPTF: Task Assignment based on Prioritising Traffic Flows
TAPTF-WC: TAPTF with Work Conserving
TCP: Transmission Control Protocol
tc-SMP: traffic-constrained SMP
UDDI: Universal Description, Discovery and Integration
UDP: User Datagram Protocol
WS: Web Services
WSDL: Web Service Description Language
WS-WRM: Web Service-Wireless Reliable Messaging
XOP: XML-binary Optimised Packaging
XML: eXtensible Markup Language

Chapter 1
Introduction

Over the past decade, we have seen phenomenal interest in the deployment of Web services in many enterprise applications. *Web services* are a new type of Web applications based on the *Simple Object Access Protocol* (SOAP) that allows interoperability between different platforms, systems and applications. There has been in the past such technologies, such as CORBA - Common Object Request Broker Architecture (139), which enabled interoperability, however they had quiet major limitations (e.g. the use of object model in the case of CORBA), which obvious have limited the scale of their use. Web services allows a text-based communication (i.e. Text based RPC), therefore not restricting communication and interoperability to specific models or patterns. Obviously there are much more advantages of Web services over other technologies, and these will be reviewed in the remaining parts of this book.

The development of Web services not only attracts interest from the research community but also from industry. Most large organisations in the software, information technology and telecommunication industries have been working closely with the World Wide Web consortium to develop Web service standards such as WS-Addressing, WS-Reliability, WS-Security and WS-Management. Many organisations have achieved some degree of success, such as generating new sources of revenue or streamlining their internal and external processes when deploying Web service technologies for their enterprise services (82, 83, 84, 146).

Web service technologies, such as SOAP, Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI), promise to provide seamless integration of services provided by different vendors in different industries and written on various platforms and languages. Examples of such services include travel booking, real-time stock quotes, currency exchange rates, credit card verification, driving directions and yellow pages. A popular example used to illustrate Web services is the travel reservation application. A travel agent company offers a complete vacation package (in a form of a workflow), which includes airline/train/bus tickets, hotel reservation, car rental and tours. This service involves many service providers such as airlines, transport companies, hotels, tour organisers and credit card companies for payment. The back-end of service applications

Z. Tari et al., *On the Performance of Web Services*, DOI.10.1007/978-1-4614-1930-3_1,

offered by these service providers are likely to be written in different programming languages (e.g. C++, Java, .Net) and implemented on different platforms (e.g. Linux, Macintosh and Windows). But if all of these services are implemented using Web service technologies and are published on a public registry such as the UDDI (114), the travel agency can search all services from the one platform. Consumers who want to book vacation packages can come to the travel agency's website and specify some criteria such as location, means of transport and price range for their travel. The travel agency will act on behalf of the consumer and search for appropriate services registered on the UDDI and return results that satisfy the user's requests.

Obviously the application explained above can be implemented using traditional distributed computing technologies such as Distributed Component Object Model (DCOM) (69) and Common Object Request Broker Architecture (CORBA) (139). However, such traditional approaches do not provide the same high level of interoperability that Web services do. In particular, if an application were to be developed using DCOM, all participating nodes in the distributed application would have to be running on Windows platform (55). CORBA is based on the object-oriented model and a binary transport, Internet Inter-ORB Protocol (IIOP) (88), hence an Object Request Broker (ORB) node assumes a certain representation exists in other nodes to allow them understand each other. SOAP endpoints are, on the other hand, not dependent on any specific data representation or platform, since all data is already formatted in a "text"-based language (called XML - *eXtensible Markup Language*).

With the exciting prospects of what Web service technologies can bring come many difficult challenges. Web services' major problem is that they generate a *very large* amount of network traffic. This comes from the fact that a given call for an operation of a Web service (from the client side) is translated (more precisely "serialised") in XML, which could be a large document that is sent through the wire (as a SOAP call) to the sever side (which in turns "de-serialise" the XML-based call into a native call, such a C++ or Java operation call). Therefore SOAP provides the basic messaging infrastructure for Web services by exchanging XML messages. It is XML's textual representation and redundant characteristics that cause the major performance bottleneck. Tian et al. in (140) performed extensive performance tests showing the number of additional bytes Web services generate. There were 589 bytes in both request and response messages for a service requesting the details of a book given an ISBN in the parameter of the request. But more than 3900 bytes had to be sent when using SOAP, while only 1200 bytes were sent when traditional Web interaction with HTML was used.

SOAP's overhead stems mainly from the use of XML. Since both SOAP and WSDL are XML-based, XML messages have to be parsed on both the client and the server side. XML parsing occurs at run time, therefore the required additional processing time results in longer total response time. There has been so many evaluations of SOAP performance made in the last decade. Here we only list of them for illustration only. Some of the limitations of SOAP for scientific computing were investigated by Davis & Parashar in (40). Their experiments compared SOAP with Java RMI by sending large arrays of doubles; the results showed that SOAP was slower than Java RMI by a factor of *ten*. Another experimental evaluation of SOAP

performance on business applications was done Kohlhoff & Steele in (73): SOAP was compared with Financial Information eXchange (FIX) protocol which also used a text based wire representation as SOAP, and with Common Data Representation (CDR), a common binary wire format. The results demonstrated that the text-based protocols (SOAP and FIX) have slightly lower performance than the binary protocol (CDR) due to the complexity of the XML syntax.

But the performance issues of Web services are not only related to SOAP! There are also major problems at the server side, when XML-requests are translated into native calls (C+/Java calls). Such problems have different nature and forms, depending on the configuration of the server. Simply, when the server receives a huge number of XML-requests (called *tasks*), this will not be able to process them as its queue/s will be full, and therefore the server will drop them. Overloaded servers are known problems in distributed systems and there has been a lot of research work done in the last two decades. *Load balancing* is a common approach used to improve performance: a server is replicated and tasks are dispatched to the "appropriate" server in a transparent way (i.e. the user is not aware of such replication). Load balancer, also called *dispatcher*, uses a specific policy, called *task assignment* policy, to decide how to allocate a given task to a given server so to make sure that this is processed in an efficient way (i.e. minimal waiting time in the server's queue). The earlier task assignment policies, called *static policies*, allocated tasks without relying on any specific information (e.g. load of a server, often measured as the # tasks in the queue). Random and Round Robin (RR) are typical static policies. It is well-known that such policies have serious limitations because of their static nature. More advanced policies were later proposed, called *dynamic policies*, which take into account dynamic information such as server load. LLF (Least Loaded First) is probably one of the well-known dynamic techniques, where tasks are allocated to the least loaded server. One may say ask the question about the way load is computed in a distributed environment like Web services. This is a good question though! Indeed measuring the load of a server, that is estimating the number of tasks in a queue, is well-known to be a very hard problem (i.e NP hard). Indeed, the computation of the *load index* can be complicated, as the load information computed at given time could be *stale*, meaning that they do not truly reflect the precise load at various queues, as tasks can arrive from different clients. Even though such a topic on the load index is important, this is not the focus of this book. The aim here is on the task assignment policies that can provide substantial performance improvement for Web services.

LLF is proven to be *optimal* under certain conditions. Indeed, if most of the tasks have similar size (i.e. the task size follows an exponential distribution), then a sensible way to measure the length of the queues (i.e. Load Index) is to count the *number of tasks* in each queue, as all the queues have similar type of tasks in terms of size. This obviously make a sense conceptually, and has been mathematically proven by Pollaczek & Khinchin in 1930s (94)(75). However such an assumption is problematic in Web services and, in general, for Internet traffic. Assuming an exponential distribution for task size does not stand anymore, as several recent studies (e.g. (13)) formally show the high variance of task size in the Internet traffic. In other words,

the task size follows a different distribution, called *Pareto* distribution, where something like 1% of tasks could take up to 50% of the CPU time. Therefore a small number of tasks could be *very* large, which is problematic for the dispatcher attempting to equalise the load across different servers, as the number of tasks in a queue is not anymore the server's load. Indeed a very large task allocated to a given server (in the queue) can be much larger than thousands of small tasks in an another server. So the dispatcher needs to find the best way to allocate very large tasks and smaller tasks so the latter are not stuck in the queues behind very large tasks waiting for processing. This problem is known as *salvation*. Even more complicated, the high variance problem in the task size for Internet traffic is accentuated by the fact that the size of tasks at the dispatcher are not know beforehand (63, 64)(26, 97, 98), as SOAP requests are just XML calls and therefore there is not way to know how much CPU these will consume. They are known neither at compile nor at run time. Therefore the dispatcher will have a difficulties trying to figure out a way to *profile* tasks so it can efficiently allocate them to the right servers with less *waiting time* in the queues.

Dealing properly with two level performance problems of Web services, namely *system level* (i.e. those related to SOAP and XML) and *server level* (i.e. those related to the processing of SOAP requests at the server side), will definitely provide a more efficient and scalable infrastructure (in terms of performance) for deploying and running Web services. Despite the rapid growth in wired network bandwidth and steady increase in wireless bandwidth with new mobile technologies such as 3G networks, it is still not infinite. The available network bandwidth is often limited and expensive, especially in mobile and wireless environments. Enterprise IT systems need to process thousands of Web service requests in a short period of time. Considerable increased traffic represents high consumption of the network resources. Web services are promised to be a source of generating increased revenues for enterprises by exposing existing enterprise applications to a wide range of other applications on different platforms. High network traffic can hold up this potential for revenue generation and needs to be addressed. It is important to design Web services that have low communication overheads and make efficient use of available bandwidth.

What has been done so far in performance?

Several solutions have been proposed to improve SOAP performance, either using binary encoding (binary XML instead of textual XML), caching (at the client side by increasing the locality of objects), compression (by reducing the size of XML payload) or optimising the SOAP run-time implementation (by efficient optimisation of the kernel).

One type of solution attempts to reduce the size of SOAP messages by binary encoding (similar to CORBA encoding). It is transmissions of SOAP messages in binary instead of textual format. Generic SOAP engines support both textual and binary XML as the encoding scheme of messages. Scientific data could be directly transmitted as binary XML. (80) developed a binary XML encoding scheme called BXSA (Binary XML for Scientific Applications). BXSA supports the ability to con-

vert a textual XML document to binary XML and vice versa. A SOAP message is modeled in the BXDM model (a scientific-data-friendly XML data model as an extension of XPath Data Model (51)) instead of the XML Infoset. To send a SOAP message, first a SOAP message is constructed in the BXDM model, then the encoding policy provider is invoked to serialise the message into an octet stream. Finally, the stream is transferred by calling the binding policy provider. The reverse procedure takes place when a message is received. Both SOAP over BXDM/TCP scheme and SOAP with HTTP data channel have similar performance. They can rebind the BXDM transport to multiple TCP streams, thus it can carry larger messages.

W3C XML Protocol Working Group recently released specifications for SOAP Message Transmission Optimisation Mechanism (MTOM) (59) and XML-binary Optimised Packaging (XOP) (60). These specifications are targeted to multimedia data (such as JPEG, GIF and MP3) and data that includes digital signatures. The specifications define an efficient means of XML Infoset serialisation. An XOP package is created by placing a serialisation of the XML Infoset inside an extensible packaging format such as MIME (60). MTOM describes how XOP is layered into the SOAP HTTP transport. However, XOP and MTOM still possess a parsing issue inherited from SOAP and XML.

Another example of work in binary SOAP encoding is a study by Oh & Fox in (89). They proposed a new mobile Web service architecture, called Handheld Flexible Representation (HHFR), that provides optimised SOAP communication using a binary message stream. HHFR architecture separates XML syntax of SOAP messages from SOAP message contents. This separation is negotiated at the beginning of a stream. An XML schema is used to characterise the syntax of the SOAP body. HHFR is most suited to Web service applications where two end-points exchange a stream of messages, because messages in a stream often share common structure and type information of the SOAP body and most parts of the SOAP headers. The message structure and type in form of XML schema are transmitted only once and the rest of the messages in the stream have only payloads. Oh and Fox compared HHFR prototype with a conventional SOAP and found the higher performance advantage of HHFR is achieved when there are multiple messages transmitted in a session. In particular, HHFR streaming communication outperforms conventional SOAP by 7 times in round trip time for a service adding float numbers.

Compression is a popular method to deal with large message sizes of Web services. Compression is particularly useful for poorly connected clients with resource-constrained devices or for clients that are charged by volume and not by connection time by their providers. However, compression decreases server performance due to the additional computation required. From experiments of XML compression in wireless networks, (140) found that in a low bandwidth network such as GPRS the service time was halved when compressing large SOAP responses. The response time during overload is however about 40% higher and the server throughput is about 50% lower when compression is used. Therefore, Tian et al. proposed that clients should decide whether they want their responses compressed. During low server demand, responses to all client requests except those that did not ask for compression are compressed. During high server demand, only responses to clients that

asked for compressed responses are compressed. Despite high response time and low throughput, Tian et al. have shown that their dynamic compression approach is beneficial for both the server and for mobile clients with poor connectivity. It is also recommended that servers should only compress replies to clients that can benefit from compression.

Many studies have researched approaches to enhance SOAP performance through caching (41, 102, 137). Devaram & Andresen (41) implemented a partial caching strategy to cache SOAP payloads on the client side. In this method, the SOAP payload is cached when it is first generated. Every time the client makes a request, the payloads stored in the cache are reused to create a new payload by replacing some values of the XML tags with new parameter values. This technique is shown to provide better performance than non-caching for request messages with small number of tags. The performance of the partial caching technique degrades when there are many parameters defined in a SOAP request because the time spent on substituting the parameter values and accessing file I/O increases as the number of parameters increases, which in turn enlarges the size of the cache.

The advantage of Web services caching is mainly in supporting disconnected operations. Terry & Ramasubramanian (131) implemented an HTTP proxy server between a Web service provider and a Web service consumer to provide a simple cache for storing SOAP messages. Their study highlighted the benefits of employing a Web service cache to support disconnected operations. Specifically, in case of disconnection, SOAP response messages that are stored in the cache, will be returned to client requests. The SOAP requests are stored in a write back queue which is later played back to the server when the connection to the Web service is restored. However, there are still many issues with caching such as consistency and availability of offline access to Web services. Another difficulty with Web service caching is that a cache manager does not know which operation needs to be played back to the server. In addition, the effectiveness of a cache is often dependent on the similarity of future requests to past requests.

Liu & Deters (102) proposed a dual caching strategy for mobile Web services. In their method, one cache resides on the client side and the other on the server side to handle any problems due to loss of connectivity during the sending/receiving of request and response messages. The two caches are coordinated by a cache manager. An ontology Web language is used to describe meta-data used on the caches such as service description, client workflow description and connectivity description. This ensures interoperability with other Web service standards. Terry & Ramasubramanian (131) also emphasised the importance of understanding the cacheability of services in their work. Therefore, they propose to add annotations in the WSDL specification to support SOAP caching. The suggested annotations include semantic information such as cacheability, life time, play-back and default-response. This however leads to issues regarding standards and interoperability.

In cases when outgoing SOAP messages are very similar in content, it is advantageous to use differential encoding. With this technique, only the difference between a message and a previous one is sent over the wire. Documents containing only the differences can be more compact in size depending on content. Important studies in

the differential encoding area are done as a differential serialisation (3), a differential deserialization (130), and differential compression (145).

After discussing the various techniques for system level optimisation (i.e. SOAP optimisation), the next step is to look at those techniques that optimise performance at the server side. As the reader may notice, one can use an optimised SOAP engine (to efficiently send requests to severs), but this does not guarantee overall performance improvement, as the server/s could be overloaded (and therefore delaying most of the executions of tasks). Carefully addressing the performance issues at the server is a key to the development of efficient and scalable Web service infrastructures. Both system and server considerations need to be addressed so substantial performance gain can be obtained. Unfortunately there is no **holistic** approach to **performance** in Web services (as well as in other systems). Both optimisation, system and sever optimisation, need to be done in combination so delays can be reduced both during the forwarding of requests as well as their processing at the server side. Obviously there exists a third dimension for Web Services performance optimisation, which relates to the application level (i.e. application programs). Applications can be badly written, which can induce additional overheads on the server side. A simple example of such bad design is when a given call (to return a given data structure - a large array) is repeated several times in a program. Therefore there will be a wastage in the usage of the network as well as CPU time in the server. Instead, a good design of an application program will store the retrieved data structure (in the cache) and later use for future processing. This book does not deal with such program design, as it has a different focus, namely a *system performance focus* for Web services. The design of "good" Web services as well as application programs is the area of software engineering. We advise the reader to look at the work by Perepletchikov & Ryan (92)(93), where they came up with a new design methodology as well as a set of (design) metrics for Web services.

In the context of server side performance, static-based approaches (like RR - Round Robin) definitely do not have a place in Web services, as they do not *properly* allocate tasks (i.e. XML-based requests) to the right servers. Indeed, RR will allocate tasks to an overloaded server (as it does not take into account the load during the task assignment process). As mentioned earlier, a more advanced policy, called LLF (Least Loaded First), does smartly assign task to the least loaded server (i.e. the server that has the least number of tasks in its queue). LLF is an optimal (task assignment) policy, under the condition that the task size does follow an exponential distribution. Of course, this can work when appropriate mechanisms to estimate the load at various servers are available. The computation of the load index is even harder when dealing with distributed environments (because of the network delays, and therefore making load information staler). Coming back to the assumption for the exponential distribution, this is not anymore true for a lot of cases of traffic, including for Internet traffic (13), and some mathematicians are even questioning the poison distribution (for arrival rate), as this many not be true for certain traffic. Here we will not go that far, but the research community do agree that task size follows a different distribution, namely Pareto distribution (or more Bounded Pareto distribution in the case of Web traffic). Such distributions are explained in