A Peer to Peer Based Data Acquisition for Quality of Service Studies

F. Lelli
INFN Laboratori Nazionali di Legnaro

INTRODUCTION

We present the data format, architecture and implementation of a Peer to Peer based Data Acquisition (DAQ) System that has been used for acquiring data regarding Web Service invocation. This system has acquired data for more that 1 month.

DATA FORMAT AND MEANING

From a generic point of view a remote method invocation can be split into 7 crucial parts, as explained in figure 1 [1]. During the interval *t1-t0* the client serializes the invocation input in SOAP format and sends it during the interval *t2-t1*. The remote peer receives the serialized message at time *t2* and starts the deserialization process that finishes at time *t3*. During the interval *t4-t3* the remote method is executed and the output is produced. As its last operation, the remote services serialize the output in SOAP format during the interval *t5-t4* and starts sending it. The invoker receives the serialized message at time *t6* and during the interval *t7-t6* starts deserializing the output message in the proper data structure.

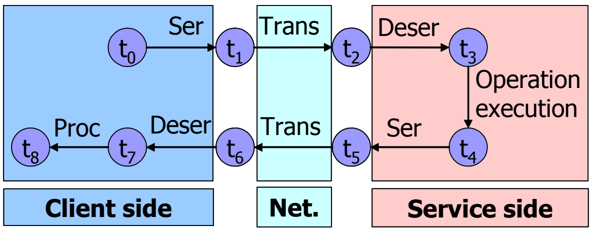

FIG *1. Critical Intervals in a Web Service invocation*

According to this time division presented in Figure 1, from the client QoS point of view, we can identify 3 critical times intervals, that are:

- t7-t0 Remote Method Execution (t_{ex})
- t4-t0 Remote Method Processing (t_{pt})
- t3-t0 Remote Method Invocation (One Way) (t_{ow})

The *t7-t0* interval represents the total execution time of a remote invocation. In other words, in a synchronous method invocation, the time that the invoker waits before starting to process the output of the remote invocation.

The *t3-t0* represents the remote procedure execution delay that is introduced by the serialization-deserialization process. This is particularly important in One-Way [1] invocation method.

Finally *t4-t0* is the time needed to complete the remote elaboration process. In Notification based systems, it represents the minimum time before the first reception of a notification.

In order to give an intuitive idea of the problem difficulty, we can note that all critical intervals (*t3-t0*, *t4-t0*, and *t7-t0*) depend on both server and client status. Furthermore, due to the clock synchronization problems [5], some intervals (*t3-t0, t4-t0*) cannot be estimate without allowing cooperation between these 2 peers.

An additional remark is related to interval *t4-t3,* which represents the execution time of a remote method.

DAQ ARCHITECTURE AND IMPLEMENTATION

From an intuitive point of view when the WS client performs a service invocation it can send also t0, t1, t6, and t7 times to a QoS Metric Collector component. In the same time the WS Server replies to the client invocation and send t2, t3, t4, t5 to the Collector too.
The Collector receives the information, correlate the times, construct the interval mentioned in the previous Section and store it.

A mandatory requirement is that both client and server should minimize the additional operation in order to let the Collector component compute the intervals without introducing unpredictable overhead.

Figure 2 shows the test-bed architecture: it has been realized as a P2P system that runs in parallel with the web service invocation and exchanges messages in the same network using TCP channels. Different software components have been developed:

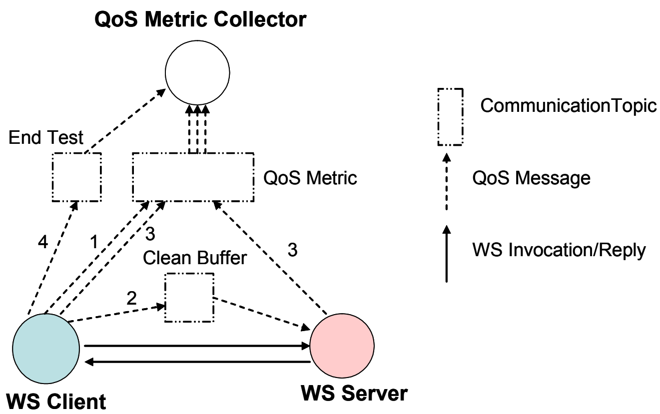

FIG *2. Software Architecture*

A WS-Server that:

- Replies to remote the client's remote call
- Sends its time information to the Collector component.
- Synchronizes itself with the client in order to let

the collector receive coherent time from both client and server

A WS-Client that:

- Remotely invokes the server
- Sends its time information to the Collector component
- Synchronizes itself with the server in order to let the collector receive coherent time from both client and server
- Notifies the QoS Metric Collector when the current test is finished.

A QoS Metric Collector that:

- Receives tests description coming from the WS-Client
- Receives and Collects times information coming from both Server and Client
- Receives test notification related to the test status coming from the WS-Client
- Saves all the QoS significant intervals

In order to allow the interaction between the system peer, three communication channels have been setup. The first one for dispatch time metric to the QoS Metric Collector (QoS Metric), the second for the Client-Server synchronization (Clean Buffer) and the last between the Client and the Collector in order to send notification related to the current test status (End Test). From a temporal point of view the component interactions are described in Figure 3 and they are also remarked in Fig. 2.

Firstly the client initiates a new test sending the test description to the Collector; secondly it sends a clean message to the server in order to force a clean up of the eventually buffered time measures, thirdly the real test begins and time information is collected during the web service invocations. Finally, once all the tests invocations are finished, the Client notifies the collector.

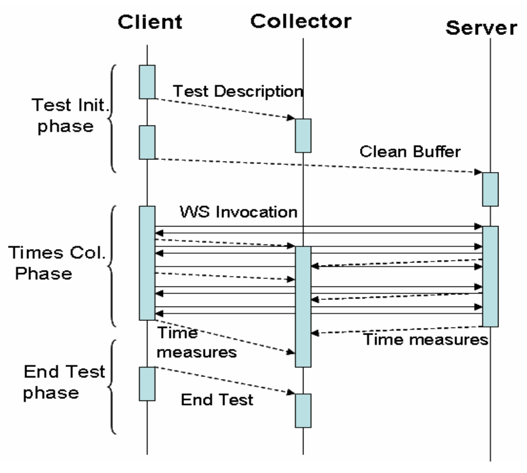

FIG *2. Component Interaction Diagram*

A critical point in these tests is the position of the timing probe in both the client and the server; a wrong position can cause unpredictable behavior in the time slice. Times like t0 and t7 can be easily collected just after and before the method execution while t3 and t4 can be taken just after and before the beginning/end of the remote method.

Unfortunately t1 and t6 at client side, and t2 and t5 at server side are inside the web service engine so we where forced to choose an open source product for our tests. The source code of Axis [1], [4] (version 1.4) was modified introducing our probe and then used in order to create the web service client and server.

To reduce additional code overhead the collector was running in a separate machine and the collected times was buffered on both client and server side and sent in a single messages to the collector using a simple TCP connection. A JMS Library [2], [3] was used to create the communication channels and it was instructed to send messages in background. Before starting the constitution of the dataset, a comparison test was performed, in order to measure the overhead introduced by this modification. This test was a comparison between a standard web service and the same remote service equipped with the above presented software and the result was negligible.

For controlling the CPU load a separate work and sleep java program has been created and it was running in both the client and the server machine.

As final remark the Collector was realized as a Java stand alone application and was instrumented using a JMX [4] library in order to provide on-line information about the tests.

It is well known that Java systems introduce 1ms of uncertain in every measure while the client-server clocks synchronization has a better precision given that the machines are directly linked by an Ethernet switch [5]. Considering that the experienced time intervals are about 100ms or more, we can consider this error negligible.

[1] S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, and Y. Nakamura. Building, Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. Sams, December 2001.
[2] Mantaray Project: http://www.mantamq.org
[3] JMS standard API: http://java.sun.com/products/jms/
[4] Vivek Chopra, Amit Bakore, Ben Galbraith, Sing Li, Chanoch Wiggers, Professional Apache Tomcat 5 Wrox May 2004
[5] Network Clocks Synchronization. http://zone.ni.com/ devzone/conceptd.nsf/webmain