Investigating the Limit of Web Service based Technology for High Energy Physic Experiments

F. Lelli[1], G. Maron[1]
1 INFN Laboratori Nazionali di Legnaro

INTRODUCTION

We investigate the limitations of XML messaging systems for high-performance and high-interactive distributed computing. Our experimental results clearly show that if we focus on improving the performance of parsers, which are routinely used for de-serializing XML messages exchanged in these systems, we can also improve the overall performance of any distributed system based on XML-based messaging technologies.

UNDERSTANDING THE XML LIMIT

The general architecture reviewed is applicable across different technologies [1], [2] so that we use it to understand the limits that an XML solution can introduce in terms of interactivity and requests handled per second. A modern web-based enterprise application has three layers (see figure 1 (a)):

A client layer, which is responsible for interacting with the user, e.g., for rendering Web Page;

A middle tier that includes:

A Presentation Layer, which interprets user inputs (e.g., her/his submitted HTML forms), and generates the outputs to be presented to her/him (e.g., a WebPage, including their dynamic content).

A Business Logic Layer, which enforces validations, and handles the interaction with the data layer.

A data layer, which stores and manages data, and offers the handling interface to the upper layers.

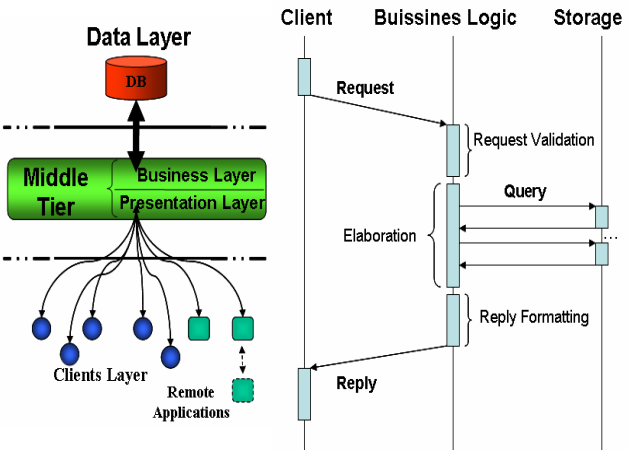

Figure 1 3-tier Architecture (a) and Sequence diagram (b)

This structure allows changes in legacy host access and development of new business logic to be kept separated from the user interface, dramatically reducing the cost of maintenance. Three-tier architectures also enable large-scale deployments, in which hundreds or thousands of end users are enabled to use applications that access business information. We would like to underline that we could have particular cases where the Data Layer size is particularly small, and the information exported by the Data Layer also change slowly. In these scenarios the business layer could act as proxy cache for the Data Layer, speeding up the service request process.

Talking about business to consumer applications, the client layer of a web application is implemented as a Web browser running on the user's client machine. Its job is to display data and let the user enter/update data.

In a business to business scenario the client layer can be a generic application, compliant to the WS standard.

The presentation sub-layer generates (or displays) Web Pages, or produces (or interprets) XML-based SOAP messages in a WS scenario. If necessary, it may include dynamic content in them. The dynamic content can originate from a database, and it is typically retrieved by the Business logic that:

- performs all required calculations and validations;
- manages a workflow (including keeping track of session data);
- handles all the needed data access.

For smaller Web applications, it may be unnecessarily complex to have two separate sub-layers in the middle tier. In addition the sub-layer communications typically do not use XML. From a temporal point of view (showed in figure 1(b)), a client (WS, Web browser, java and c/c++ programs, etc) performs a request to the Business logic, which dynamically retrieves the requested information. During the elaboration phase, the server can either perform one or more queries to a persistent storage, or interact with other sub-business unit. Once the information is retrieved, the server formats the retrieved information in a way that the client is able to understand, and sends it back to the client. In a WS scenario, the clients need to exploit a further SOAP protocol layer over HTTP communications, while in human client scenario the communication can be made just via HTML over HTTP. In order to set up a system with an architecture similar to the one presented at the beginning of this Section, in the same local LAN we deployed two different server machines, the former hosting an Oracle DB Server as a storage system, and the latter hosting a Tomcat Application Server as a Middle Tier.

Then we run a set of clients on other different machines in the same LAN (Ethernet 1 Gb/s).

The hardware used in this test is:

Database Server: Dual Xeon 1.8 GHz, with 2 GB RAM, OS Red Hat Linux Advanced Server 2.1.

Application Server: Dual Xeon1.8 GHz, with 1.5 GB RAM, OS Red Hat Linux Advanced Server 2.1.

Clients: 25 machines, equipped with Pentium III 600 MHz, with 256 MB RAM, OS Linux Red Hat 9.0.

As Client/Server communications we used:

- Simple HTTP
- SOAP/XML over HTTP

Within the same Tomcat server, we also set up an Axis engine in order to evaluate the performance of a WS communication (i.e., SOAP/XML+HTTP with automatic generated stubs) between client and server.

The DB table structure and the complexity of the queries are really simple (i.e., a single relational table, with only five attributes). Our benchmark is thus the following: the clients (Java applications) perform a request to a Tomcat Servlet or to a WS. The Business logic layer, in order to present the result to the client, performs a query using a pre-generated JDBC connection, to the DB server. Then the same layer formats the result and sends it to the client.

In this scenario we evaluate the application server throughput in terms of satisfied requests per second. The results are shown in next paragraph.

EXPERIMENTAL RESULTS

We performed a test aimed to understand the number of requests handled per second in two different scenarios. In the first one, a servlet or a Web Service, once invoked, performs a query on the storage system and sends back the result. In the second scenario, we remove the Data Layer in order to exactly measure the service invocation time. The plot in Figure 2 is related to the first set of measures. The "ServletRPCXML" curve refers to a servlet that performs exactly the same job as a WS. In particular, only when all the result data are completely prepared, i.e. formatted according to an XML schema, such data are returned to the client. The "ServletRPC" refers to a servlet that does exactly the same job as in the case, but the exchanged data is simply serialized using HTTP. The "Servlet" curve differs from the first one because, instead of buffering the result data, and sending them at the end of the preparation phase, the server immediately starts sending it in a streaming way using an HTTP custom serialization. We can also note that the performance of WS is similar to "ServletRPCXML". This means that its large performance degradation with respect to the "Servlet" case is due to: (1) buffering of the results and (2) XML serialization/de-serialization. In particular, the latter entails additional computational and synchronization overheads on both the client and server sides. In addition, the exploitation of a WS infrastructure may prevent simple, but effective, code optimizations, like the one adopted in the "Servlet" case.

The plot in Figures 3 refers to the invocation of an empty service that immediately returns the results with a constant size (10 tags) to the client. While in the previous set of tests the size of the service input was fixed, and the size of the service output was varied, in this second set of tests we varied the size of the input messages, by keeping constant the output ones. As we can see, once again, the cost of WSs in terms of number of handled requests per second is not negligible, and limits scalability since this number does not increase when we raise the number of clients.

Finally, as one can expect, when messages are short and less complex (in terms of XML tags) the throughput is better than when messages are large and more complex.

We need to point out that these tests have been repeated varying the number of tags in input and the query results size. In this report we decide to show only a fixed size.

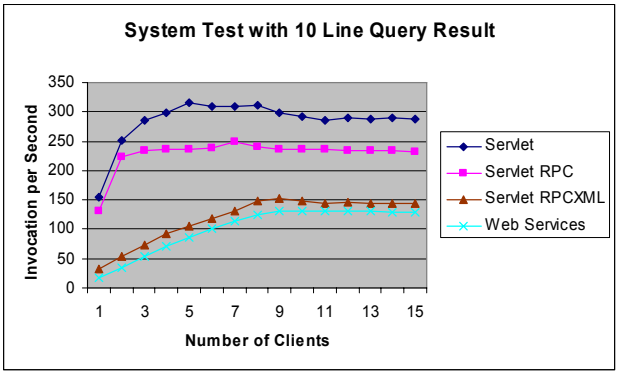

Figure 2: Handled request per second using a storage tier query result size of 10 elements

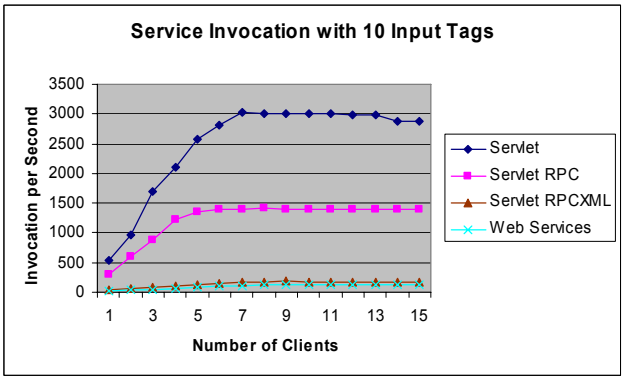

Figure 3: Handled request per second, input size of 10 elements

[1] D. Menasce, V. A. F. Almeida Capacity Planning for Web Performance: Metrics, Models, and Methods (Paperback) Prentice Hall PTR; Bk&CD Rom edition (May, 1998)

[2]J. Blommers, HP Professional Books Architecting Enterprise Solutions with UNIX Networking (Paperback) Prentice Hall PTR; 1st edition (October 15, 1998)