Pre-Printing ID: INFN-LNL-212(2006)

Enabling the Web Service Quality of Service

F. Lelli[1,2], G. Maron[1], S. Orlando[2]

[1]*Istituto Nazionale di Fisica Nucleare, Laboratori Nazionali di Legnaro, ITALY*

[2]*Dipartimento di Informatica, Università Ca' Foscari di Venezia, ITALY*

Version 1.0
9/13/2006

Version:	1.0
Date:	9/13/2006
Authors:	F. Lelli[1,2], G. Maron[1], S. Orlando[2]
CI identifier	INFN-LNL-212(2006)

Revision History

Date	Version	Description	Author
9/10/2006	0.9	First Public Draft	Francesco Lelli
9/13/2006	1.0	Document Finalization	Francesco Lelli

CI Record

Field	Description
Description	Web Service Dead Line Computation Methodology
Submission Date	9/10/2006
Submitted By	Francesco Lelli
Components	1 document
Dependencies/Related	none
External Identifier	INFN-LNL-212(2006)
Point of Contact	Francesco.Lelli@lnl.infn.it
Comments	This document addresses and solves the Quality of Service (QoS) issues in a Web Service scenario
Physical Location	INFN Laboratori Nazionali di Legnaro

Table of Contents

Enabling the Web Service Quality of Service

1 Introduction

This document addresses and solves the Quality of Service (QoS) issues in a Web Service scenario. The main contributions of this report include (1) the creation of a uniform and coherent dataset that enable QoS studies and (2) a methodology that allows the clients estimation of a remote method execution time. (3) Experimental results in real environments validate the proposed algorithms that allow the dead lines estimation of a method invocation and (4), finally, 3 different QoS enabled architecture solutions that adopt the proposed methodology and algorithms have been outlined.

1.1 Document Organization

This document is organized as follow:

Chapter 2 resumes the main contributions of this document providing an overview of the obtained results.

Chapter 3 report and introduce the QoS in a distribute environment mapping the key features in a Grid and in a WS environment

Chapter 4 introduces the main QoS issue in a WS environment presenting the status of art and analyzing the critical parameters that influence the Quality of Service

Chapter 5 propose a dataset organization that enable QoS studies

Chapter 6 describes how the proposed dataset has been created and verify the data consistence

Chapter 7 describe a subset of the collected raw data

Chapter 8 describe a methodology that enable the estimation of a remote method execution time

Chapter 9 validate the proposed methodology in a real scenario

Chapter 10 propose an al algorithm that automatize the procedures of the developed methodology

Chapter 11 suggest 3 different software architectures where the developed algorithm could be integrated

Appendix I and II provides some detailed on the used methodology.

2 Executive Summary

This document analyzes the present QoS status of art and prose a set of solutions that enable a remote service invocation with a given Quality of Service.
Chapter 3 and 4 formalize the problem and recognize the need of a uniform dataset where study the QoS in a Web Service context. The created data set consists in 2304000 samples (of 9 significant values each) organized in 2304 different tests; the information organization (Ch5), the test bed architecture (Ch6) and the obtained raw data (Ch7) unsure the quality of the produced information.
In the second part this document analyzes the realized dataset in a more exhaustive way. 2 variants of a methodology that utilize a Gaussian approximation of the dataset distributions in combination with regression model of the key factors that influence the average and the standard deviation have been developed in chapter 8 and validate with experimental results in a real scenario in chapter 9. Finally, in chapter 11, this document proposes 3 different software architectures that can utilize the developed algorithms presented in chapter 10.

3 *Quality of Service in a grid Scenario and in a Web Service Scenario*

This paragraph tries to introduce the quality of service from a general point of view while next will personalize this approach into a grid, interactive grid, and WS scenario.
The Quality of service (QoS) is a combination of several qualities or properties of a service [20], such as:

- **Availability** is the percentage of time that a service is operating.
- **Security** properties include the existence and type of authentication mechanisms the service offers, confidentiality and data integrity of messages exchanged, non repudiation of request or messages, and resilience to denial-of service attacks.
- **Response Time** is the time a service takes to respond to various types of requests. Response time is a function of load intensity, which can be measured in terms of arrival rates (such as requests per second) or number of concurrent requests. QoS takes into account not only the average response time, but also the percentile (95th percentile, for example) of the response time.
- **Throughput** is the rate at which a service can process requests. QoS measures can include the maximum throughput or a function that describes how throughput varies with load intensity.
- **Application Dependent Parameters** are values that belong to the particulars service semantic such as the money cost, recall in a Google query and so on.

Considering a set of service request (**Transaction**) additional parameters must be considered. A Service Transaction is an inseparable list of operations which must be executed either in its entirety or not at all. In database parlance, it is a sequence of actions that must be executed as a unit. For example, when a Web site sells a travel package to a customer, the site must confirm all components of the package (flights, hotels, and car rental reservations). It is common to require distributed transactions to have the ACID property in the presence of any type of site or network failures:

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not at all.
- **Consistency**: A correct execution of the transaction must take the system from one consistent state to another.
- **Isolation**: A transaction should not make its updates visible to other transactions until it is committed. That is, it should run as if no other transaction is running.
- **Durability**: Once a transaction commits, the committed changes must never be lost in the event of any failure.

As final remark we need to observe that the **QoS measure is observed by services users**. These users are not only human beings but also programs that send requests for services to service providers.

3.1 QoS in a Grid Scenario

Standard Grid only considers batch job execution where:

- A user can require the execution of a single job or the coordinated execution of a set of jobs, typically modeled as a workflow;
- A user can specify requirements for one or more resources that should serve the

request;

- A user can specify requirements on the network connectivity of the resource executing the job and the input data that can be present in different replicas at several locations;
- A user may express policies about the job, or the user job itself can be subject to policies provided by the resource provider or by the virtual organization to which the user belongs.

Mapping the presented use cases in QoS scenario and considering a single job execution: Availability represent the possibility to submit a job with optionally given constrain; security represents the different types of user authorization and identification capability while, taking into account that we are considering batch job execution, parameters like the response time and throughput represent the job handling capability. Finally the users' job policies are application dependent parameters. Considering a set of job (workflow) the ACID properties must be taken into account and for guarantee these request standards and software component that provide advance reservation, like the Agreement Service [25], has been proposed and developed.

3.2 QoS in a Interactive Grid Scenario

Interactive grids systems not only refer to batch execution job but also the need to control the job flow is outlined. We can repeat the same QoS remarks presented in paragraph 1.1 and in addition we need to consider that in this particular case the jobs execution times are typically shorter than the one that runs in a standard grid. Finally, considering that interactivity is a key requirement, the jitter of the job execution time assumes a non negligible meaning.

3.3 QoS in a Web Service Scenario

In this document we refer to a Web Service as a *platform and implementation independent* software component that can be:

- Described using a service;
- Published to registry of service;
- Discovered through a standard mechanism (at run time or design time);
- Invoked through a declared API, usually over a network;
- Composed with other services;

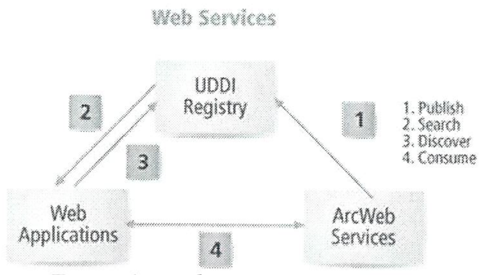


Figure 1: Web Service Architecture.

According with the given definition one point we would like to stress out is that Web

Services need not necessarily to exist on the World Wide Web; a web service can live anywhere on the network, inter or intra-net. Some Web Services can be invoked by a simple invocation method in the same operating system process, or using shared memory between tightly coupled processes running on the same machine. Another important point is that a Web Service implementation and deployment platform details are not relevant to a program that is invoking the service; a Web Service is available through its declared API and invocation mechanism (network protocol, data encoding schemes and so on). Finally we would like to remark that web services are just building blocks that can be composed in order to create the final application therefore this software layer do not care about the invocation semantic remanding it to the WS-developers.
Considering a single service invocation and this last remark Availability, Security, Response Time and Throughput are important QoS parameters and they have a fundamental impact on the WS-Quality of Service while Application Dependent Parameters should be taken into account to higher software layers.
Finally, considering that the service semantic is not part of this software layer and that web service just provide an interface that allow the remote access to a generic software component, ACID properties can not be part of the WS level but, again, are business of the web service developers.

4 Approaching the Web Service Quality of Service

As mentioned in paragraph 3.3 the first remark is that web services are method semantic agnostic, therefore an application non intrusive framework that is and able to provide high quality of service parameters, like the money or electrical energy needed, can not be provided without enforce a modification of the final programmer application. These QoS parameters and others, like particular software compliance or the method semantic in general, are application dependent and belong to the software layers that use web services as hardware and software independent invocation framework in order to execute a remote service.
If we consider a Web service call as just a remote procedure execution, how often a the server can be successfully connected by a client or how frequently an exception is detected by the client due to a server failure (Availability), are interesting parameters; a scenario where a set of servers offer the same service to a pool of clients is the typical use case that require a distinction between "good" and "bed" servers. QoS parameter related to the Security can be considerate static or quasi-static information because typically are properties established at deployment time.
An additional parameter could be a sort of "magic number", either a number (or a set of them) that statically characterize the responsiveness of the server. This number(s) is a measure of the hardware and software quality and do not take into account the network and/or the machines load; it just describe the potentiality of the remote service. Unfortunately this number (s) can not be used in order to determinate the End to End Response Time and/or the Throughput of a service from the client point of view because the server and the clients load have a considerable and not negligible impact. By the way achieving this type of QoS is quite easy. If we assume that these parameters are not client dependent this information could be inserted into the UDDI description or in a centralized repository; as opposite, if the client could influence the particular remote method failure, it could maintain this information locally. Considering that the "magic number" is a static value it can be computed using a set of tests that could run during the installation procedure.
As last parameter, and most crucial in remote procedure calls, is the remote method execution time. The estimation of this factor, unlike the previous mentioned, is not trivial and quite more important in real time, near real time and interactive distributes applications.
The following table resumes the QoS parameters type for a WS scenario:

QoS Parameter	**Type**	**Component Influence**	**Trivial Determination**
Availability	dynamic	Server, Network	Yes
Security	Static	Server	Yes
Response Time	dynamic	Client, Network, Server	no
Throughput	dynamic	Client, Network Server	Yes
Application Dependent Parameters	Out of WS scope	---	---
ACID	Out of WS scope	---	---

Table 1: Parameters characterization in Web Service QoS

This problem is quite new and some preliminary results were presented in [6] [7] [8] [9] [10], [11] [12], [13], [14] as we will outline in the next paragraph.

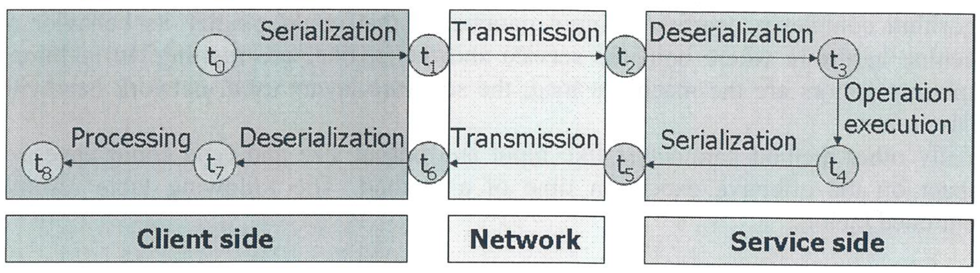
4.1 Related Work

In [6] authors develop a service composition architecture that optimizes the aggregate bandwidth utilization within an operator's network. In [7] authors propos a new Web services discovery model based on UDDI in which the functional and non-functional requirements (i.e. quality of services) are taken into account for the service discovery while [9] propose a fault tolerance solution or performance prediction [14] for distributed systems based on Web service technology by extending the UDDI. [8] address service selection coupled with a QoS ontology proposing architecture based on an agent framework while [10] presented a QoS-assured composed service delivery framework, called QUEST, and design efficient approximate optimal algorithms to compose service paths under multiple QoS constraints. In [11] authors propose a service composition architecture that optimizes the aggregate bandwidth utilization within a operator's network while in [12] Cardoso et all present a predictive QoS model that makes possible to compute the quality of service for workflows automatically based on atomic task QoS attributes. Finally in [13] authors study the Web Service end to end QoS proposing a centralized broker that is responsible for coordinating individual service components to meet the quality constraint for the client.
We need to note that researches are focussing there attentions on a techniques and/or architecture that assume the response times well known and they leave the parameter estimation to future investigation. In addition some works assume instantaneous and costless the propagation of the QoS information between the system peers.

4.2 Critical Times in a Remote Procedure Call

Is well known that the majority of the web service request are performed across a LAN or internet so the used network channel have an important role during a remote procedure call and it can deeply influence the execution time. Several projects [1], [2], [3], [4], [5] are trying to estimate the end to end network quality of service between 2 remotely devices. Unfortunately not only the wire influence the performance of a WS invocation but the serialization and deserialization process on both, server and client side [18], [19], and the remote method algorithm have a important role. By the way a good estimation of the network quality is an important piece of the puzzle if we are invoking a service using a channel with limited bandwidth or high latency.
From a generic point of view a remote method invocation can be spitted into 7 crucial parts [24] as explained in figure 2:

A Remote method Invocation:



Remote Method Invocation activities

- **t1-t0** input serialization
- **t2-t1** input communication
- **t3-t2** input deserialization
- **t4-t3** remote method execution time
- **t5-t4** output serialization
- **t6-t5** output communication
- **t7-t6** output deserialization

Crucial Times are:

t7-t0 Remote Method Execution Time
t4-t0 Remote Method Elaboration Time
t3-t0 Remote Method Invocation Time

Figure 2: Critical Intervals in a Web Service invocation

During the interval t1-t0 the client serializes the invocation input in SOAP format and sends it during the time t2-t1. The remote peer receives the serialized message at the time t2 and start the deserialization process that will finish at the time t3. During the interval t4-t3 the remote method will be executed and the output will be produced. As last operation the remote service will serialize in SOAP format the output during the interval t5-t4 and it will start the sending process. The invoker will receive the serialized message at time t6 and during the interval t7-t6 start deserialize the output message in the proper language.
According with this time division, from the client quality of service point of view, we can identify 3 critical times intervals, that are:

- t7-t0 Remote Method Execution Time
- t4-t0 Remote Method Elaboration Time
- t3-t0 Remote Method Invocation Time (One Way)

t7-t0 interval represent the total execution time of a remote invocation. In other words, in a synchronous method invocation, the time that the invoker will wait before start processes the output of the remote invocation.
t3-t0 represent the remote procedure execution delay that is introduced by the serialization-deserialization process. This is particular important in One Way [17] invocation method.
Finally *t4-t0* is the time needed to finish the remote elaboration process. In Notification base systems [17] represent the minimum time before the first reception of a notification.
With this formalization equip Web Services with QoS can be translated in try to guaranteed a service execution in a given interval of time.
Of course, noting that no real time Web Service engine has been provided, the Service Requestors can not negotiate a particular service but still can try estimate the execution time of a remote method invocation. In order to give an intuitive idea of the problems difficulty, we can note that all critical intervals (t3-t0, t4-t0, and t7-t0) depend on both server and client parameters; in addition some intervals (t3-t0, t4-t0) can not be estimate without allowing cooperation between these 2 peers.

4.3 Factors that Influence a Remote Method Invocation:

Generally speaking the method execution time depends on several factors; first of all, the algorithm complexity, particular input parameters that could modify its behavior and the machine hardware where both the service and the invoker are running. In addition, other important factors are the machines load, the software layers used, network bandwidth and delay.
Finally other method parameters like input and output size and type could have a crucial impact on the effective execution time of a method. The following table resumes, the mentioned factors:

Parameter	**Type**	**Label**
Hardware Client Side	Static	A
Hardware Server Side	Static	B
Software Layers Client Side	Static	C
Software Layers Server Side	Static	D
Client Load	Dynamic	E
Server Load	Dynamic	F
Algorithm of the Method	Semantic	G
Key Factor that change the algorithm behavior	Semantic	H
Input Size	Run Time Static	I
Input Type	Run Time Static	L
Output Size	Run Time Static	M
Output Type	Run Time Static	N
Network channel bandwidth	Dynamic	O
Network channel delay	Dynamic	P

Table 2: factors that influence a remote method cal

Parameters like A, B, C, D can be considered static, they depend on hardware and software update. Parameters G and H depend on the particular method semantic and a characterization must be provided by the user of a WS-framework. Factors like I, L, M, N can be considered note before starting the method execution even if, from a general point of view, M could be noted only before the time t4. Finally parameters like E, F, O, P can be considered fully dynamic and, in general, out of the program control.

5 An Homogeneous and Consistent Data Repository for Web Services QoS case of Study.

The main motivation of this preliminary work it that QoS over a Web service system is a relatively new topic. Some results were presented in [10] [13] but the problem has never been approached in a homogeneous way, as result it is impossible to evaluate the effective performance of different techniques. We believe that, in analogy with the Data Mining discipline, the creation of Data Set Repositories can allow the possibility to run comparison tests between different QoS methodology developed by the research community.
As we will point later, an addition important motivation for this work is the problem breaks down; retrieve the metric needed for the analysis require advanced programming and software architecture knowledge in order to put the proper time measurement in the software. As opposite, in order to elaborate the acquired data and propose a set of prediction algorithm a mathematic expert is needed.
Finally, if different expert programmers decide to construct there own test sets they could put the time measures in different place of the programming code and this does not contribute to the constitution of a uniform test stand for the WS QoS.
What is following is a description of the information memorized in this first proposed data set.

5.1 Test Sample: Definition and Memorized Information

A generic data set that must be used in conjunction with different techniques should contains all possible information that different methodologies can use in order to estimate the execution of remote invocation. Considering a remote method invocation as a test sample and, according with the model described in the previous chapter, we decide to store all possible web service crucial intervals:

- t1-t0 Client Serialization
- t2-t1 Client Network Transmission
- t3-t2 Server Deserialization
- t4-t3 Effective Remote Method
- t5-t4 Server Serialization
- t6-t5 Server Network Transmission
- t7-t6 Client Deserialization

In addition will be useful the computation of the mentioned QoS crucial interval in an independent way:

- t7-t0 Remote Method Execution Time
- t4-t0 Remote Method Elaboration Time
- t3-t0 Remote Method Invocation Time

5.2 Test Sets Dissertation and Recommendations

Factors that influence a remote method invocation time was described in paragraph 4.3 therefore each single factor must be considered choosing the candidates for the data base test sets.
Parameters that depend on the hardware and software layers (A,B,C,D) in a single set can be considered fix but the test software code must be portable in order to allow the possibility to repeat the tests in different configuration. Parameters like G and H belong to the semantic of

the method and a characterization of these factors can not be performed without knowing the method meaning therefore we decide to leave these parameters out of the first test sets.
I,L,M,N factors, that belong to the input and output size, mast be included in the test set because the exchanged messages size have a big impact on the system performance, and consequently in the method execution time, as demonstrate in several benchmark [18] [19].
Dynamic and unpredictable parameters like E,F,O,P must be handler in 2 different way. First of all in a real and concurrent scenario, where different clients try to access to the same service secondly where concurrent processes are running in a random way in both the client and the server machine. And finally in a synthetic scenario where all the dynamic parameters are under the test control in order to be able to predict this factor during the previous more concrete scenarios.
To summarize we can divide the test sets in 2 different categories

- Synthetic test
- Real tests

In the first one test factors are under the control of the test, while, in the second set dynamic parameters are leaved out of the test control.
We can conclude this dissertation noting that, due to the amount of software layers and the usage of not real time operative system, even if we are able to fix all the mentioned factors we will still experience a fluctuation on the estimated time interval so every single test must be repeated a sufficient number of times.

5.3 Test Sets Candidates Description

According with the previous paragraph we run the 2 different types of tests set Synthetics and Real.
The variables that we consider in the Synthetic test are:

- Input Type= [none, String, double, String and Double]
- Output Type= [none, String, double, String and Double]
- Input Size, = [0, 100, 1000, 10000]
- Output Size, = [0, 100, 1000, 10000]
- CPU usage Server=[0%, 70%,80%]
- CPU usage Client=[0%, 70%,80%]

In all test both, client and server, where running in a Dual Xeon 2.40GHz, 1.5GB RAM machines running the CERN Scientific Linux 3.0.4 Operating System, with Kernel 2.4.21-27.0.2.EL.cernsmp and Java 1.4.2_08-b03, linked to each other by a 100 MB Ethernet switch. Finally on top of this hardware and software layers we install Tomcat 5.0.28 and Axis 1.4 as Web Service provider.
For every single test 1000 samples was taken and a full test set consists in all possible combination of the above mentioned parameters. In other words the data set consist in 2304000 samples (of 9 significant values each) organized in 2304 different tests.
Still some variable are not covered from this first dataset and are:

- Long, float, integer in the input of a method
- Long, float, integer in the output of a method
- String size different from 10 character in both input and output size

In addition the network connection, the hardware and the software were always considered fix:

- Network occupancy different than 0%

- Network delay different than less that 1 ms.
- Hardware and Software Server side
- Hardware and Software Client side

The above mentioned variables are currently in evaluation and will be included in future improvement of this dataset. By the way, we believe that the analysis of this preliminary set of sample can be sufficient for the developing of a prediction methodology that can allow service invoker to predict the execution time of the remote method.
We also believe that an analysis of the web service performance varying the network channel is the most important miss think in this dataset. But, as we point in paragraph 4, different projects [1], [2], [3], [4], [5] are currently studying this problem and are trying to provide a service oriented system for estimate the effective performance of a link connection across internet so, considering the a web service call consist in exchanging 1 or 2 messages with a given size the result coming out from this project could help in the network delay estimation.
As Real test we consider a scenario where a heterogeneous set of clients is performing a web service invocation whit a random input and output size and types:

- Input Type= random between [none, String, double, String and Double]
- Output Type= random between [none, String, double, String and Double]
- Input Size, = random between [0, 10000]
- Output Size, = random between [0, 10000]
- Number of connected clients = [0, 1, 2, 3, 4, 6, 10]

Clients were continuously and randomly choose the input and the output of the remote method and than requesting the service; The CPUs of all the nodes were totally dedicated to the tests where we were collecting and memorizing 5000 samples.
We believe that this second independent set of tests can help the validation of developed estimation techniques with the synthetic dataset without the need of use part of the previous dataset for the validation phase.

6 Test Bed Architecture, Implementation Detail and Experimental Error on the Sample Measures.

This chapter describes the test bed architecture providing all the implementation details that ensure the validity of the collected information.
From an intuitive point of view when the WS client perform a service invocation send also t0, t1, t6, t7 times to a QoS Metric Collector component. In the same time the WS Server reply to the client invocation and send t2, t3, t4, t5 to the Collector too.
The Collector receives the information, correlate the times and construct the interval mentioned in paragraph 4 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 introduce unpredictable overhead.
What is following is a short description of the architecture and a discussion on the error introduced in measures using the developed software.

6.1 Test Bed Architecture.

Figure 3 resume the test bed architecture that has been realized as a P2P system that run in parallel with the web service invocation. As mentioned in the introduction of this chapter different software components have been developed:

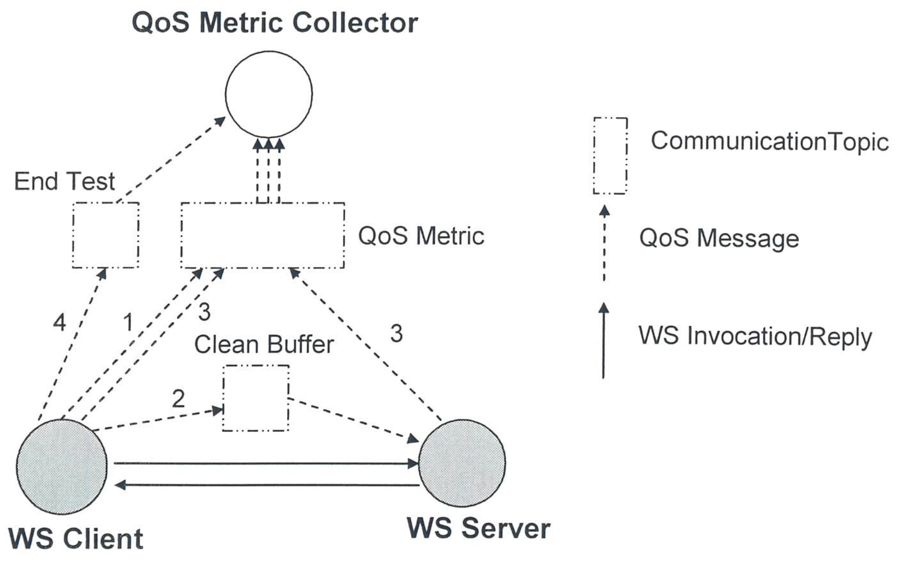


Figure 3: Testbet software architecture

WS-Server that:

- Reply to remote to the client remote call
- Send its time information to the Collector component.
- Synchronize itself with the client in order to let the collector receive coherent time

from both client and server

WS-Client that:

- Remotely invoke the server
- Send its time information to the Collector component
- Synchronize itself with the client in order to let the collector receive coherent time from both client and server
- Notify the QoS Metric Collector when the current test is finishing

QoS Metric Collector that:

- Receive tests description coming from the WS-Client
- Receive and Collect times information coming from both Server and Client
- Receive an test notification related to the test status coming from the WS-Client
- Save all the QoS significant intervals

In order to allow the interaction between the system peer 3 communication channel have been set upped. 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 by figure 4 and are also remarked in figure 3. Firstly the client initiate a new test sending the test description to the Collector; secondly it send a clean message to the server in order to force a clean up of the server information thirdly the real test begin and times information are collected during the web service invocations. Finally, once all the test needed invocations are finished, the Client notifies the collector.

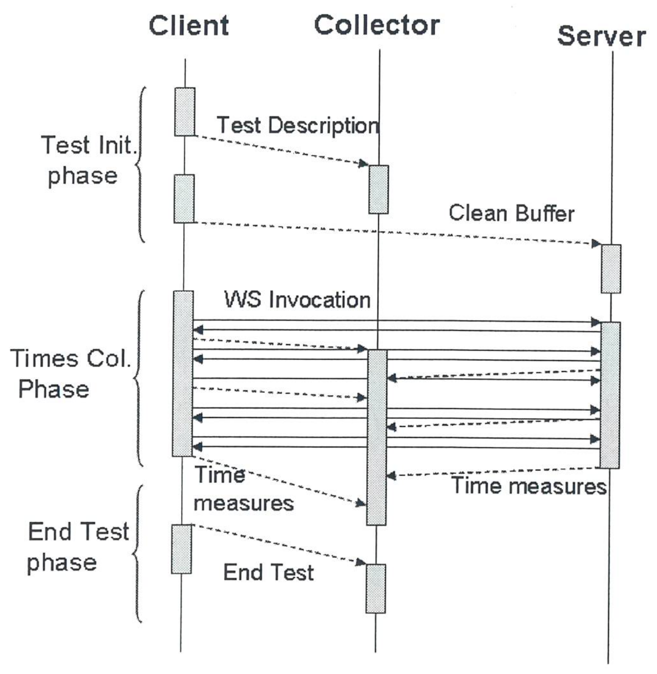


Figure 4: component interaction diagram

6.2 Implementation Details

A critical importance 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,t6 client side and t2,t5 server side are inside the web service engine therefore we was force to chose an open source product for our tests. The source code of Axis (version 1.4) was modified introducing our probe and than 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 [21] was used in order to create the communication channels and it was instruct to send messages in background. Before start the constitution of the dataset, in order to measure the overhead introduced by this modification, was performed a comparison test between a standard web service and the same remote service equipped with the above outlined software and the result was negligible.
As final remark the Collector was realized as a java stand alone application and was instrumented using a JMX [22] library in order to provide on-line information about the tests.

6.3 Experimental Error on the Sample Measures

Is well known that java systems introduce 1ms of incertitude in every measure [15] while the client-server clocks synchronization has a better precision considering that the machines are directly linked by an Ethernet switch [16].
In conclusion, considering that the experienced time intervals are O(100ms) or more we can consider this error negligible.

7 A taste on the collected information

This section presents the obtained experimental results. As mentioned in paragraph 5.3 the data set consists in 2304000 samples (of 9 significant values each) organized in 2304 different tests. We believe that provide 20736 plots related to the samples distribution of each significant value of the entire test bed is just mining less; we prefer show and discuss a subset of the collected raw data in order to let understand the reader what are the effective possibilities of the produced dataset. For the same reason in this chapter we will present and comment just the raw data distribution without any additional analysis that could be performed on top.

7.1 Samples Distribution for Invocation with Big Input and Output

In this first set of plots we consider a remote method invocation with 10000 String and Doubles in input and output. The following 4 figures show the samples distributions of the total invocation time (t7-t0) when the CPU occupancy on both client and server side occupancy varying between 0% and 80% (figure 5-a,b,c,d). The remote method algorithm is empty; in other words the server immediately start the serialization of a pre-generated output, once the deserialization of the input is accomplished,.

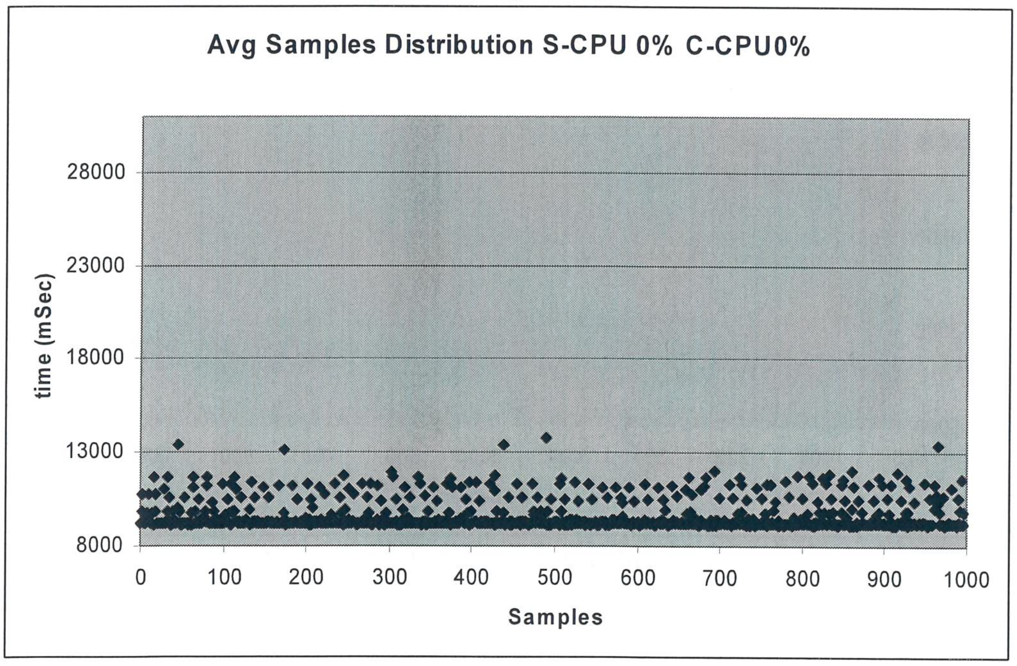


Figure 5a: Sample distribution S-CPU 0% C- CPU 0%

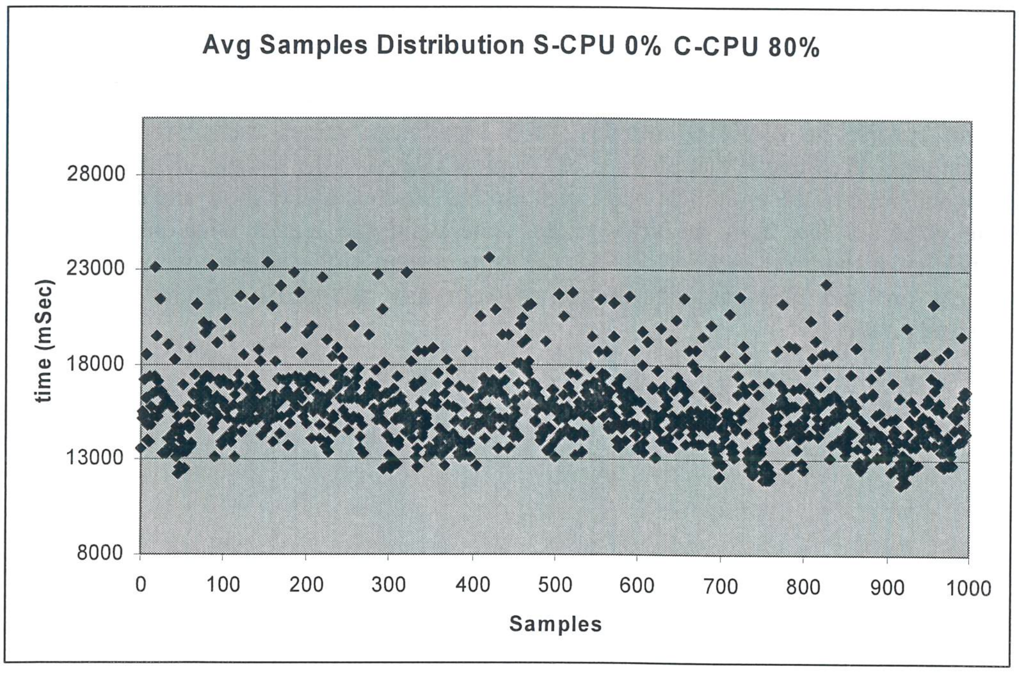


Figure 5b: Sample distribution S-CPU 0% C- CPU 80%

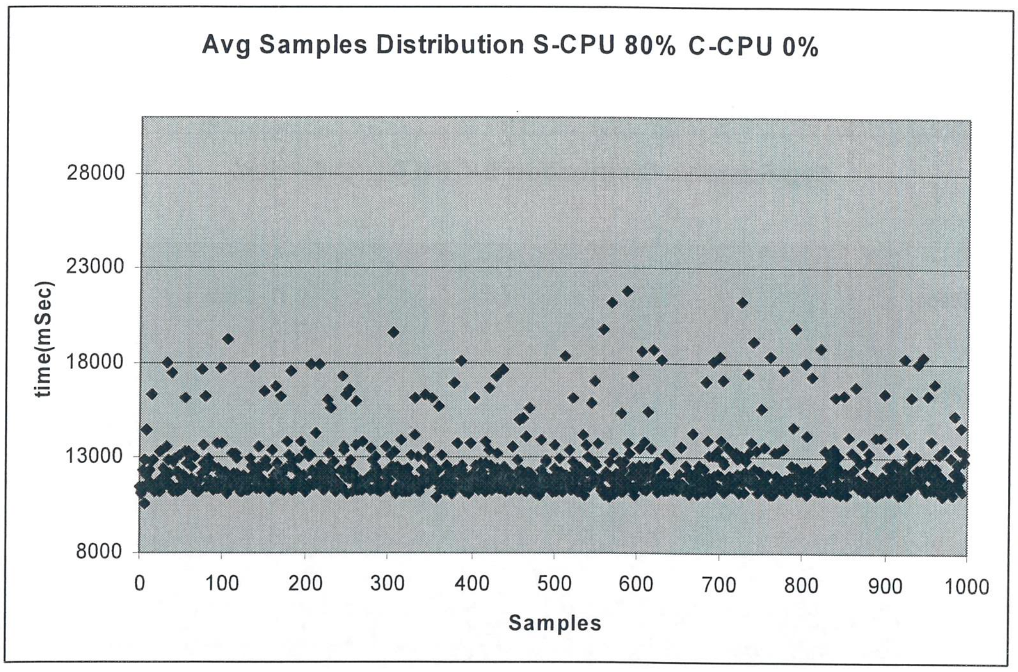


Figure 5c: Sample distribution S-CPU 80% C- CPU 0%

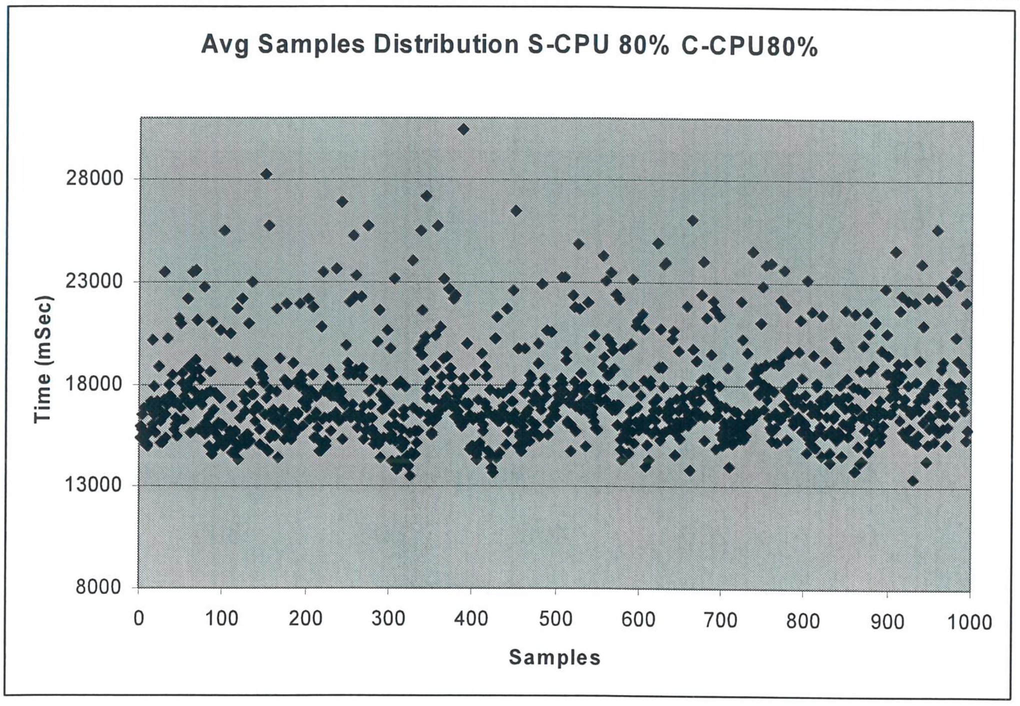


Figure 5d: Sample distribution S-CPU 80% C- CPU 80%

As the intuition suggests, we can note that both the mentioned factors (CPU Client and server side) influence the raw data distributions; more the machines are busy more the average and the standard deviation increase. We can also note that the CPU occupancy client side factor is more influent than the CPU occupancy server side and finally the remote method execution time increase remarkably when both the machines are busy.

7.2 Server De-Serialization time for Different Input Size.

In this example we consider only the server de-serialization time varying the method input. In this particular set of plots the machine was totally dedicated to remote method handling except for the plot 6c where we replicate the experiment showed in 6b with a server occupancy fixed to the 80%.

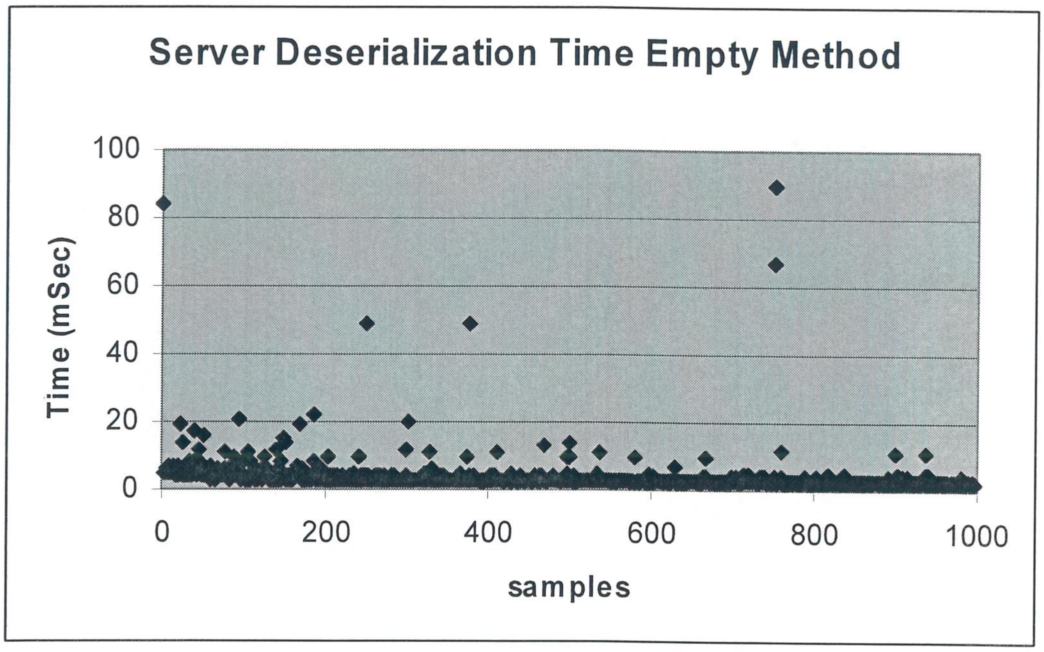


Figure 6a: Sample distribution of the Server Deserialization Time (empty method)

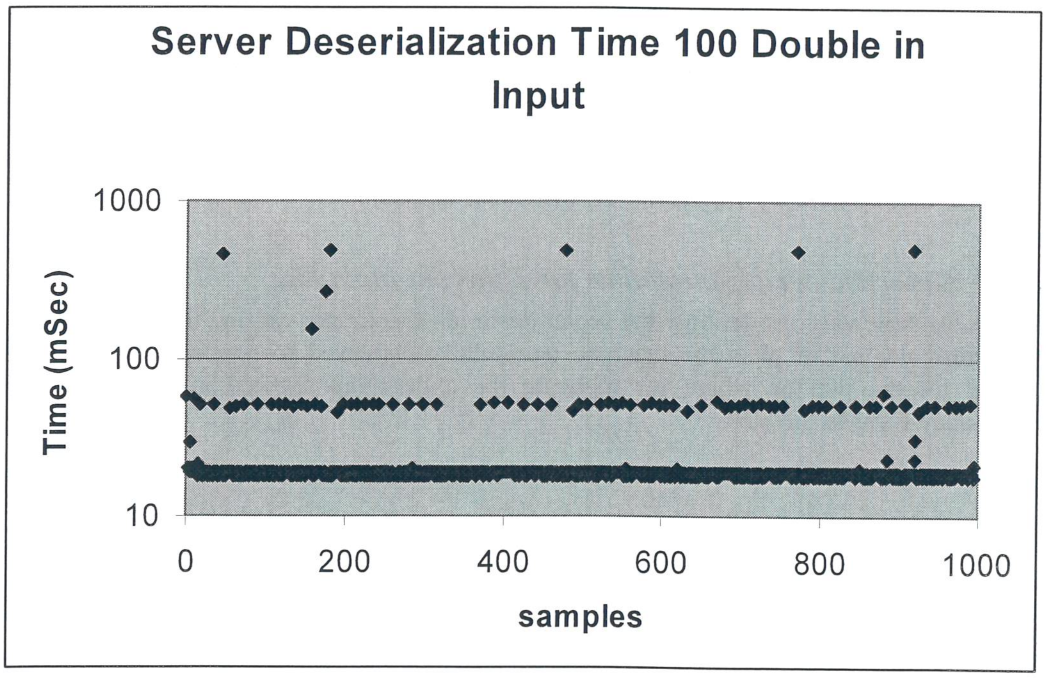


Figure 6b: Sample distribution of the Server Deserialization Time (Input 100 Double)

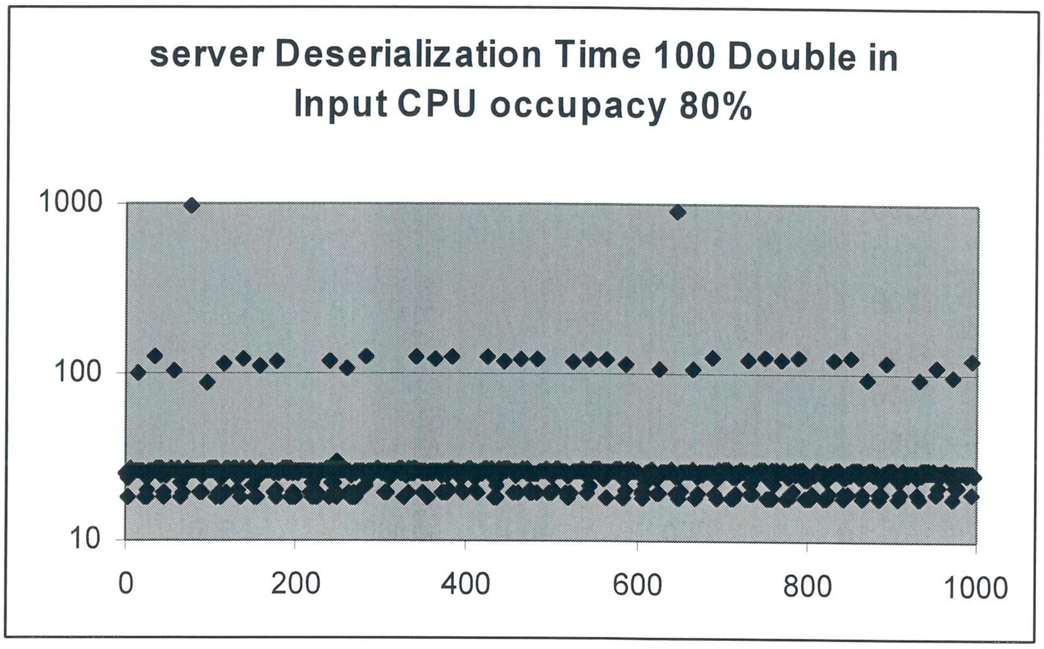


Figure 6c: Sample distribution of the Server Deserialization Time (Input 100 Double) whit server CPU occupancy of 80%

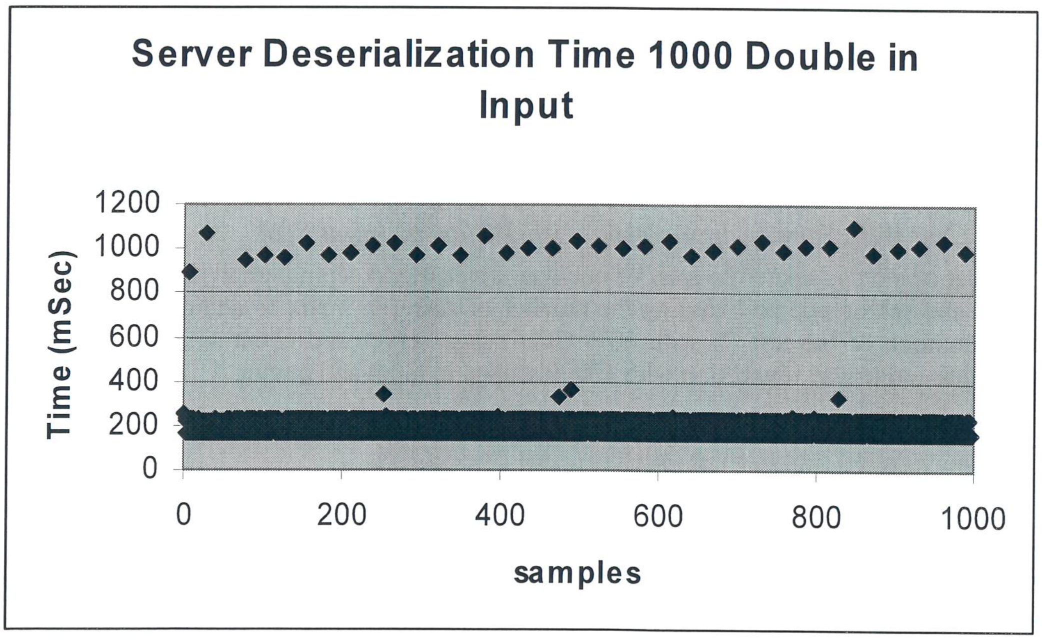


Figure 6d: Sample distribution of the Server Deserialization Time (Input 1000 Double)

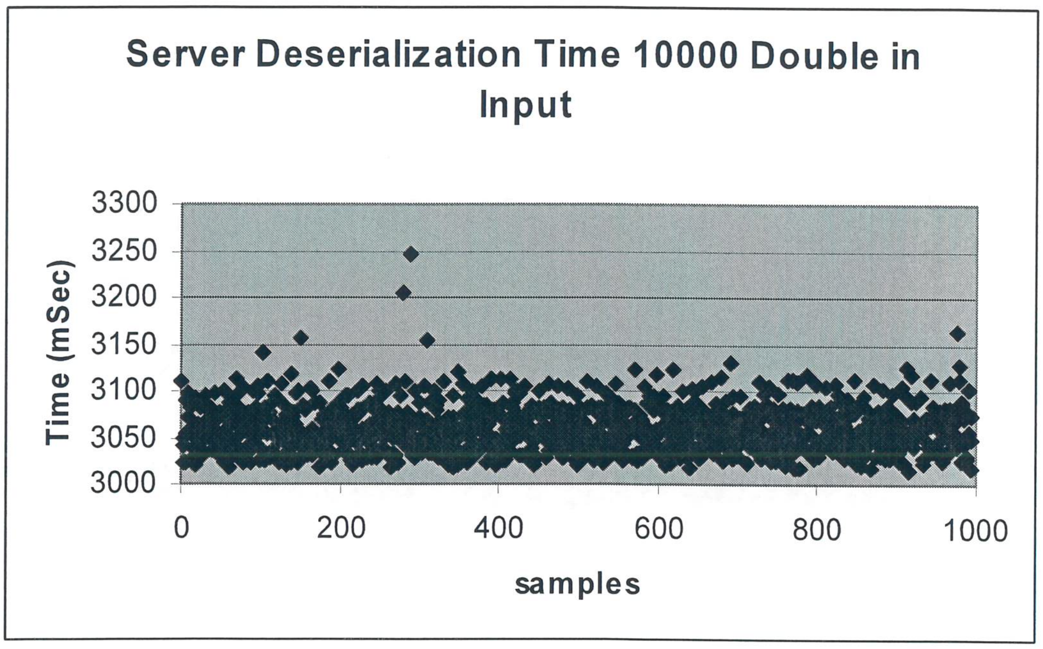


Figure 6e: Sample distribution of the Server Deserialization Time (Input 10000 Double)

As expected the deserialization server side of the SOAP message has a non negligible impact on the remote method execution time. The time needed by the deserialization process increase with the input size. For short input we can also note a consistent number of coherent anomalies typical of cache miss phenomena or to the process migration in a different CPU. We can also note that for big input messages these phenomena disappear. Finally as proved by a comparison between figure 6b and 6c the CPU occupancy server side has a key role in the deserialization process.

7.3 One Way Service Invocation with Different Input Size.

This set of plots is related to a set of one way invocation keeping empty the CPU occupancy client and server side and varying the number of string in input. In addition plot 7c repeats the condition of the test 7b with 80% CPU usage server and client side. Finally plot 7e considers a one way invocation with 10000 strings in input and output.

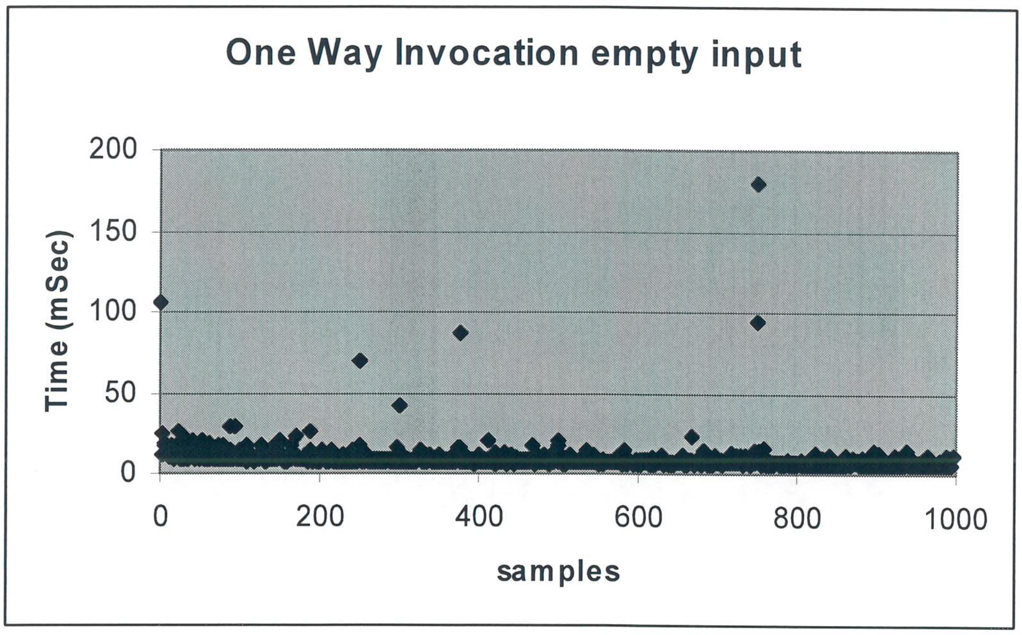


Figure 7a: Sample distribution of a one way invocation of an empty method

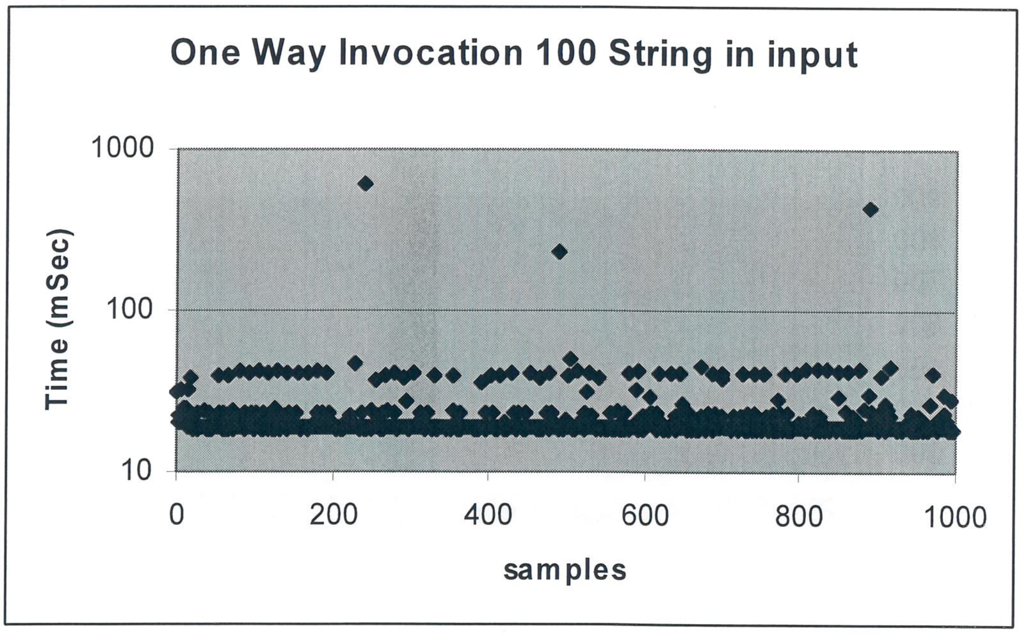


Figure 7b: Sample distribution of a one way invocation (Input 100 String)

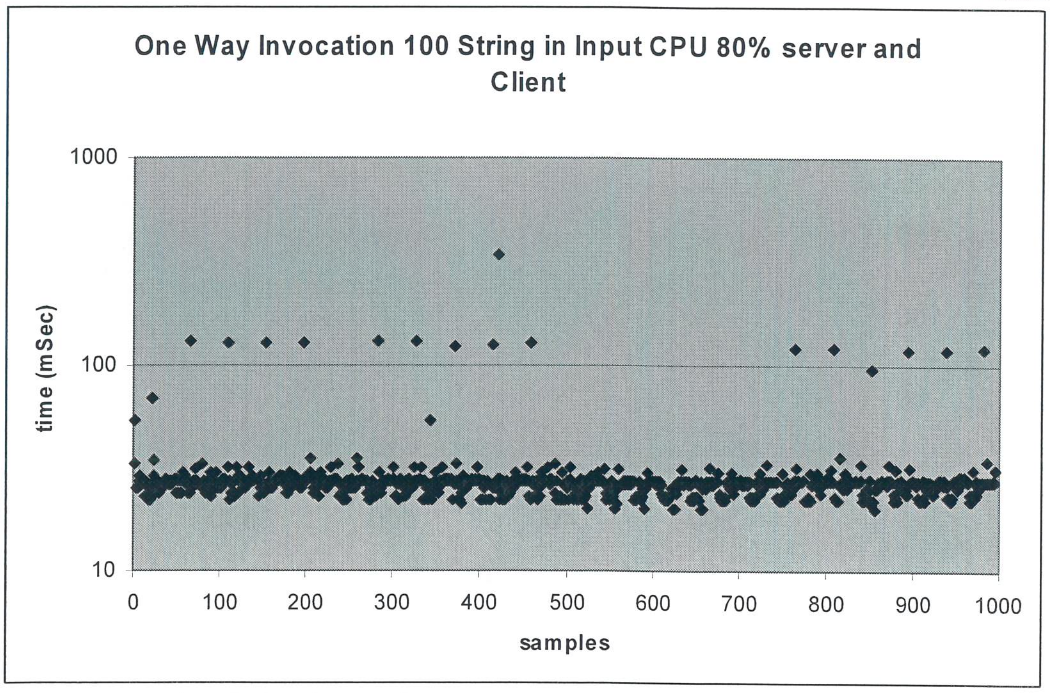


Figure 7c: Sample distribution of a one way invocation (Input 100 String) whit server CPU occupancy of 80%

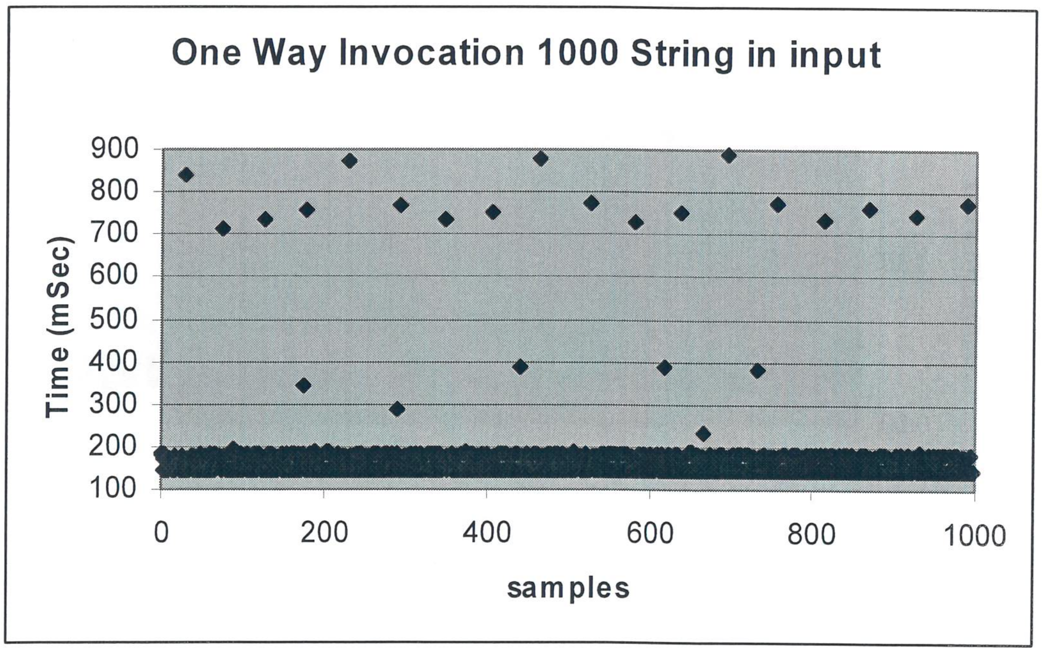


Figure 7d: Sample distribution of a one way invocation (Input 1000 String)

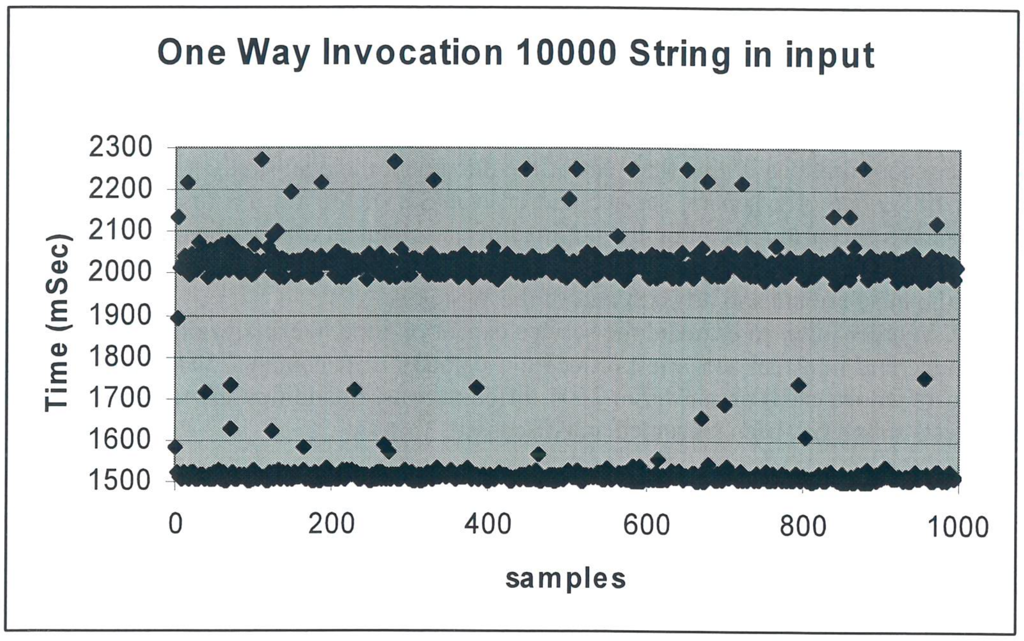


Figure 7d: Sample distribution of a one way invocation (Input 10000 String)

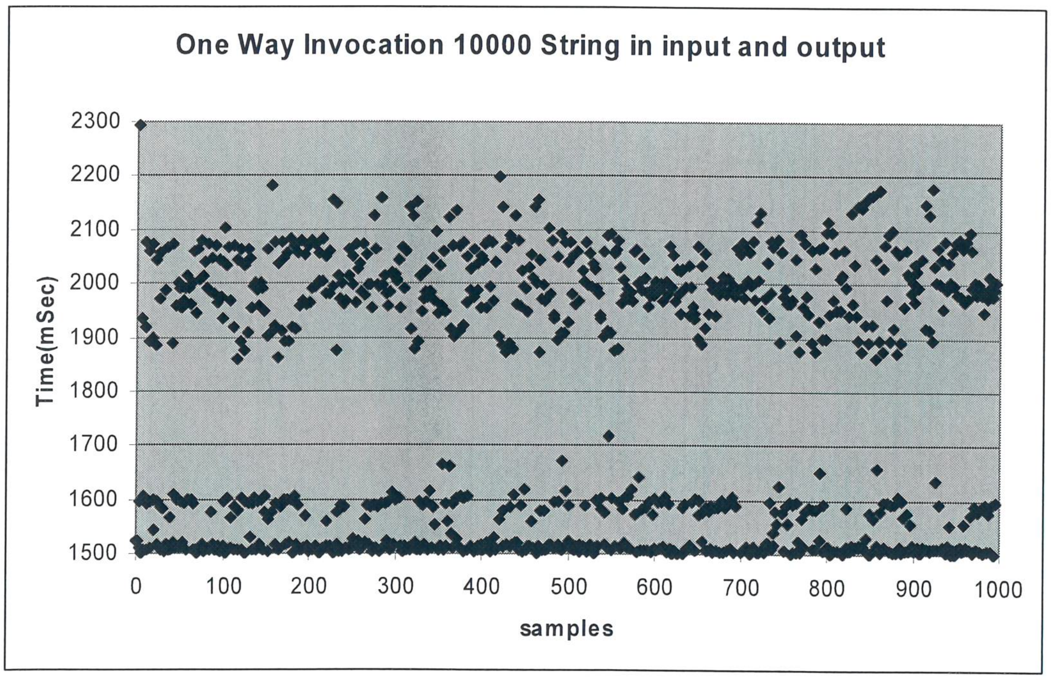


Figure 7e: Sample distribution of a one way invocation (Input and output 10000 String)

As expected the number of string in input deeply influence the one way execution time of a remote method. Like the serialization plots presented in 7.2 also these plots present a phenomenon typical of cache miss or process rescheduling in different CPU. By comparing figure 7b and 7c we can also note that the CPU usage server and client side influence the one way service time. Finally, comparing plots 7d and 7e the output size in a first appreciation do not influence the method execution time while in a more detailed analysis this sentence is not correct.

7.4 Conclusions and Final Remarks on the Data Set

The main contributions of this first part of the document include: firstly a presentation of the quality of service of a generic service and an analysis of the key factors that influence the QoS in a WS scenario. Secondly the outlined of the status of art and the on going activity in this research area and thirdly the identification of the method execution time prediction as one of the most critical and difficult part of the WS QoS.
Finally, in order to try to estimate the service execution time, we analyze the WS behavior in a synthetic and in a real scenario producing a database that contain 2304000 samples (of 9 significant values each) organized in 2304 different tests. In addition the raw data of a really small tests subset has been presented and discussed.

8 Dead Line Estimation Methodology

In this chapter we propose a method that predicts the execution time of a remote service invocation. From a mathematical point of view a remote method invocation represents a time interval where the method will be executed with a given percentile (95th percentile, for example). In other words if

$$f(x)$$

Represent the probability distribution of a statistic variable that describes the service invocation execution time:

$$Y = P(x < \overline{X}) = \int_{-\infty}^{\overline{X}} f(x)$$

Represent the minimum time interval that can guaranteed the execution time with a given probability. Considering that we are trying to estimate a possible dead line an upper bound estimation Y is still an acceptable value while can not be considered a lower bound.
An additional remark is that the probability function must take into account all the factors that are represented in table 2 of paragraph 4.3.
Finally we need to point that the exact probability distribution depends from many factors and it is quite difficult to estimate, so we could try to give an upper bound like the following:

$$P(x < \overline{X}) \leq P(k < \overline{X})$$

Where k is described by a Gaussian distribution larger that the x distribution with a given average (Avg) and standard deviation (sDev)

$$N(\mu, \sigma) \qquad \mu = f(\underline{x}) \qquad \sigma = g(\underline{x})$$

We can note that μ and σ are deterministic function of the key factors (see paragraph 4.3) that influences the WS invocation. If we succeed in the estimation of the average and the standard deviation of the presented Gaussian we will be able to provide an upper bound of the remote method execution time noting that:

$$Y = P(x < \overline{X}) \leq q_{\overline{X}} = g(\underline{x})\Phi_{\overline{X}} + f(\underline{x})$$

Where Φ_x represent the quantile of order X.

Considering that both f(x) and g(x) must be determinate in an empiric way we can not ignore the experimental error due to this estimation, in particular

$$\mu = f_e(x) \pm \varepsilon_{\mu} \qquad \sigma = g_e(\underline{x}) \pm \varepsilon_{\sigma}$$

Again, considering that we want to estimate an upper bound of a time only positive errors must be considered.
In conclusion the final dead line estimation will be:

$$Y_e = g_e(\underline{x})\Phi_{\overline{X}} + \varepsilon_\mu \Phi_{\overline{X}} + f_e(\underline{x}) + \varepsilon_\sigma$$

If some factors influence can not be considered under the client control we need to substitute the worst case scenario into the just presented equation.
As final remark we need to consider factors like the algorithm of the method (G) and the Key input that change the algorithm behavior (H); they must be provided by the server because, as discussed in paragraph 4, web services to not take into account the semantic of the remote methods. Assuming that the remote algorithm execution time is Ra(x) and that it will be provided by the programmers the final estimation of the critical intervals (see paragraph 4.2) will be:

$$t7 - t0 = T_{ex} = g_{ex}(\underline{x})\Phi_{\overline{X}} + \varepsilon_{\mu_{ex}} \Phi_{\overline{X}} + f_{ex}(\underline{x}) + \varepsilon_{\sigma_{ex}} + Ra(\underline{x})$$

$$t3 - t0 = T_{ow} = g_{ow}(\underline{x})\Phi_{\overline{X}} + \varepsilon_{\mu_{ow}} \Phi_{\overline{X}} + f_{ow}(\underline{x}) + \varepsilon_{\sigma_{ow}}$$

$$t4 - t0 = T_{el} = g_{ow}(\underline{x})\Phi_{\overline{X}} + \varepsilon_{\mu_{ow}} \Phi_{\overline{X}} + f_{ow}(\underline{x}) + \varepsilon_{\sigma_{ow}} + Ra(\underline{x})$$

Next paragraphs will present and discuss a methodology for the estimation of the functions according with the experimental data of the dataset.

8.1 Empiric Function Estimation

We would like to consider 2 different approaches to the functions estimation problem. Fist of all we could directly estimate the needed functions from the QoS time factor t7-t0, t4-t0 and t3-t0; as second possibility we could derive the same functions from each single interval of a web service invocation. In other words considering an empty remote method invocation we could have:

$$\mu = f(\underline{x}) = f_e(\underline{x}) \pm \varepsilon_\mu = f_{cs}(\underline{x}) + f_{cn}(\underline{x}) + f_{sd}(\underline{x}) + f_{ss}(\underline{x}) + \\ + f_{sn}(\underline{x}) + f_{dc}(\underline{x}) \pm \varepsilon_{\mu_{cs}} \pm \varepsilon_{\mu_{cn}} \pm \varepsilon_{\mu_{sd}} \pm \varepsilon_{\mu_{ss}} \pm \varepsilon_{\mu_{sn}} \pm \varepsilon_{\mu_{dc}}$$

$$\sigma = g(\underline{x}) = g_e(\underline{x}) \pm \varepsilon_\sigma = g_{cs}(\underline{x}) + g_{cn}(\underline{x}) + g_{sd}(\underline{x}) + g_{ss}(\underline{x}) + \\ + g_{sn}(\underline{x}) + g_{dc}(\underline{x}) \pm \varepsilon_{\sigma_{cs}} \pm \varepsilon_{\sigma_{cn}} \pm \varepsilon_{\sigma_{sd}} \pm \varepsilon_{\sigma_{ss}} \pm \varepsilon_{\sigma_{sn}} \pm \varepsilon_{\sigma_{dc}}$$

With the following functions meaning:

Function	**Meaning**

$f_{cs}(\underline{x})$	Describe the Average of the Client Serialization process
$f_{cn}(\underline{x})$	Describe the Average of the input message transmission process
$f_{sd}(\underline{x})$	Describe the Average of the input deserialization process
$f_{ss}(\underline{x})$	Describe the Average of the Server Serialization process
$f_{sn}(\underline{x})$	Describe the Average of the output message transmission process
$f_{dc}(\underline{x})$	Describe the Average of the output deserialization process
$g_{cs}(\underline{x})$	Describe the Standard Deviation of the Client Serialization process
$g_{cn}(\underline{x})$	Describe the Standard Deviation of the input message transmission process
$g_{sd}(\underline{x})$	Describe the Standard Deviation of the input deserialization process
$g_{ss}(\underline{x})$	Describe the Standard Deviation of the Server Serialization process
$g_{sn}(\underline{x})$	Describe the Standard Deviation of the output message transmission process
$g_{dc}(\underline{x})$	Describe the Standard Deviation of the output deserialization process

Table 3: Functions break down description descriptions

Where for one way message types (t4-t0 and t3-t0):

$$f_{ss}(\underline{x}) = f_{sn}(\underline{x}) = f_{dc}(\underline{x}) = g_{ss}(\underline{x}) = g_{sn}(\underline{x}) = g_{dc}(\underline{x}) = 0$$

because the output transmission do not influence the execution time.

The second presented approach can allow an interval composition avoiding clock synchronization problems for one way messages type and the possibilities to perform independent measures for the clients, servers and networks channels computing the estimation time just as a linear composition of factors.
We need to remark that we will consider an upper bound of each real function so the dead line computed in this second way probably will be larger.

8.2 2^k factorial analysis design on the dataset

In this paragraph we will design a factorial analysis [23] around the experimental data in order to present a linear regression model that give an approximation of the function g(x) and f(x) as discuss in the previous paragraph.
According with the test set description presented in paragraph 5.3 we consider the following factors:

A= Number of String in Input of the method	[0.........10000]
B= Number of String in Output of the method	[0..........10000]
C= Number of Double in Input of the method	[0..........10000]
D= Number of Double in Output of the method	[0..........10000]
E= CPU Usage Server Side	[0..........80%]
F= CPU Usage Client Side	[0..........80%]

That generates the following full 2^6 factor design composed by 64 tests:

Run	A	B	C	D	E	F	labels
1	-	-	-	-	-	-	(1)
2	+	-	-	-	-	-	a
3	-	+	-	-	-	-	b
4	+	+	-	-	-	-	ab
5	-	-	+	-	-	-	c
6	+	-	+	-	-	-	ac
7	-	+	+	-	-	-	bc
8	+	+	+	-	-	-	abc
9	-	-	-	+	-	-	d
10	+	-	-	+	-	-	ad
11	-	+	-	+	-	-	bd
12	+	+	-	+	-	-	abd
13	-	-	+	+	-	-	cd
14	+	-	+	+	-	-	acd
15	-	+	+	+	-	-	bcd
16	+	+	+	+	-	-	abcd
17	-	-	-	-	+	-	e
18	+	-	-	-	+	-	ae
19	-	+	-	-	+	-	be
20	+	+	-	-	+	-	abe
21	-	-	+	-	+	-	ce
22	+	-	+	-	+	-	ace
23	-	+	+	-	+	-	bce
24	+	+	+	-	+	-	abce
25	-	-	-	+	+	-	de
26	+	-	-	+	+	-	ade
27	-	+	-	+	+	-	bde
28	+	+	-	+	+	-	abde
29	-	-	+	+	+	-	cde
30	+	-	+	+	+	-	acde
31	-	+	+	+	+	-	bcde
32	+	+	+	+	+	-	abcde
33	-	-	-	-	-	+	f
34	+	-	-	-	-	+	af
35	-	+	-	-	-	+	bf
36	+	+	-	-	-	+	abf
37	-	-	+	-	-	+	cf
38	+	-	+	-	-	+	acf
39	-	+	+	-	-	+	bcf
40	+	+	+	-	-	+	abcf
41	-	-	-	+	-	+	df
42	+	-	-	+	-	+	adf
43	-	+	-	+	-	+	bdf
44	+	+	-	+	-	+	abdf
45	-	-	+	+	-	+	cdf
46	+	-	+	+	-	+	acdf
47	-	+	+	+	-	+	bcdf
48	+	+	+	+	-	+	abcdf
49	-	-	-	-	+	+	ef
50	+	-	-	-	+	+	aef

51	-	+	-	-	+	+	bef
52	+	+	-	-	+	+	abef
53	-	-	+	-	+	+	cef
54	+	-	+	-	+	+	acef
55	-	+	+	-	+	+	bcef
56	+	+	+	-	+	+	abcef
57	-	-	-	+	+	+	def
58	+	-	-	+	+	+	adef
59	-	+	-	+	+	+	bdef
60	+	+	-	+	+	+	abdef
61	-	-	+	+	+	+	cdef
62	+	-	+	+	+	+	acdef
63	-	+	+	+	+	+	bcdef
64	+	+	+	+	+	+	abcdef

Table 4: Full factorial analysis

The functions that we would like to estimate are:

- Average Method Invocation
- Standard deviation Method Invocation
- Average One Way Invocation
- Standard deviation One Way Invocation
- Average Computation Remote Site Executed
- Standard deviation Computation Remote Site Executed
- all the functions presented in table 3

Considering that the dataset was build considering that all the remote method execution was empty there are no difference between the intervals t4-t0 and t3-t0. In addition the network transmission (O(1ms)) can be considered negligible compared to others web service invocation process (O(100ms) like the serialization and the deserialization.
Finally once the key inplicant will be determinate we would like to understand if the functions to estimate can fit in the following regression model:

$$y = \beta_0 + \beta_1 x_A + \beta_2 x_B + \beta_3 x_C + \beta_4 x_D + \beta_5 x_E + \beta_6 x_A x_B + \beta_7 x_B x_E + \beta_8 x_C x_E + \\ + \beta_9 x_D x_E + \beta_{10} x_F + \beta_{11} x_A x_F + \beta_{12} x_B x_F + \beta_{13} x_C x_F + \beta_{14} x_D x_F \pm \varepsilon$$

β_i are constant coefficients and x_i represent a generic value between [-1,+1] where -1 represent the lowest value of the considered interval factor and +1 the biggest.
This particular regression model is very useful because the coefficient have the following intuitive meaning:

Coefficient	Average
β_0	Constant
β_1	String Input Influence
β_2	String Output Influence
β_3	Double Input Influence
β_4	Double Output Influence
β_5	CPU Load Server Side Influence
β_6	How CPU Load server side influence the string deserialization
β_7	How the CPU load server side influence the string serialization
β_8	How CPU Load server side influence the double deserialization
β_9	How the CPU load server side influence the double serialization
β_{10}	CPU Load Client Side Influence
β_{11}	How CPU Load client side influence the string deserialization
β_{12}	How the CPU Load client side influence the string serialization
β_{13}	How CPU Load client side influence the double deserialization
β_{14}	How the CPU Load client side influence the double serialization
ε	Error

Table 5: Linear regression coefficients description

What is follow are the factor analysis and the regression model fitting based on the estimated key factors. After that a regression model with the above mentioned function was computed and, if the two functions show the same residuals, just this last one, which is more compressible, is presented.
We will show the complete analysis just for the Total Execution time (paragraph 8.3) wile for all other factors we will present only the results leaving intermediate computation in appendix II.

8.3 Gaussian upper bound of a Sample distribution

For each of the 64 tests samples average and standard deviation was computed using the following standard estimators:

$$\overline{\mu} = \frac{1}{n}(x_1 + x_s + \cdots + x_n)$$

$$\overline{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{\mu})^2}$$

After that we run the following hypothesis test:

$H_0 = N(\overline{\mu} + \varepsilon, \overline{\sigma} + \xi)$ is a larger distribution than the one experience during the test

Where ε, ξ are the minimum non negative numbers that allow the success of the hypothesis test. The just determinate values was used as tests values for the contrast analysis and the regression model estimation

8.4 Total Execution Time Estimation

The hypothesis test was satisfied with $\varepsilon,\xi=0$ and the following table resume the contrast and the sums of square of each factor for both average and standard deviation

Factor	AVG Cont. x10 5	SS x108	Red. SS	SDev Contrast x10 4	SS x10 6	Red. SS
a	0.7751	0.9387	0.9387	1.8293	5.2288	5.2288
b	0.8445	1.1144	1.1144	0.5331	0.4440	0.4440
ab	-0.0677	0.0072	0	-1.0265	1.6463	1.6463
c	1.8746	5.4908	5.4908	1.6064	4.0322	4.0322
ac	0.0724	0.0082	0	1.2243	2.3420	2.3420
bc	-0.0434	0.0029	0	-1.0314	1.6621	1.6621
abc	-0.0781	0.0095	0	-0.8466	1.1200	1.1200
d	0.9356	1.3678	1.3678	1.6363	4.1836	4.1836
ad	0.0565	0.0050	0	0.1765	0.0487	0
bd	-0.0028	0.0000	0	-0.1600	0.0400	0
abd	-0.0007	0.0000	0	0.0873	0.0119	0
cd	0.0618	0.0060	0	0.0887	0.0123	0
acd	0.0361	0.0020	0	0.3341	0.1745	0
bcd	0.0173	0.0005	0	0.1173	0.0215	0
abcd	-0.0069	0.0001	0	-0.0313	0.0015	0
e	0.3057	0.1460	0	0.5529	0.4776	0.4776
ae	0.0983	0.0151	0	0.6345	0.6291	0.6291
be	0.0463	0.0033	0	-0.2737	0.1170	0
abe	0.0182	0.0005	0	-0.2682	0.1124	0
ce	0.1651	0.0426	0	0.5397	0.4551	0.4551
ace	0.0215	0.0007	0	0.5662	0.5010	0.5010
bce	0.0286	0.0013	0	-0.1623	0.0412	0
abce	0.0285	0.0013	0	-0.1531	0.0366	0
de	0.0734	0.0084	0	0.2396	0.0897	0
ade	0.0252	0.0010	0	0.2506	0.0981	0
bde	-0.0085	0.0001	0	-0.0197	0.0006	0
abde	-0.0035	0.0000	0	-0.0116	0.0002	0
cde	-0.0113	0.0002	0	0.1517	0.0360	0
acde	0.0141	0.0003	0	0.2584	0.1043	0
bcde	-0.0495	0.0038	0	-0.2318	0.0839	0
abcde	-0.0068	0.0001	0	-0.0718	0.0081	0
f	0.8019	1.0048	1.0048	1.6066	4.0333	4.0333
af	0.0497	0.0039	0	-0.4161	0.2705	0.2705
bf	0.2982	0.1389	0	1.0788	1.8184	1.8184
abf	0.0126	0.0002	0	0.3477	0.1889	0
cf	0.2399	0.0900	0	-0.1031	0.0166	0
acf	0.0006	0.0000	0	-0.3850	0.2316	0
bcf	0.0355	0.0020	0	0.2604	0.1060	0
abcf	0.0061	0.0001	0	0.3381	0.1786	0
df	0.2994	0.1401	0	0.9519	1.4157	1.4157
adf	0.0488	0.0037	0	0.3412	0.1819	0
bdf	-0.0308	0.0015	0	-0.4191	0.2745	0.2745
abdf	-0.0251	0.0010	0	-0.3624	0.2053	0
cdf	0.0624	0.0061	0	0.2386	0.0889	0
acdf	0.0415	0.0027	0	0.3401	0.1808	0
bcdf	-0.0053	0.0000	0	-0.1735	0.0470	0

abcdf	-0.0253	0.0010	0	-0.2943	0.1353	0
ef	-0.0599	0.0056	0	-0.2455	0.0942	0
aef	0.0063	0.0001	0	-0.0952	0.0142	0
bef	-0.0308	0.0015	0	-0.0874	0.0119	0
abef	-0.0118	0.0002	0	-0.0522	0.0043	0
cef	-0.0345	0.0019	0	-0.0652	0.0066	0
acef	0.0218	0.0007	0	0.0115	0.0002	0
bcef	-0.0045	0.0000	0	-0.0133	0.0003	0
abcef	0.0055	0.0000	0	0.0112	0.0002	0
def	-0.0044	0.0000	0	0.0988	0.0152	0
adef	-0.0074	0.0001	0	0.1085	0.0184	0
bdef	0.0053	0.0000	0	-0.1273	0.0253	0
abdef	0.0167	0.0004	0	-0.0725	0.0082	0

Table 6: Contrast and Sum of Square analysis for the total execution time linear regression

The regression model with the estimate key factors and the one presented in paragraph 8.2 present the same residual analysis so we decide to use this second for future analysis. As final result the regression model is represent by the following table:

Coefficient	Average	Standard Deviation
β_0	6788.1	970.5104
β_1	1211.1	285.8330
β_2	1319.6	83.2913
β_3	2929.1	251.0055
β_4	1461.9	255.6742
β_5	477.6	86.3843
β_6	153.5	99.1425
β_7	72.3	-42.7626
β_8	258	84.3240
β_9	114.7	37.4323
β_{10}	1253.0	251.0383
β_{11}	77.7	-65.0114
β_{12}	465.9	168.5594
β_{13}	374.9	-16.1034
β_{14}	467.8	148.7315
ε	+- 1000	+- 1000

Table 7: total execution time linear regression coefficients

While the following figure show the residual analysis for the Average formula

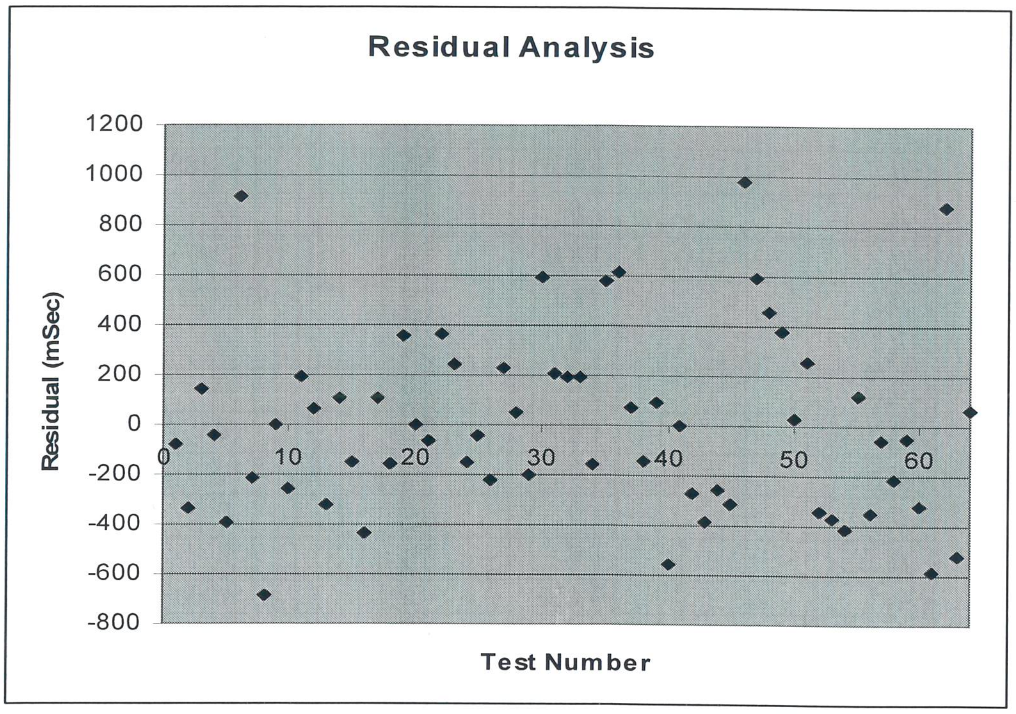


Figure 8: Residual analysis for the total execution time linear regression

As we can see the experimental error is not negligible compared to values that the functions assume, especially for invocation with short input and output.

8.5 A Special Case: Server Deserialization

For this particular formula computation the linear regression of the average suggested by the contrast analysis is different that the one presented in paragraph 8.3 therefore 2 possible regressions are presented. For future analysis we decide to keep the second formula considering that the error introduced by this simplification will be negligible compared with the final results.

$$y = \beta_0 + \beta_1 x_A + \beta_2 x_B + \beta_{12} x_A x_B + \beta_3 x_C + \beta_{13} x_A x_C + \beta_{23} x_B x_C + \beta_{123} x_A x_B x_c + \beta_4 x_D$$
$$+ \beta_5 x_E + \beta_6 x_A x_B + \beta_7 x_B x_E + \beta_8 x_C x_E + \beta_{135} x_A x_C x_E + \beta_9 x_D x_E + \beta_{10} x_F +$$
$$+ \beta_{11} x_A x_F + \beta_{12} x_B x_F + \beta_{13} x_C x_F + \beta_{14} x_D x_F + \varepsilon$$

Coefficient	Average	Standard Deviation
β_0	2704.3	463.7945
β_1	872.8	400.4378
β_2	-115.1	-142.0726
β_{12}	-109.2	-142.8330
β_3	1907.9	280.6679

β_{13}	79.8	224.9553
β_{23}	-121.5	-154.7009
β_{123}	-115.4	-151.8307
β_4	0	0
β_5	424.5	136.3823
β_6	140.6	119.7188
β_7	0	0
β_8	301.7	112.2975
β_{135}	18.1	97.2722
β_9	0	0
β_{10}	0	0
β_{11}	0	0
β_{12}	0	0
β_{13}	0	0
β_{14}	0	0
ε	+- 850	+- 1000

Table 8: Linear regression coefficient with the key factors analysis

$$y = \beta_0 + \beta_1 x_A + \beta_2 x_B + \beta_3 x_C + \beta_4 x_D + \beta_5 x_E + \beta_6 x_A x_B + \beta_7 x_B x_E + \beta_8 x_C x_E + \\ + \beta_9 x_D x_E + \beta_{10} x_F + \beta_{11} x_A x_F + \beta_{12} x_B x_F + \beta_{13} x_C x_F + \beta_{14} x_D x_F \pm \varepsilon$$

Coefficient	Average	Standard Deviation
β_0	2704.3	463.7945
β_1	872.8	400.4378
β_2	0	0
β_3	1907.9	280.6679
β_4	0	0
β_5	424.5	136.3823
β_6	140.6	119.7188
β_7	0	0
β_8	301.7	112.2975
β_9	0	0
β_{10}	0	0
β_{11}	0	0
β_{12}	0	0
β_{13}	0	0
β_{14}	0	0
ε	+- 1000	+- 1000

Table 9: Linear regression coefficient with the standard linear regression model

8.6 Different f(x) estimations

The following table resumes the different functions estimation related to the average function discussed in paragraph 8.1 while the last 2 colons try to describe the average of the total execution time (t7-t0) and the one way times (t4-t0, t3-t0) by computing the influence of each interval. As anticipate the error is bigger that the one experienced using a direct computation.

	Total Exec Time	**One way Time**	**Client Ser**	**Server Deser**	**Server Ser**	**Client Deser**	**Comp Total Exec Time**	**Comp One way Time**
β_0	6788.1	4101.6	1396.6	2704.3	658.8124	2024.2	6783.9124	4100.9
β_1	1211.1	1221.5	348.6	872.8	0	0	1221.4	1221.4
β_2	1319.6	0	0	0	298.3185	1065.9	1364.2185	0
β_3	2929.1	2934.7	1026.7	1907.9	0	0	2934.6	2934.6
β_4	1461.9	0	0	0	362.8475	987.7	1350.5475	0
β_5	477.6	371.8	0	424.5	124.0629	0	548.5629	424.5
β_6	153.5	136.6	0	140.6	0	0	140.6	140.6
β_7	72.3	0	0	0	58.2112	0	58.2112	0
β_8	258	254.6	0	301.7	0	0	301.7	301.7
β_9	114.7	0	0	0	68.9741	0	68.9741	0
β_{10}	1253.0	531.9	525.4	0	0	719.2	1244.6	525.4
β_{11}	77.7	107	124.3	0	0	0	124.3	124.3
β_{12}	465.9	0	0	0	0	389	389	0
β_{13}	374.9	390.3	395.2	0	0	0	395.2	395.2
β_{14}	467.8	0	0	0	0	356.6	356.6	0
ε	+- 1000	+- 1000	+- 500	+- 1000	+- 200	+- 500	+- 2200	+- 1500

Table 10: coefficients for the linear regression estimation of different f(x)

8.7 Different g(x) estimations

The following table resumes the different functions estimation related to the standard deviation function as discussed in paragraph 8.1; we can outline the same observations presented in the previous paragraph

	Total Exec Time	**One way Time**	**Client Ser**	**Server Deser**	**Server Ser**	**Client Deser**	**Comp Total Exec Time**	**Comp One way Time**
β_0	970.5104	652.5936	352.0763	463.7945	71.1530	526.4341	1413.4579	815.8708
β_1	285.8330	366.4459	73.9326	400.4378	0	0	474.3704	474.3704
β_2	83.2913	0	0	0	39.1351	226.0633	265.1984	0
β_3	251.0055	397.8573	193.0729	280.6679	0	0	473.7408	473.7408
β_4	255.6742	0	0	0	31.7847	230.6709	262.4556	0
β_5	86.3843	104.7193	0	136.3823	36.6325	0	173.0148	136.3823
β_6	99.1425	113.2111	0	119.7188	0	0	119.7188	119.7188
β_7	-42.7626	0	0	0	17.2659	0	17.2659	0
β_8	84.3240	83.7679	0	112.2975	0	0	112.2975	112.2975
β_9	37.4323	0	0	0	16.2223	0	16.2223	0
β_{10}	251.0383	91.6492	171.6536	0	0	236.4690	408.1226	171.6536
β_{11}	-65.0114	-45.0087	31.4990	0	0	0	31.4990	31.4990
β_{12}	168.5594	0	0	0	0	112.1133	112.1133	0
β_{13}	-16.1034	43.8107	105.9269	0	0	0	105.9269	105.9269
β_{14}	148.7315	0	0	0	0	109.0708	109.0708	0
ε	+- 1000	+- 1000	+- 500	+- 1000	+- 200	+- 450	+- 2150	+- 1500

Table 11: coefficients for the linear regression estimation of different g(x)

8.8 Dead line function estimation

The combinations of the linear regressions presented in the previous 2 paregrpash according with the last formula of paragraph 8 are presented in the following table:

	Total Exec Time 95	One way Time 95	Comp Total Exec Time 95	Comp One way Time 95	Total Exec Time 975	One way Time 975	Comp Total Exec Time 975	Comp One way Time 975
β_0	8384.4	5175.0	9108.8	5442.9	8690.3	5380.7	9554.2	5700.0
β_1	1681.3	1824.3	2001.7	2001.7	1771.3	1939.7	2151.1	2151.1
β_2	1456.6	0	1800.4	0.0	1482.8	0.0	1884.0	0.0
β_3	3342.0	3589.1	3713.8	3713.8	3421.1	3714.5	3863.1	3863.1
β_4	1882.4	0	1782.2	0.0	1963.0	0.0	1865.0	0.0
β_5	619.7	544.0	833.1	648.8	646.9	577.0	887.7	691.8
β_6	316.6	322.8	337.5	337.5	347.8	358.5	375.2	375.2
β_7	2.0	0	86.6	0.0	-11.5	0.0	92.1	0.0
β_8	396.7	392.4	486.4	486.4	423.3	418.8	521.8	521.8
β_9	176.3	0	95.7	0.0	188.1	0.0	100.8	0.0
β_{10}	1665.9	682.6	1915.9	807.7	1745.0	711.5	2044.5	861.8
β_{11}	-29.2	33.0	176.1	176.1	-49.7	18.8	186.0	186.0
β_{12}	743.2	0	573.4	0.0	796.3	0.0	608.7	0.0
β_{13}	348.4	462.4	569.4	569.4	343.3	476.2	602.8	602.8
β_{14}	712.4	0	536.0	0.0	759.3	0.0	570.4	0.0
ε	+- 2644	+- 2644	+- 5736	+- 3967	+- 2960	+- 2960	+- 6413	+- 4439

Table 12: coefficients for the linear regression for the dead line estimation functions

Where β_i represent the coefficient of the following formula

$$y = \beta_0 + \beta_1 x_A + \beta_2 x_B + \beta_3 x_C + \beta_4 x_D + \beta_5 x_E + \beta_6 x_A x_B + \beta_7 x_B x_E + \beta_8 x_C x_E + \\ + \beta_9 x_D x_E + \beta_{10} x_F + \beta_{11} x_A x_F + \beta_{12} x_B x_F + \beta_{13} x_C x_F + \beta_{14} x_D x_F + \varepsilon$$

The first and the second colon present the total execution time and the one way time functions with a directly estimation and a 95 percentile while the third and the 4th present the same results estimated with the interval decomposition technique presented in paragraph 8.1. Finally the last 4 colons present the same results of previous with the 97.5 percentile.
Next chapter will prove that using these functions we will be able to predict the method execution time of a web service in both a synthetic and a concurrent scenario where the server is not overloaded.

9 Web Service QoS Proof of Concept

In this chapter we will show the prediction capability of techniques discussed in chapter 8. We will validate the methodology using the second independent part of the dataset presented in paragraph 5.3 in order to use different samples than the one used for build the dead line estimators. In addition, in order to validate the methodology in the worst case scenario we will assume that the client CPU Load is always bigger than 80%.

9.1 Proof of concept in a synthetic scenario

Considering that in a real scenario it is impossible maintain the server CPU usage constant during the entire remote method invocation we decide test our predictor considering the case where the server is in the worst case scenario. In other words we consider the case where the server CPU usage from different service is always 80%. In addition we analyze the behaviors of predictors in the case where the server is completely empty in order to understand how much impact this wrong assumption.
We need to point out that via an agreement between the client and the server [25] a client could totally reserve the remote machine in order to be sure that the server CPU occupancy is always 0% and be able to build a more accurate a short dead line.

The test was computed using all 5000 samples of the test bed counting the number of dead line (DL) miss. As expected in both cases less than the 5% and 2.5% of dead line miss was experience due to the upper bound that we was forced to consider in the dead line functions estimations.
Instead of plot all the 5000 samples the following 2 figure represent a samples subset that show the behaviors of the estimators, in both, one way and end to end method invocation, in a scenario where the client CPU load is 80% (x_F=1) while the server is completely unloaded (x_E=-1).

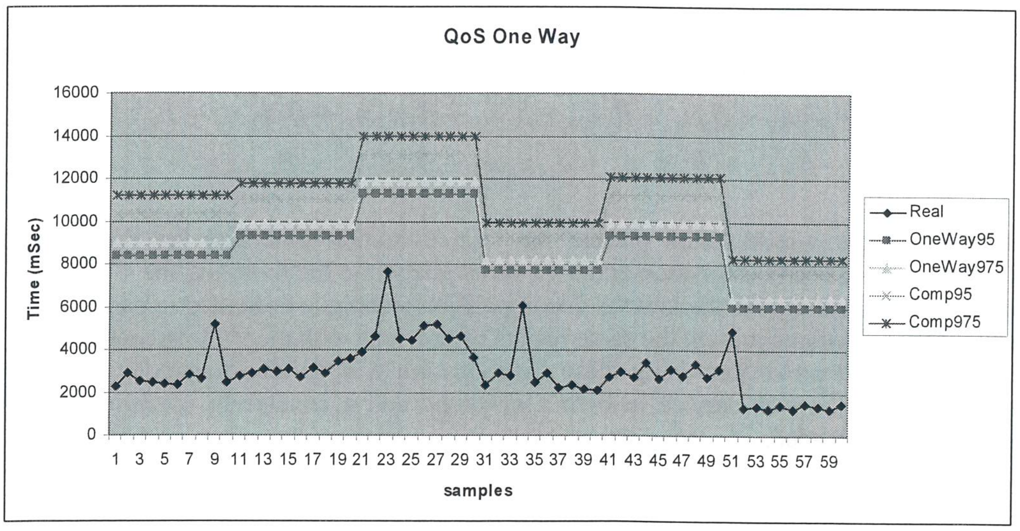


Figure 9a: One Way Estimation CPU server 0% CPU Client 80%

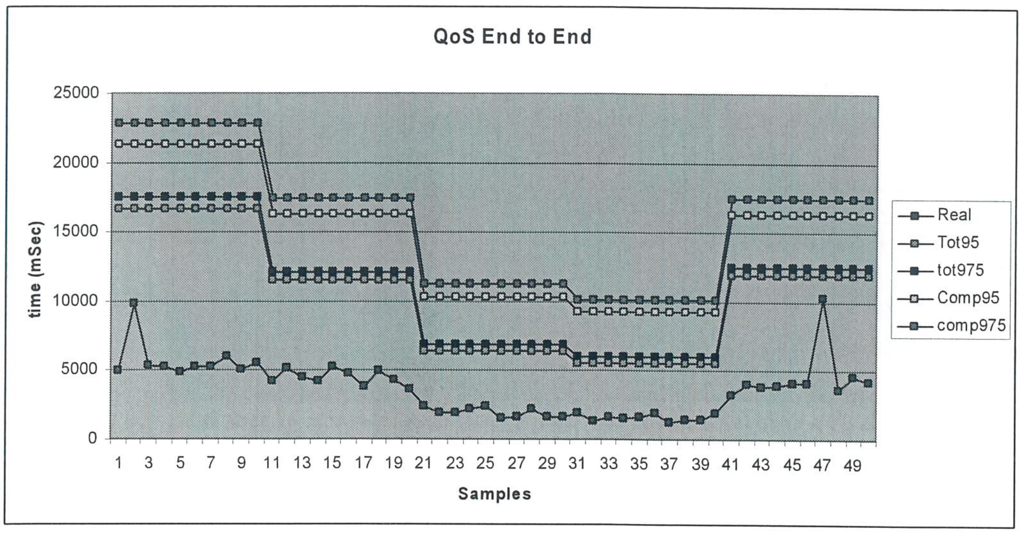


Figure 9a: End To End Estimation CPU server 0% CPU Client 80%

The "Real" line represent the experience invocation time while the "total95" and "total975" plots represents the dead line estimation using a direct computation of the linear regression with a quantile of 95 and 97.5 respectively . Finally "comp95" and "comp975" lines show the dead line estimations using the interval decomposition technique.
This second set of plots represents where the CPU usage server side was forced to be constant at 80% (x_E=1). As we can note the dead line start to be more close to the real system behavior but still the number DL misses are less than the expected 5% and 2.5%.

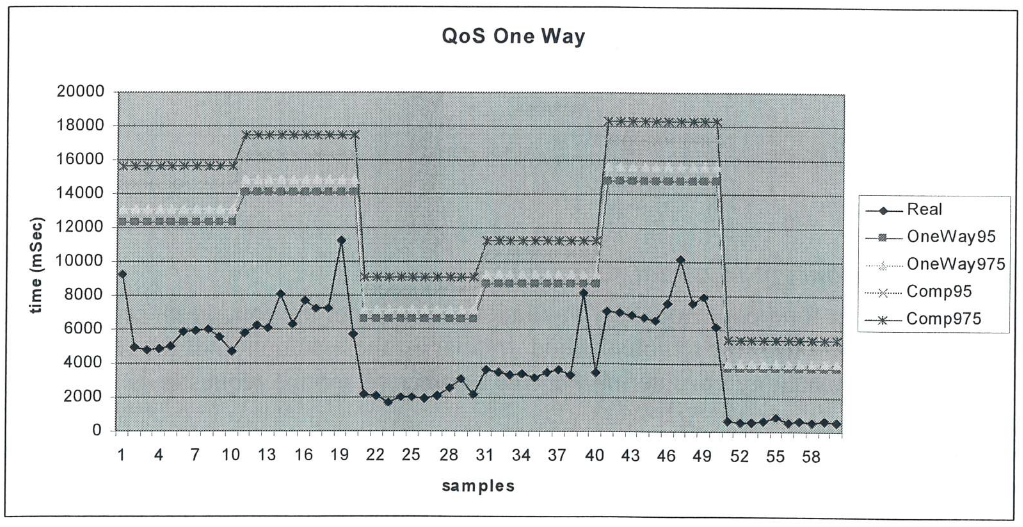


Figure 10a: One Way Estimation CPU server 80% CPU Client 80%

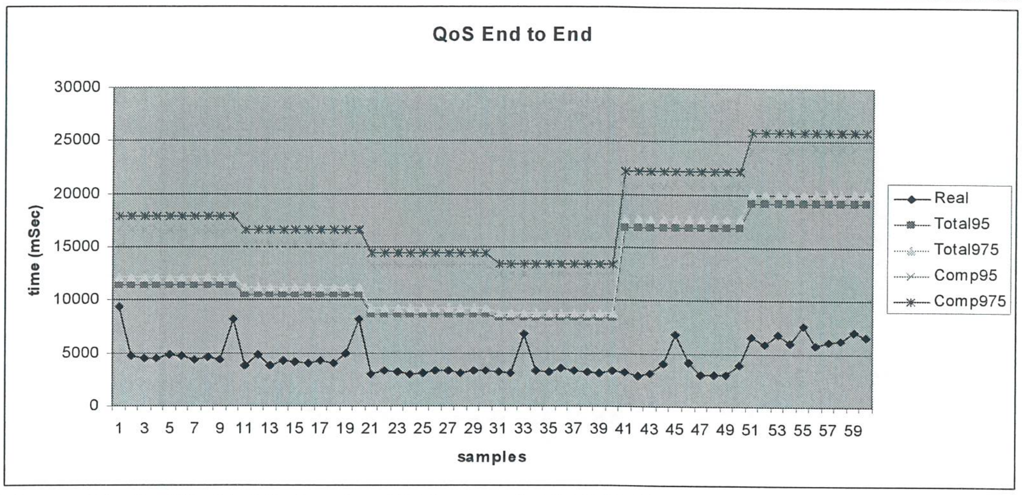


Figure 10b: End To End Estimation CPU server 80% CPU Client 80%

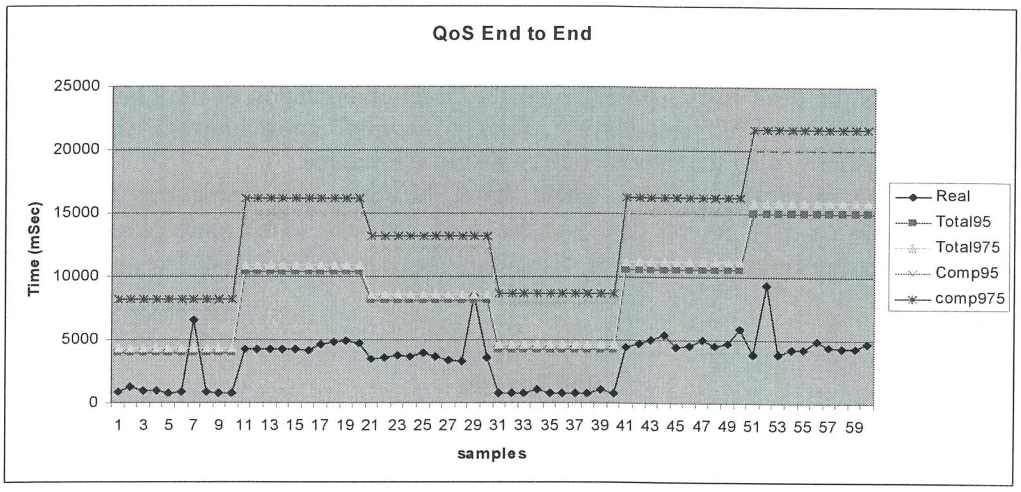


Figure 10c: End To End Estimation CPU server 80% CPU Client 80%

9.2 *Proof of concept in a real scenario*

As second tests set for the methodology validation we consider the concurrent scenario where more than one client uses the remote method. In this test the assumption of the server CPU at 80% (x_E=1) is mandatory considering that the number of service requestor is unknown by the clients.

The following plots are related to a scenario where 2 clients are concurrently request the service:

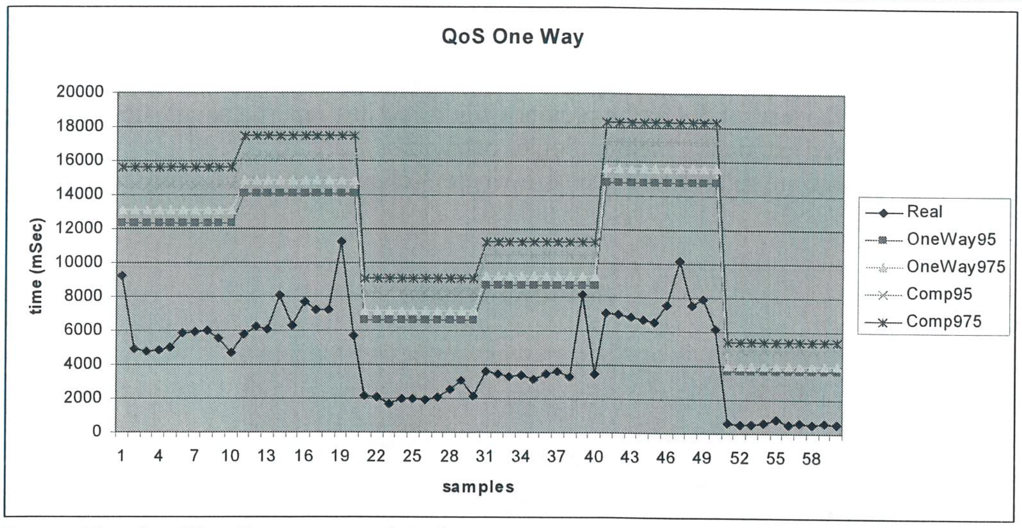


Figure 11a: One Way Estimation with 2 clients connected

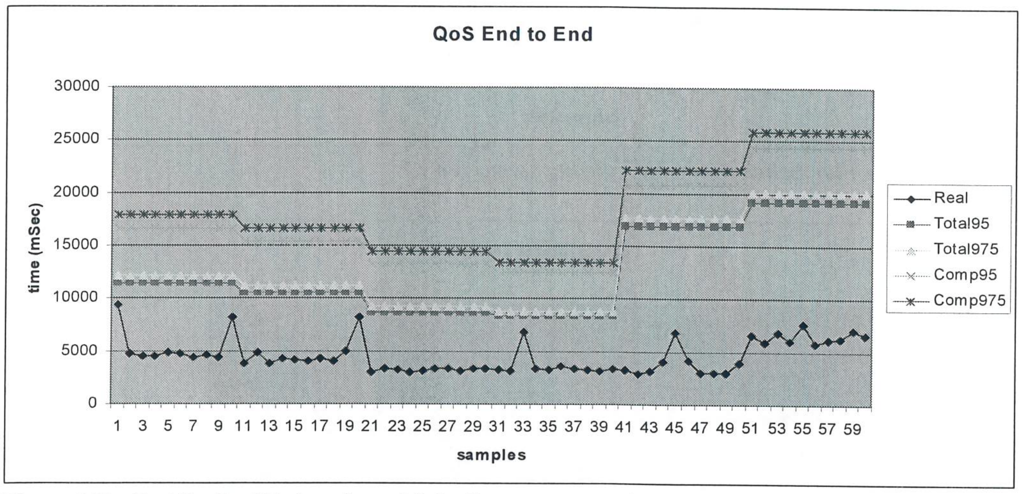


Figure 11b: End To End Estimation with 2 clients connected

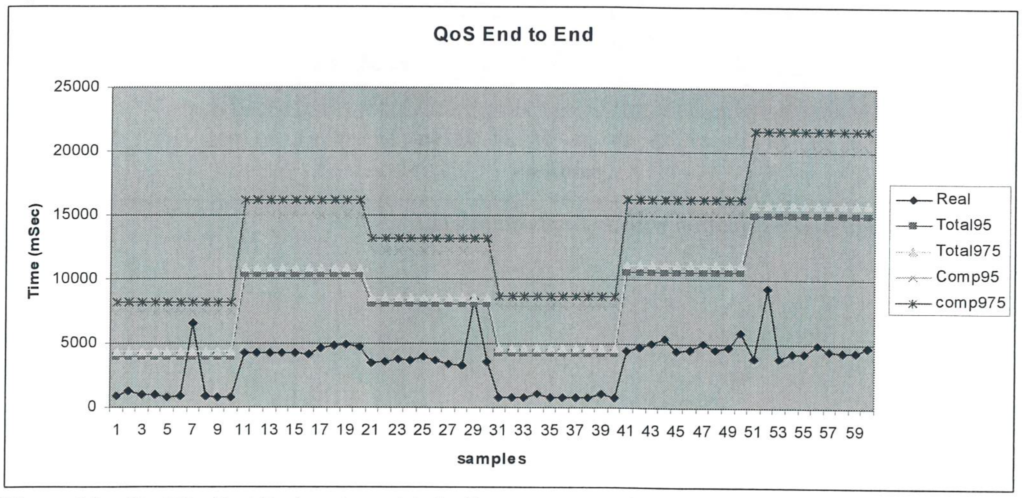


Figure 11c: End To End Estimation with 2 clients connected

Except for the number of clients connected to the system the plots line meaning is exactly the same explained in the previous paragraph. The methodology has been validated with all the 5000 samples of the test set but in the previous figure we just report a small subset to give an idea of the proposed estimation methodologies behaviors.
The following plots are related to a scenario where 3 clients are concurrently requesting the service:

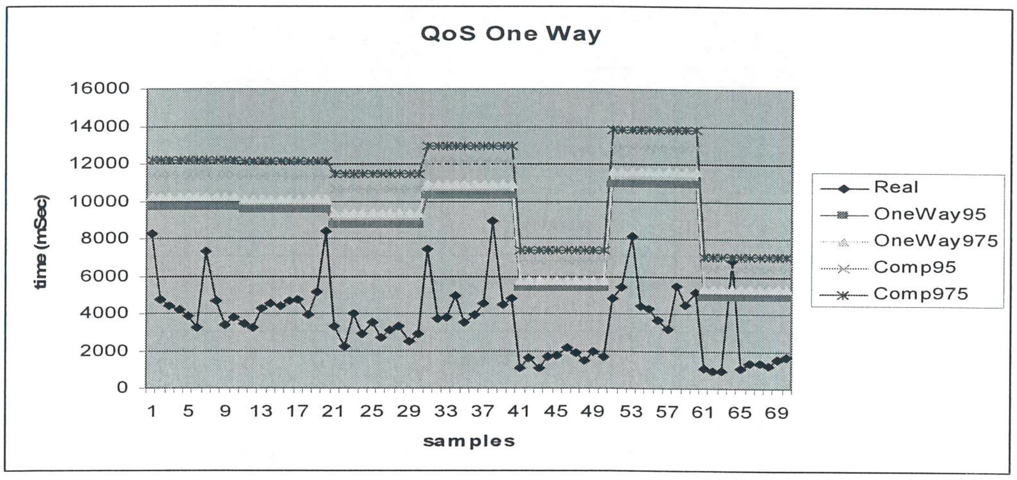


Figure 12a: One Way Estimation with 3 clients connected

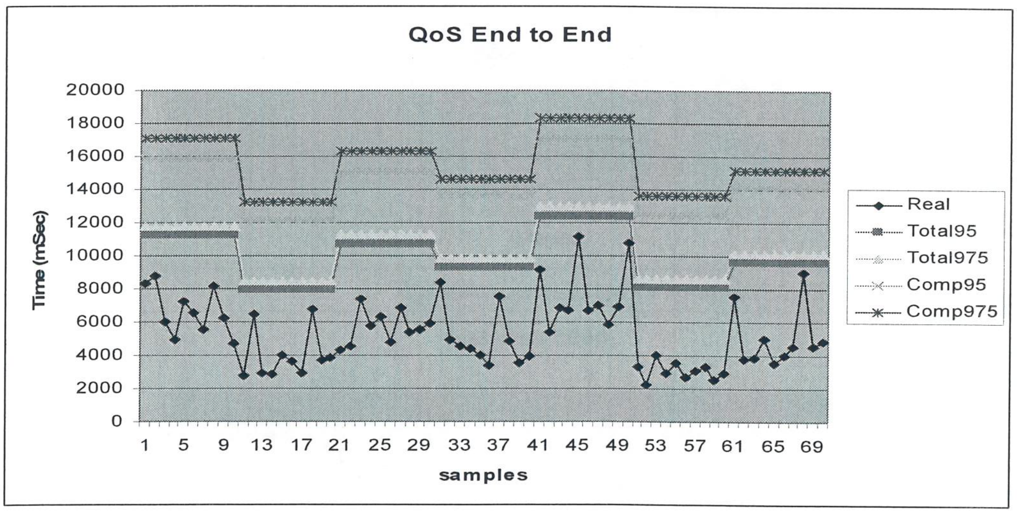


Figure 12b: End to End Estimation with 3 clients connected

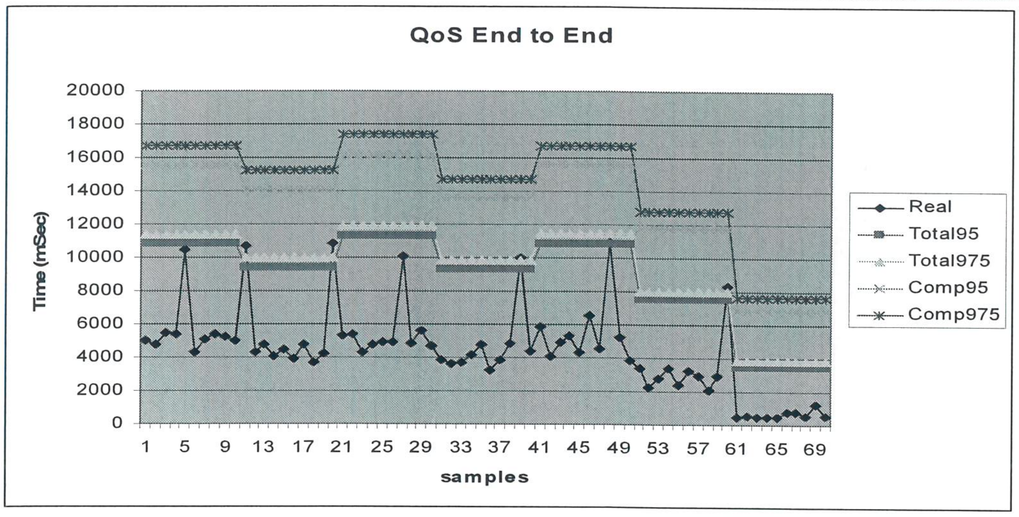


Figure 12c: End to End Estimation with 3 clients connected

Again, like all the previous tests the methodology was validated with all 5000 samples and in the plots we just report a small subset, as we can see the number of dead line miss is increasing due to the extra overhead of the server but it is still less than 5% and 2.5%.
According with [18], 3 concurrent clients that try to use the remote service as fast as they can bring the server to the maximum request throughput that it can handle. In other words if we try to increase the number of remote method invocation we expect a service time increase because the server is overloaded; consequently the number of dead line miss will increase in an unpredictable way.
The following figure are related to a test where 4 different clients are trying to access to the remote server.

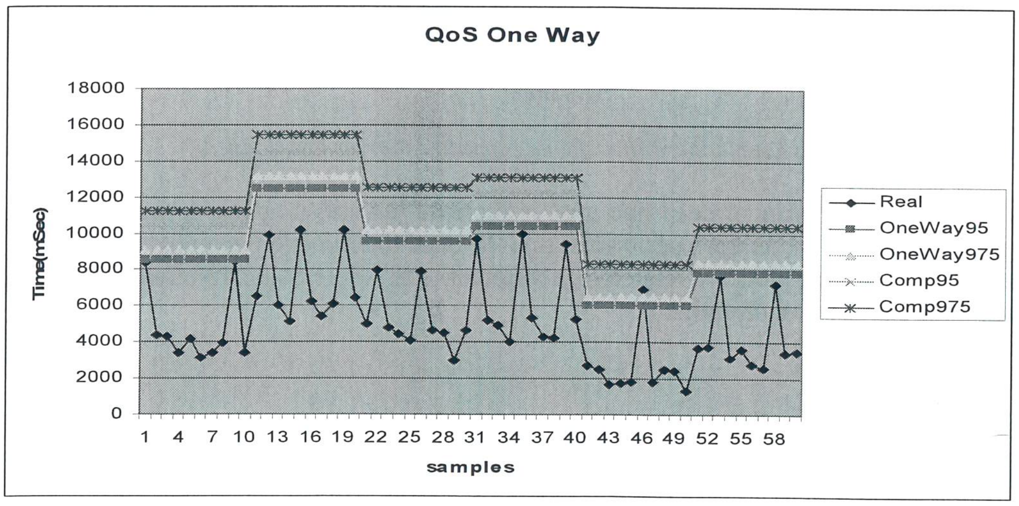


Figure 13a: One Way Estimation with 4 clients connected (Server Overloaded)

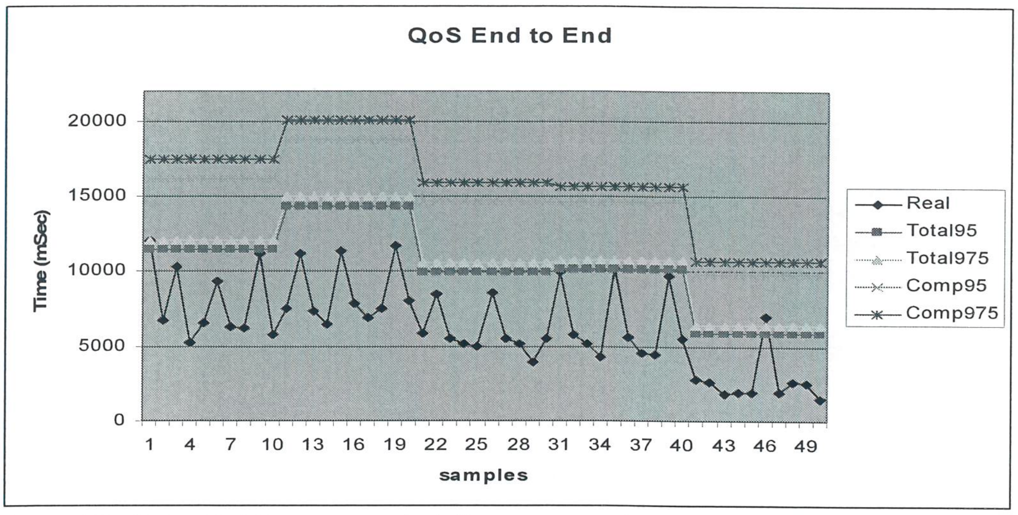


Figure 13b: End to End Estimation with 4 clients connected (Server Overloaded)

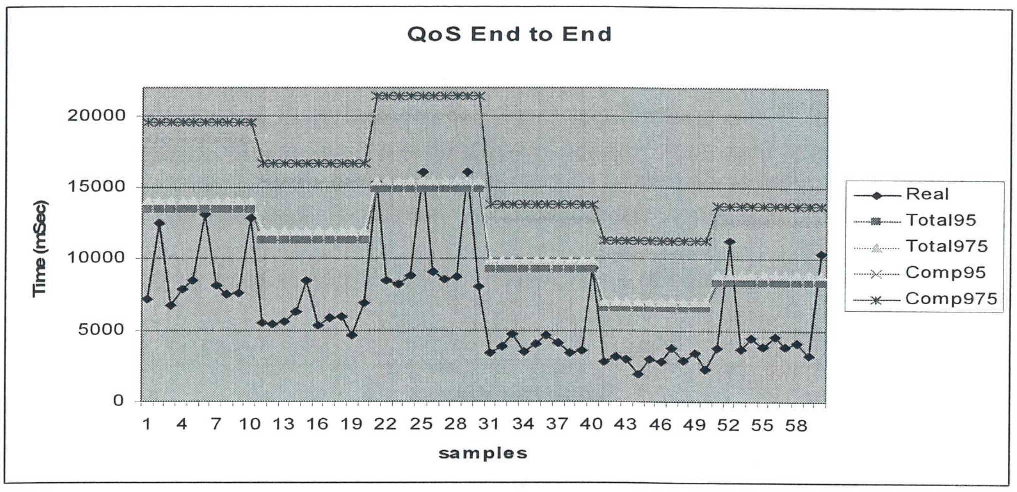


Figure 13c: End to End Estimation with 4 clients connected (Server Overloaded)

As expected the number of dead line miss is increase but still it appear to be under control due to the upper bound that we perform during the dead line function computation.
Finally the following plots are related to 2 different tests where 6 and 10 clients are trying to request the service concurrently:

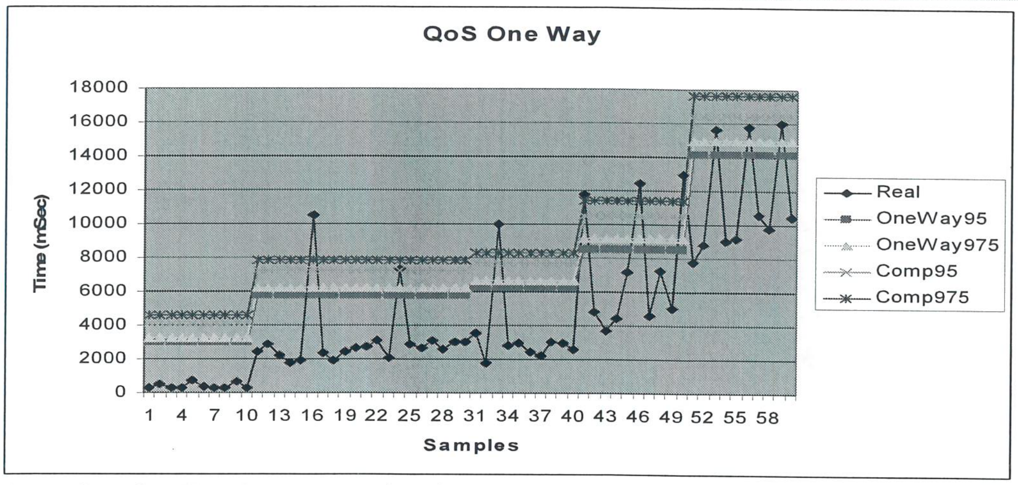


Figure 14a: One Way Estimation with 6 clients connected (Server Overloaded)

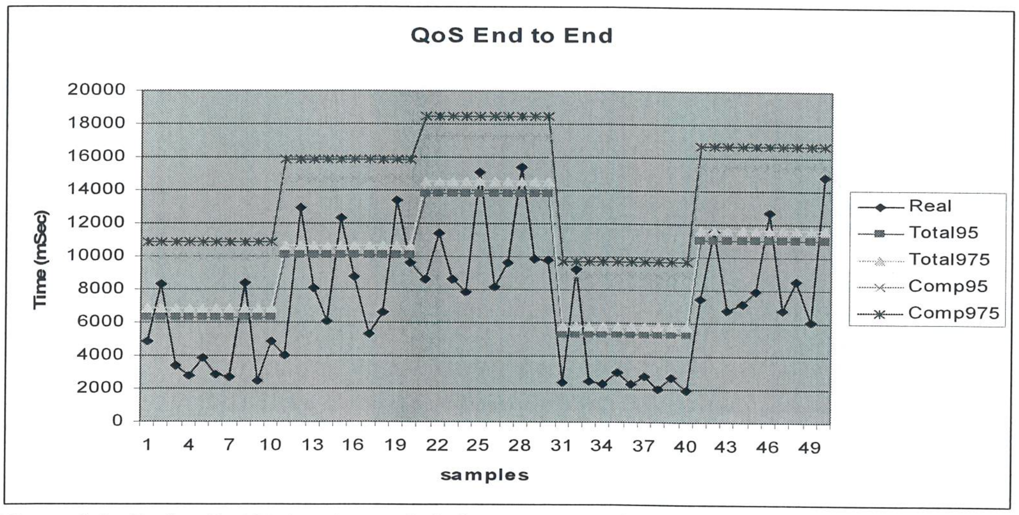


Figure 14b: End to End Estimation with 6 clients connected (Server Overloaded)

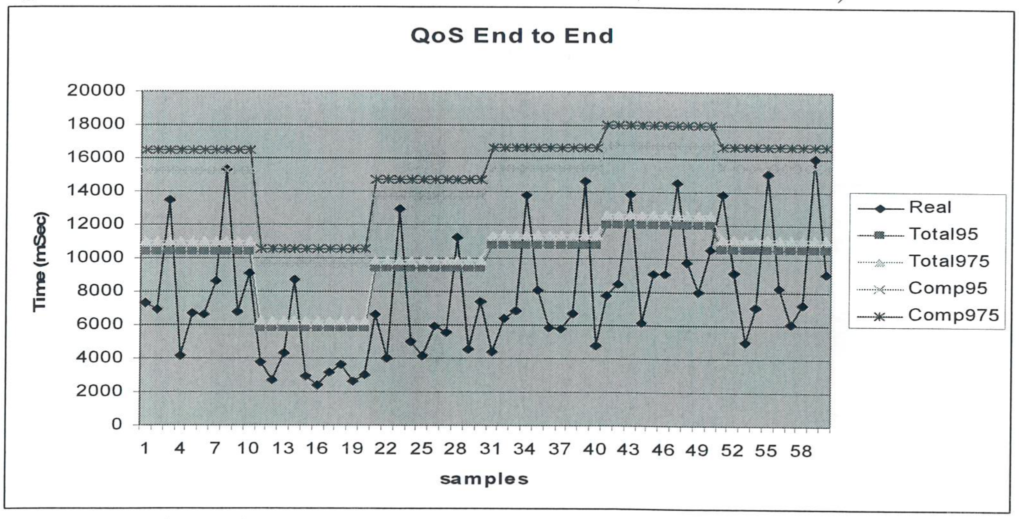


Figure 14c: End to End Estimation with 6 clients connected (Server Overloaded)

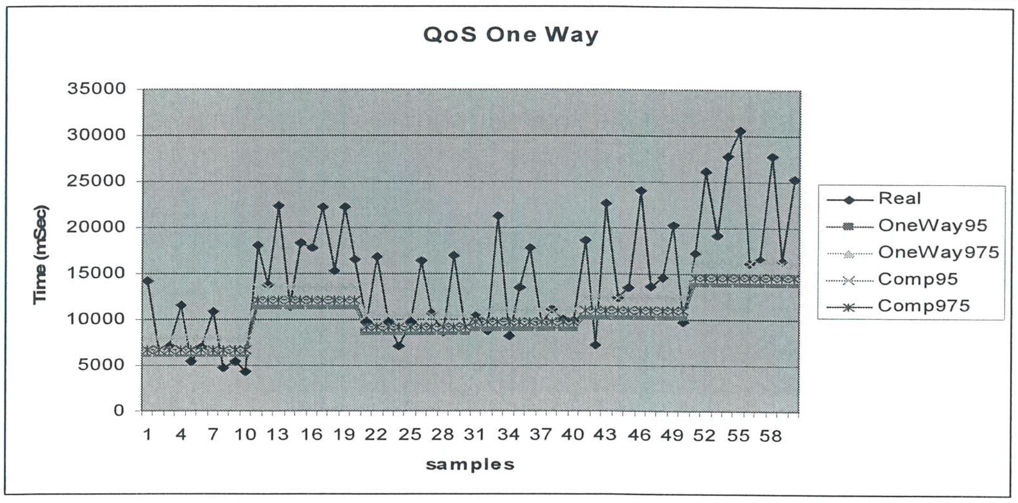


Figure 15a: One Way Estimation with 10 clients connected (Server Overloaded)

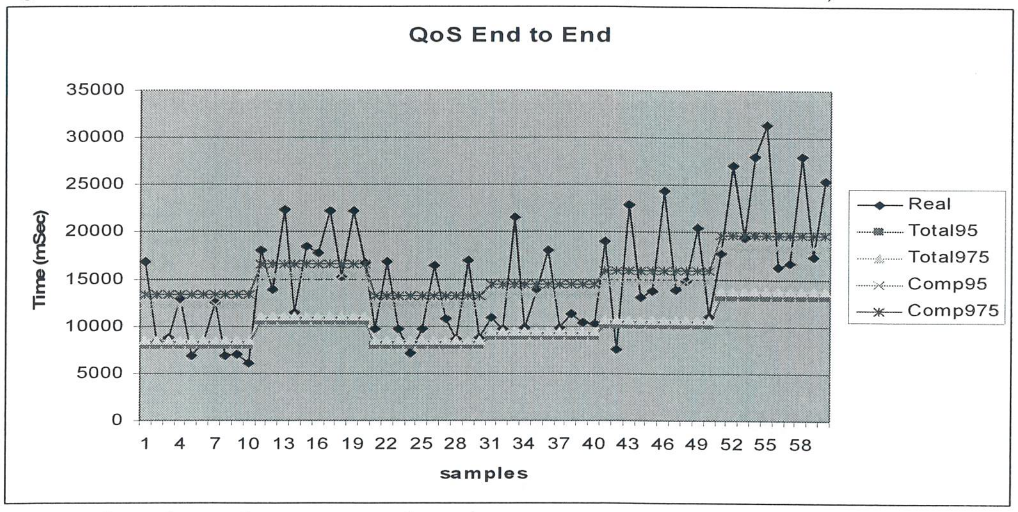


Figure 15b: End to End Estimation with 10 clients connected (Server Overloaded)

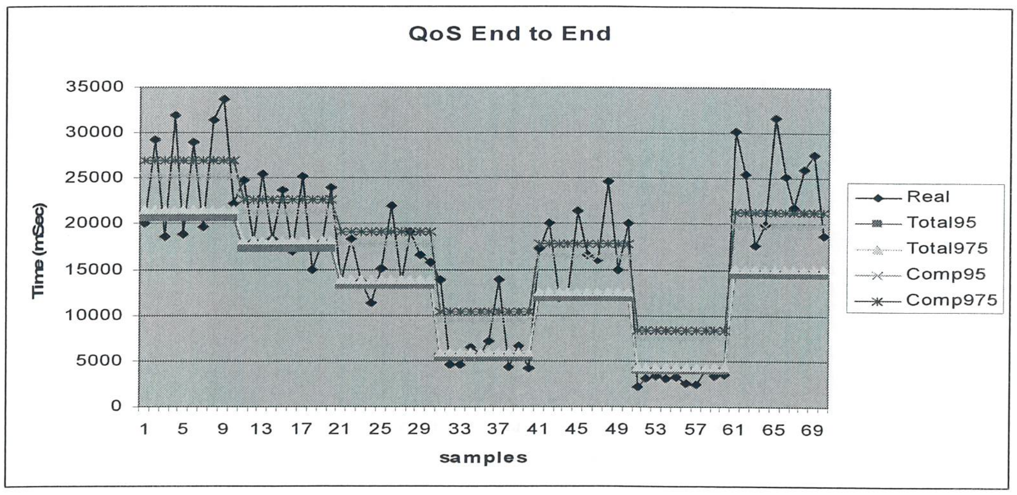


Figure 15c: End to End Estimation with 10 clients connected (Server Overloaded)

As expected the number of dead line miss is out of control due to the fact that the server is overloaded.

9.3 *Final remarks*

We believe that the tests showed in this chapter demonstrate the validity of the presented methodologies for estimation of e service execution time not only from a mathematical point of view but also in a concrete scenario. We also experience that in the case of server overloaded the number of dead line miss increase and this suggest possible approach for an automatic organization of the clients that balance the servers load: if the number of dead line miss is more than the expected probably the server is overload so one or more connected clients could decide to use a different machine where is deployed the same service.

10 Web Service Profile Algorithm

The methodology developed in chapter 8 and validate in chapter 9 can allow the development of an algorithm for estimation of the remote execution time:

1) Run the characterization tests according with the input and output region that you need to characterize
2) For each test find the minimum ε, ξ where
   $$N(\overline{\mu}+\varepsilon, \overline{\sigma}+\xi)$$
   Is a Gaussian distribution with a sample distribution larger than the one experienced in the tests.
3) Using the generated $\overline{\mu}+\varepsilon$ and $\overline{\sigma}+\xi$ values of each test find the best set of β_i for the function:
   $$y = \beta_0 + \beta_1 x_A + \beta_2 x_B + \beta_3 x_C + \beta_4 x_D + \beta_5 x_E + \beta_6 x_A x_B + \beta_7 x_B x_E + \beta_8 x_C x_E + \\ + \beta_9 x_D x_E + \beta_{10} x_F + \beta_{11} x_A x_F + \beta_{12} x_B x_F + \beta_{13} x_C x_F + \beta_{14} x_D x_F \pm \varepsilon$$
   That minimize the error ε by solving the equation:
   $$\hat{\beta} = (X'X)^{-1} y$$
4) Compute the error of the estimated functions looking the residual:
   $$\hat{y} = X\hat{\beta} \qquad e = y - \hat{y}$$

4b) (optional) sum the linear regression in the case of the interval decomposition technique

5) Compute the dead line function as the requested percentile of the set of Gaussians generated by
   $$N(\mu(\underline{x}), \sigma(\underline{x}))$$

11 Suggested Web Service QoS Enabled Architectures

In this chapter we will discuss 3 possible high level software architectures that will utilize the developed methodology in order to predict the remote method execution time. For each option we would like to provide an intuitive service orchestration and an advantage and disadvantage description of this solution.

11.1 Full client side logic

In this approach each client will utilize the algorithm present in section 10 in order to directly compute the dead line functions:

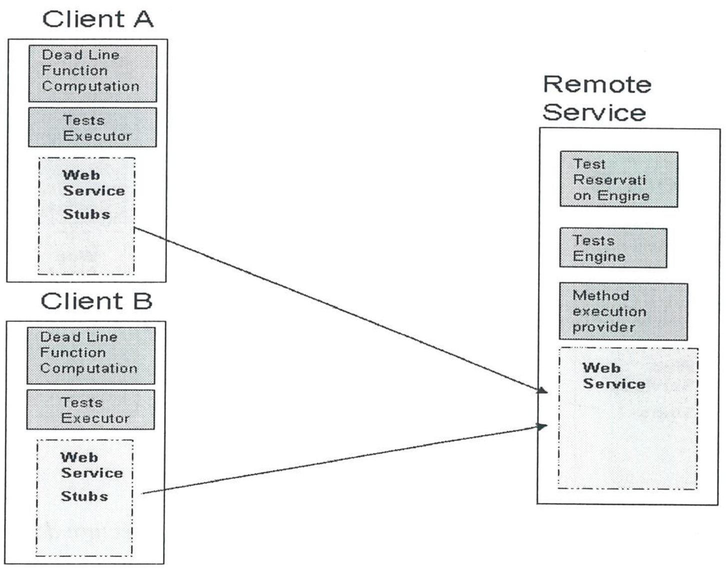


Figure 16: Full client Side logic Architecture description

For each client the Tests Executor component reserve the remote service using the Reservation Engine and perform the needed tests; than it compute the dead line functions via the Dead Line Function Computation component.
The remote service provide also a method execution characterization via the Method execution provider

Note on this solution:

- The server must provide a description of the remote methods execution time (t4-t3) for each method
- The clock synchronization is not needed during the test procedure
- During the client tests (step 1 of the algorithm) the server need to be totally dedicated to the client.
- Considering that all the tests will be computed on client side the one way computation could be not trivial and probably an upper bound estimation will be the only possibility
- During the tests procedure the network channel properties must be maintained constant

11.2 Server and clients factors break down

The key ideas of this approach is to break the dead line function in several piece and let the server and the client compute each part separately. Than the dead line function will be computed aggregating the functions of each factor.

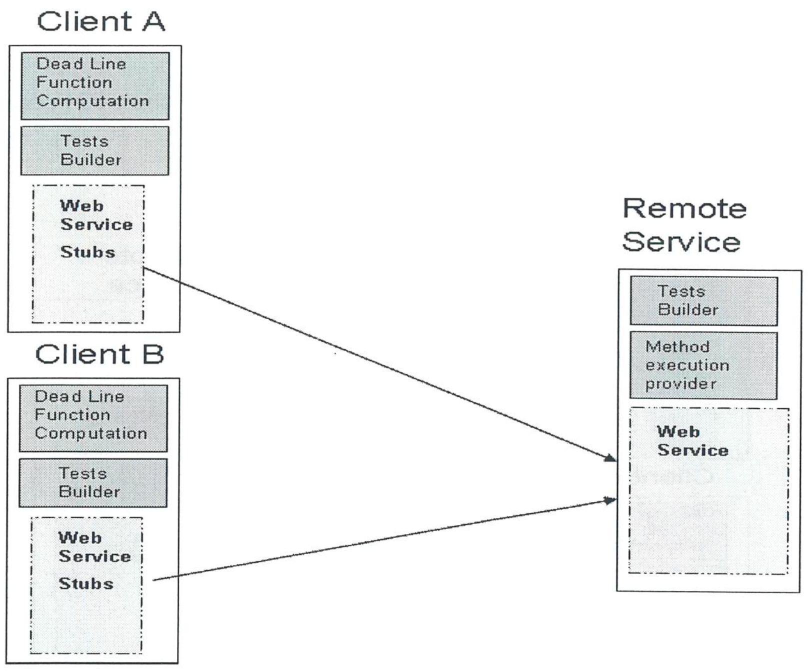


Figure 17: Server and clients factors break down Architecture description

In this case the servers build the characterization related to the intervals

- t3-t2 input deserialization
- t4-t3 remote method execution time
- t5-t4 output serialization

using the tests Builder component and make it public to the client using the Method execution provider component

The client profile itself providing a characterization of just following intervals

- t1-t0 input serialization
- t7-t6 output deserialization

using the test builder component, and, in addition, it compute the dead line functions retrieving the server characterization values via the Dead Line Function Computation component.
If a network characterization channel is required the client can retrieve it from a third party element.

Note on this solution:

- The server must provide a description of the remote methods execution time (t4-t3) for each method
- The server profile is build just once and does not change using different clients

- The client can build its test without asking the server cooperation
- The testing procedure do not care about the network channel
- The clock synchronization is not needed during the test procedure
- The estimate dead line will be bigger compared to a direct estimation due to the experimental errors introduced in each functions.
- A modification of the source code of the web service provider in both client and server side is mandatory since time like t1,t2,t5,t6 are internal part of the WS-Engine

11.3 Third party QoS Enabler

During the test procedure the server and the clients send the experienced times to a third party component that build the dead line functions and send it to the client.

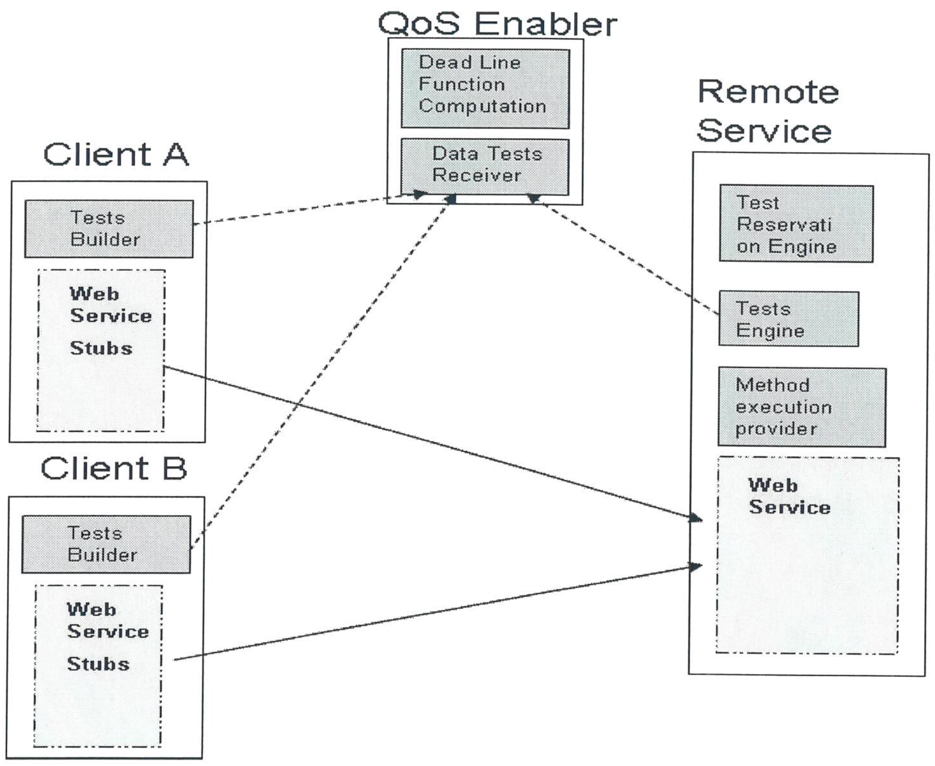


Figure 18: Third party QoS Enabler Architecture description

For each client the Tests Builder component reserve the remote service using the Reservation Engine and perform the needed tests. During the tests procedure both server and client send information related to t3-t2, t4-t3, t5-t4, t1-t0, t7-t6, intervals to the QoS Enabler component that builds the dead line functions.

Note on this solution:

- The server must provide a description of the remote methods execution time (t4-t3) for each method
- The external component can directly build the functions without the interval composition technique allowing a more accurate estimation

- The testing procedure does not care about the network channel since al the intervals are part of the server or of the client.
- The clock synchronization is not needed during the test procedure since al the intervals are part of the server or of the client.
- A centralized external component introduce a single point of failure and an additional software complexity
- A modification of the source code of the web service provider in both client and server side is mandatory since time like t1,t2,t5,t6 are internal part of the WS-Engine

12 Bibliography

[1] INTERMON project. http://www.intermon.org/.

[2] MONitoring Agents using a Large Integrated Services Architecture (MonALISA), California

[3] Institute of Technology. http://monalisa.caltech.edu/.

[4] PlanetLab project. http://www.planet-lab.org/.

[5] PerfSonar Project: http://www.perfsonar.net/

[6] Xia Gao, Ravi Jain, Zulfikar Ramzan, Ulas Kozat, "Resource Optimization for Web Service Composition ", SCC 2005, Jul. 2005

[7] Ran, Sh.: "A Model for Web Services Discovery With QoS", ACM SIGecom Exchanges, Vol. 4, No.1, 2003, pp. 1-10

[8] Maximilien, E.M., Singh, M.P.: "A Framework and Ontology for Dynamic Web Services Selection", IEEE Internet Computing, 5, 2004

[9] Jeckle, M., Zengler, B.: "Active UDDI - an Extension to UDDI for Dynamic and Fault-Tolerant Service Invocation", Web and Database-Related Workshops on Web, Web-Services and Database Systems, LNCS, 2003, pp. 91—99

[10] Gu, X., Chang, R.: "OoS-Assured Service Composition in managed Service Overlay Networks", In Proc. of The IEEE 23rd International Conference on Distributed Computing Systems (ICDCS 2003), 2003

[11] Gao, X., Jain, R., Ramzan, Z., Kozat, U.: "Resource optimization for Web Service Composition", in Proceedings of IEEE SCC2005, 2005

[12] Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: "Quality of service for workflows and web service processes", Journal of Web Semantics, Vol. 1, No. 3, 2004, pp. 281—308

[13] Yu, T., Lin, K.J.: "Service Selection Algorithms for Web Services with End-toend QoS Constraints", Journal of Information Systems and E-Business Management, Volumn 3, Number 2, July 2005

[14] Zhou, C., Chia, L.-T. & Lee, B.-S, QoS-Aware and Federated Enhancement for UDDI, International Journal of Web Services Research, Vol. 1, No. 2

[15] Java Clock precision: http://www.javaworld.com/javaworld/javaqa/2003-01/01-qa-0110-timing.html

[16] Network Clocks synchronization: http://zone.ni.com/devzone/conceptd.nsf/webmain/B40CBBA4A2B06A0B862570AF0055BDBA

[17] P. T. Eugster, P. A. Felber, R Guerraoui, A Kermarrec The many faces of publish/subscribe ACM Computing Surveys (CSUR) Volume 35, Issue 2 (June 2003) Pages: 114 - 131

[18] F. Lelli, G. Maron, S. Orlando, Improving the Performance of XML Based Technologies by Caching and Reusing Information, in proceeding of IEEE International Conference of Web Service (ICWS) September 2006

[19] A. Slominski, M. Govindaraju, M. R. Head, K. Chiu, M. J. Lewis, R. van Engelen, P. Liu, N Abu-Ghazaleh. A Benchmark Suite for SOAP-based Communication in Grid Web Services. In Proceedings of SC|05 (Supercomputing): International Conference for High Performance Computing, Networking, and Storage, Seattle WA, November, 2005.

[20] D.A. Menascé, "QoS Issues in Web Services," IEEE Internet Computing, vol. 6, no. 6, 2002, pp. 72–75.

[21] Mantaray Project: http://www.mantamq.org

[22] Vivek Chopra, Amit Bakore, Ben Galbraith, Sing Li, Chanoch Wiggers, Professional Apache Tomcat 5 Wrox May 2004

[23] D. C. Montgomery, Design and Analysis of Experiments 5th edition, John Wiley & Sons Inc, December 2004

[24] S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, R. N. Building, Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. Sams , December 2001

[25] G. Maron, A. Lenis, S. Moralis, M. Grammatikou, T. Karounos, S. Papavassiliou, V. Maglaris, P. Sphicas, S. Papavassiliou, T. Ferrari, C. A. Kotsokalis, A. S. McGough, T. Kalganova, P. Hobson, R. Pugliese, F. Lelli, D. Colling, The GridCC Architecture(GridCC Architecture design also available at www.gridcc.org) May 2005. (Authors are in no particular order).

13 Appendix I: a taste on the 2^K factorial analysis

Certain special types of factorial design are very useful in process development. One of these is a factorial design with k factors, each at two levels. Because each complete replicate of the design has 2^k runs, the arrangement is called a 2^k factorial design. These designs have a greatly simplified analysis, and they also form the basis of many other useful designs.

13.1 The 2^2 Design

The simplest type of 2^k design is the 2^2 - that is two factors A and B, each at two levels. We usually think of these levels as the "low" and "high" levels of the factor. The 2^2 design is shown in the graph below. Note that the design can be represented geometrically as a square with the $2^2 = 4$ runs forming the corners of the square.

```
 High(+) b= •                              • ab=
B
Low(-) (1) = •                              • a =
             Low(-)                        High(+)

                         A →
```

A special notation is used to represent the runs. In general, a run is represented by a series of lower case letters. If a letter is present, then the corresponding factor is set at the high level in that run; if it is absent, the factor is run at its low level. For example, run *a* indicates that factor *A* is at the high level and factor *B* is at the low level. The run with both factors at the low level is represented by (1). This notation is used throughout the 2^k design series. For example, the run in a 2^4 with *A* and *C* at the high level and *B* and *D* at the low level is demoted by *ac*.

The effects of interest in the 2^2 design are the main effects *A* and *B* and the two-factor interaction *AB*. Let the letters (1), *a*, *b*, and *ab* also represent the totals of all *n* observations taken at these designs points. It is easy to estimate the effects of these factors. To estimate the main effect of *A*, we would average the observations on the right side of the square when *A* is at the high level and subtract from this the average of the observations on the left side of the square where *A* is at the low level, or

$$A = \frac{a + ab}{2n} - \frac{b + (1)}{2n}$$

$$A = \frac{1}{2n}\left[a + ab - b - (1)\right]$$

Similarly, the main effect of *B* is found by averaging the observations on the top of the square where *B* is at the high level and subtracting the average of the observations on the bottom of the square where *B* is at the low level,

Finally, the AB interaction is estimated by taking the difference in the diagonal averages.

$$AB = \frac{ab + (1)}{2n} - \frac{a + b}{2n} = \frac{1}{2n}\left[ab + (1) - a - b\right]$$

The quantities in brackets in the above equations are called contrasts. For example, the A contrasts is:

$\text{Contrast}_A = a + ab - b - (1)$

In these equations, the contrast coefficients are always either +1 or -1. A table of plus and minus signs can be used to determine the sign on each run for a particular contrast.The column headings for the table are the main effects A and B, in the AB interaction, and I, which represents the total. The row headings are the runs. Note that the signs in the AB column are the products of signs from columns A and B. To generate a contrast from this table, multiply the signs in the appropriate column of the table by the runs listed in the rows and add.
To obtain the sums of squares for A, B, and AB, we use:

$$SS = \frac{(contrasted)^2}{n\sum(ContrastCoefficients)^2}$$

Therefore, the sums of squares for A, B, and AB are

$$SS_A = \frac{[a + ab - b - (1)]^2}{4n}$$

$$SS_B = \frac{[b + ab - a - (1)^2]}{4n}$$

$$SS_{AB} = \frac{[ab + (1) - a - b]^2}{4n}$$

Signs for effects in the 2^2 design

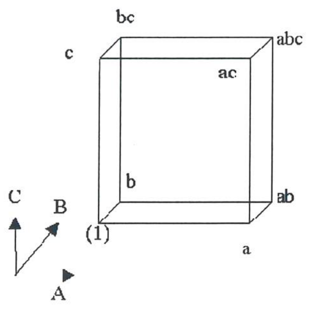
Factor Effects

Run	I	A	B	AB
1 (1)	+	-	-	+
2 a	+	+	-	-
3 b	+	-	+	-
4 ab	+	+	+	+

The analysis of variance is completed by computing the total sum of squares SS_T (with 4n - 1 degrees of freedom) as usual, and obtaining the error sum of squares SS_E [with 4(n-1) degrees of freedom] by subtraction.

13.2 The 2^2 Design for k≥3 Factors

The method presented in the above section for factorial designs with k = 2 factors each at two levels can be easily extended to more than two factors. For example, consider k = 3 factors, each at two levels. This design is a 2^3 factorial design, and it has eight factor-level combinations. Geometrically, the design is a cube as can be seen below, with eight runs forming the corners of the cube. This design allows three main effects to be estimated (A,B, and C) along with three tow-factor interactions (AB, AC, and BC) and the three-factor interactions (ABC).
The main effects can be estimated pretty easily. Remember that the lower case letters (1), a, b, c, ac, bc, and abc represent the total of all n replicates at each of the eight runs in the design. Referring to the cube below, we would estimate the main effect of a A by averaging the four runs on the right side of the cube where



A is at the high level and subtracting form that quantity the average of the four runs on the left side of the cube were A is at the low level. This gives:

$$A = \frac{1}{4n}[a + ab + ac + abc - b - c - bc - (1)]$$

In a similar manner, the effect of B is the average difference of the four runs in the back face of the cube and the four in the front, or:

$$B = \frac{1}{4n}[b = ab = cb = abc - a - c - ac - (1)]$$

and the effect of C is the average difference between the four runs in the top face of the cube and the four in the bottom, or:

$$C = \frac{1}{4n}[c + ac + bc + abc - a - b - ab - (1)]$$

Now consider the two-factor interaction AB. When C is at the low level, AB is just the average difference in the A effect at the two levels of B, or

$$AB(Clow) = \frac{1}{2n}[ab - b] - \frac{1}{2n}[a - (1)]$$

Similarly, when C is at the high level, the AB interaction is:

$$AB(Chigh) = \frac{1}{2n}[abc - bc] - \frac{1}{2n}[ac - c]$$

The AB interaction is just the average of these tow components, or:

$$AB = \frac{1}{4n}[ab + (1) + abc + c - b - a - bc - ac]$$

Notice that the AB interaction is just the difference in averages on two diagonal planes in the cube. Using similar approach, we can show that the AC and BC interaction effects estimates are as follows:

$$AC = \frac{1}{4n}[ac + (1) + abc + b - a - c - ab - bc]$$

$$BC = \frac{1}{4n}[bc + (1) + abc + a - b - c - ab - ac]$$

The ABC interaction effect is the average difference between the AB interaction and the two levels of C, thus:

$$ABC = \frac{1}{4n}\{[abc - bc] - [ac - c] - [ab - b] + [a - (1)]\}$$

$$= \frac{1}{4n}[abc - bc - ac + c - ab + b + a - (1)]$$

The quantities in brackets in the above equations are contrasts in the eight factor-level combinations. These contrasts can be obtained from a table of plus and minus signs for the 2^3 design, shown in the following table. Signs for the main effects (columns A,B, and C) are obtained by associating a plus with the high level. Once the signs for the main effects have been established, the sighs for the remaining columns are found by multiplying the appropriate preceding columns, row by row. For example, the signs in column AB are the products of the signs in column A and B.

Signs for effects in the 2^3 design

Treat. Combo	I	A	B	AB	C	AC	BC	ABC
(1)	+	-	-	+	-	+	+	-
a	+	+	-	-	-	-	+	+
b	+	-	+	-	-	+	-	+
ab	+	+	+	+	-	-	-	-
c	+	-	-	+	+	-	-	+
ac	+	+	-	-	+	+	-	-
bc	+	-	+	-	+	-	+	-
abc	+	+	+	+	+	+	+	+

What about this table?

1. Except for the identity column I, each column has an equal number of plus and minus signs.
2. The sum of products of signs in any two columns is zero; that is, the columns in the table are orthogonal.
3. Multiplying any column by column I leaves the column unchanged; that is, I is an identity element.
4. The product of any two columns yields a column in the table, for example, $AxB = AB$, and $ABxABC = A^2B^2C = C$, since any column multiplied by itself is the identity column.

The estimate of any main effect or interactions is determined by multiplying the factor-level combinations in the first column of the table by the signs in the corresponding main effect or interaction column, adding the result to produce a contrast, and then dividing the contrast by one-half the total number of runs in the experiment. Expressed in good old math terms looks like:

$$Effect = \frac{Contrast}{n2^{k-1}}$$

The sum of squares for any effect is:

$$SS = \frac{(Contrast)^2}{n2^k}$$

13.3 Other Methods for Judging the Significance of Effects

The analysis of variance that we have already covered is a formal way to determine which effects are nonzero. Two other methods are useful. In the first method, we can calculate the standard errors (which we have also seen in class/homework) of the effects and compare the magnitude of the effects to their standard errors. The second method uses the normal probability plots (quantile plots) to assess the importance of the effects.

The standard error of an effect is easy to find. If we assume that there are n replicates at each of the 2^k runs in the design, and if y_{i1}, y_{i2},…, y_{in} are the observations at the ith run then :

$$S_i^2 = \frac{1}{n-1}\sum_{j=1}^{n}(y_{ij} - \overline{y}_i)^2$$

is an estimate of the variance at the ith run. The 2^k variance estimates can be pooled to give an overall variance estimate:

$$S^2 = \frac{1}{2^k(n-1)}\sum_{i=1}^{2^k}\sum_{j=1}^{n}(y_{ij} - \overline{y}_i)^2$$

This is also the variance estimate given by the error mean square from the analysis of variance procedure. Each effect estimate has variance given by:

$$V(effect) = \frac{1}{n2^{k-2}}\sigma^2$$

The estimated standard error of an effect would be found by replacing σ^2 by its estimate S^2 and taking the square root of:

$$V(effect) = \frac{1}{n2^{k-2}}\sigma^2$$

14 Appendix II details on the performed 2^k factorial analysis

14.1 Total Execution Time

Factor	AVG Cont. x10 5	SS x108	Red. SS	SDev Contrast x10 4	SS x10 6	Red. SS
a	0.7751	0.9387	0.9387	1.8293	5.2288	5.2288
b	0.8445	1.1144	1.1144	0.5331	0.4440	0.4440
ab	-0.0677	0.0072	0	-1.0265	1.6463	1.6463
c	1.8746	5.4908	5.4908	1.6064	4.0322	4.0322
ac	0.0724	0.0082	0	1.2243	2.3420	2.3420
bc	-0.0434	0.0029	0	-1.0314	1.6621	1.6621
abc	-0.0781	0.0095	0	-0.8466	1.1200	1.1200
d	0.9356	1.3678	1.3678	1.6363	4.1836	4.1836
ad	0.0565	0.0050	0	0.1765	0.0487	0
bd	-0.0028	0.0000	0	-0.1600	0.0400	0
abd	-0.0007	0.0000	0	0.0873	0.0119	0
cd	0.0618	0.0060	0	0.0887	0.0123	0
acd	0.0361	0.0020	0	0.3341	0.1745	0
bcd	0.0173	0.0005	0	0.1173	0.0215	0
abcd	-0.0069	0.0001	0	-0.0313	0.0015	0
e	0.3057	0.1460	0	0.5529	0.4776	0.4776
ae	0.0983	0.0151	0	0.6345	0.6291	0.6291
be	0.0463	0.0033	0	-0.2737	0.1170	0
abe	0.0182	0.0005	0	-0.2682	0.1124	0
ce	0.1651	0.0426	0	0.5397	0.4551	0.4551
ace	0.0215	0.0007	0	0.5662	0.5010	0.5010
bce	0.0286	0.0013	0	-0.1623	0.0412	0
abce	0.0285	0.0013	0	-0.1531	0.0366	0
de	0.0734	0.0084	0	0.2396	0.0897	0
ade	0.0252	0.0010	0	0.2506	0.0981	0
bde	-0.0085	0.0001	0	-0.0197	0.0006	0
abde	-0.0035	0.0000	0	-0.0116	0.0002	0
cde	-0.0113	0.0002	0	0.1517	0.0360	0
acde	0.0141	0.0003	0	0.2584	0.1043	0
bcde	-0.0495	0.0038	0	-0.2318	0.0839	0
abcde	-0.0068	0.0001	0	-0.0718	0.0081	0
f	0.8019	1.0048	1.0048	1.6066	4.0333	4.0333
af	0.0497	0.0039	0	-0.4161	0.2705	0.2705
bf	0.2982	0.1389	0	1.0788	1.8184	1.8184
abf	0.0126	0.0002	0	0.3477	0.1889	0
cf	0.2399	0.0900	0	-0.1031	0.0166	0
acf	0.0006	0.0000	0	-0.3850	0.2316	0
bcf	0.0355	0.0020	0	0.2604	0.1060	0
abcf	0.0061	0.0001	0	0.3381	0.1786	0
df	0.2994	0.1401	0	0.9519	1.4157	1.4157
adf	0.0488	0.0037	0	0.3412	0.1819	0
bdf	-0.0308	0.0015	0	-0.4191	0.2745	0.2745
abdf	-0.0251	0.0010	0	-0.3624	0.2053	0
cdf	0.0624	0.0061	0	0.2386	0.0889	0
acdf	0.0415	0.0027	0	0.3401	0.1808	0
bcdf	-0.0053	0.0000	0	-0.1735	0.0470	0

abcdf	-0.0253	0.0010	0	-0.2943	0.1353	0
ef	-0.0599	0.0056	0	-0.2455	0.0942	0
aef	0.0063	0.0001	0	-0.0952	0.0142	0
bef	-0.0308	0.0015	0	-0.0874	0.0119	0
abef	-0.0118	0.0002	0	-0.0522	0.0043	0
cef	-0.0345	0.0019	0	-0.0652	0.0066	0
acef	0.0218	0.0007	0	0.0115	0.0002	0
bcef	-0.0045	0.0000	0	-0.0133	0.0003	0
abcef	0.0055	0.0000	0	0.0112	0.0002	0
def	-0.0044	0.0000	0	0.0988	0.0152	0
adef	-0.0074	0.0001	0	0.1085	0.0184	0
bdef	0.0053	0.0000	0	-0.1273	0.0253	0
abdef	0.0167	0.0004	0	-0.0725	0.0082	0

14.2 One way Time

Factor	AVG Cont. x10 5	SS x108	Red. SS	SDev Contrast x10 4	SS x10 7	Red. SS
a	0.7818	0.9549	0.9549	2.3453	0.8594	0.8594
b	-0.0292	0.0013	0	-0.3956	0.0245	0
ab	-0.0631	0.0062	0	-0.7751	0.0939	0.0939
c	1.8782	5.5119	5.5119	2.5463	1.0131	1.0131
ac	0.0484	0.0037	0	0.8959	0.1254	0.1254
bc	-0.0564	0.0050	0	-0.7303	0.0833	0.0833
abc	-0.0877	0.0120	0	-1.0694	0.1787	0.1787
d	0.0705	0.0078	0	0.7706	0.0928	0.0928
ad	0.0496	0.0039	0	0.4239	0.0281	0
bd	-0.0260	0.0011	0	0.0645	0.0007	0
abd	-0.0093	0.0001	0	0.0849	0.0011	0
cd	0.0445	0.0031	0	0.4701	0.0345	0
acd	0.0272	0.0012	0	0.1570	0.0039	0
bcd	-0.0167	0.0004	0	-0.1255	0.0025	0
abcd	-0.0017	0.0000	0	-0.1185	0.0022	0
e	0.2380	0.0885	0	0.6702	0.0702	0.0702
ae	0.0874	0.0119	0	0.7246	0.0820	0.0820
be	0.0212	0.0007	0	-0.1714	0.0046	0
abe	0.0148	0.0003	0	-0.1811	0.0051	0
ce	0.1629	0.0415	0	0.5361	0.0449	0
ace	0.0126	0.0002	0	0.6063	0.0574	0.0574
bce	0.0259	0.0011	0	-0.1716	0.0046	0
abce	0.0198	0.0006	0	-0.1668	0.0043	0
de	0.0197	0.0006	0	0.2254	0.0079	0
ade	0.0197	0.0006	0	0.2762	0.0119	0
bde	-0.0201	0.0006	0	-0.0465	0.0003	0
abde	-0.0016	0.0000	0	0.0105	0.0000	0
cde	0.0136	0.0003	0	0.2572	0.0103	0
acde	0.0140	0.0003	0	0.3122	0.0152	0
bcde	-0.0251	0.0010	0	-0.1336	0.0028	0
abcde	-0.0066	0.0001	0	-0.0703	0.0008	0
f	0.3404	0.1811	0	0.5866	0.0538	0.0538
af	0.0685	0.0073	0	-0.2881	0.0130	0
bf	0.0487	0.0037	0	0.5781	0.0522	0.0522
abf	0.0205	0.0007	0	0.4485	0.0314	0
cf	0.2498	0.0975	0	0.2804	0.0123	0
acf	-0.0129	0.0003	0	-0.4535	0.0321	0

bcf	0.0264	0.0011	0	0.3585	0.0201	0
abcf	0.0011	0.0000	0	0.2676	0.0112	0
df	0.0707	0.0078	0	0.4879	0.0372	0
adf	0.0486	0.0037	0	0.3778	0.0223	0
bdf	-0.0479	0.0036	0	-0.3697	0.0214	0
abdf	-0.0360	0.0020	0	-0.3473	0.0188	0
cdf	0.0530	0.0044	0	0.3784	0.0224	0
acdf	0.0346	0.0019	0	0.3019	0.0142	0
bcdf	-0.0335	0.0018	0	-0.3138	0.0154	0
abcdf	-0.0240	0.0009	0	-0.3390	0.0180	0
ef	-0.0272	0.0012	0	-0.1795	0.0050	0
aef	0.0093	0.0001	0	-0.0558	0.0005	0
bef	-0.0073	0.0001	0	-0.0197	0.0001	0
abef	-0.0122	0.0002	0	-0.0195	0.0001	0
cef	-0.0216	0.0007	0	-0.0473	0.0003	0
acef	0.0151	0.0004	0	0.0710	0.0008	0
bcef	-0.0005	0.0000	0	-0.0057	0.0000	0
abcef	-0.0054	0.0000	0	-0.0017	0.0000	0
def	-0.0040	0.0000	0	0.0562	0.0005	0
adef	-0.0080	0.0001	0	0.0956	0.0014	0
bdef	-0.0034	0.0000	0	-0.1741	0.0047	0
abdef	0.0127	0.0003	0	-0.0876	0.0012	0

14.3 Client Serialization

Factor	AVG Cont. x10 4	SS x108	Red. SS	SDev Contrast x10 4	SS x10 7	Red. SS
a	2.2312	0.7779	0.7779	0.4732	0.3498	0.3498
b	0.4443	0.0308	0	0.6124	0.5860	0.5860
ab	0.0680	0.0007	0	0.2032	0.0645	0
c	6.5707	6.7459	6.7459	1.2357	2.3857	2.3857
ac	-0.0270	0.0001	0	-0.3843	0.2307	0.2307
bc	0.2139	0.0071	0	0.1654	0.0427	0
abc	-0.1380	0.0030	0	-0.2094	0.0685	0
d	0.3885	0.0236	0	0.4853	0.3680	0.3680
ad	0.0845	0.0011	0	0.1093	0.0187	0
bd	-0.1381	0.0030	0	0.0002	0.0000	0
abd	-0.0014	0.0000	0	0.0476	0.0035	0
cd	0.1921	0.0058	0	0.1465	0.0335	0
acd	-0.0777	0.0009	0	-0.1981	0.0613	0
bcd	-0.0052	0.0000	0	-0.0667	0.0069	0
abcd	0.1113	0.0019	0	-0.0499	0.0039	0
e	-0.3366	0.0177	0	-0.0974	0.0148	0
ae	-0.0256	0.0001	0	0.0060	0.0001	0
be	0.0737	0.0008	0	0.0294	0.0014	0
abe	-0.0059	0.0000	0	0.0146	0.0003	0
ce	-0.3026	0.0143	0	-0.1100	0.0189	0
ace	0.0099	0.0000	0	-0.0012	0.0000	0
bce	0.1202	0.0023	0	0.0168	0.0004	0
abce	0.0424	0.0003	0	0.0093	0.0001	0
de	0.0335	0.0002	0	-0.0328	0.0017	0
ade	-0.0023	0.0000	0	0.0097	0.0001	0
bde	-0.1080	0.0018	0	-0.0483	0.0037	0
abde	0.0602	0.0006	0	0.0192	0.0006	0
cde	-0.0167	0.0000	0	-0.0369	0.0021	0

acde	-0.0508	0.0004	0	0.0034	0.0000	0
bcde	-0.1594	0.0040	0	-0.0433	0.0029	0
abcde	0.0092	0.0000	0	0.0209	0.0007	0
f	3.3623	1.7664	1.7664	1.0986	1.8858	1.8858
af	0.7956	0.0989	0	0.2016	0.0635	0
bf	0.3363	0.0177	0	0.2286	0.0816	0
abf	0.0515	0.0004	0	0.0904	0.0128	0
cf	2.5291	0.9994	0.9994	0.6779	0.7181	0.7181
acf	0.0541	0.0005	0	-0.0687	0.0074	0
bcf	0.1468	0.0034	0	-0.0094	0.0001	0
abcf	-0.1124	0.0020	0	-0.1104	0.0190	0
df	0.2968	0.0138	0	0.1120	0.0196	0
adf	0.0830	0.0011	0	-0.0037	0.0000	0
bdf	-0.1049	0.0017	0	0.0262	0.0011	0
abdf	0.0192	0.0001	0	0.0559	0.0049	0
cdf	0.1469	0.0034	0	0.0906	0.0128	0
acdf	-0.0311	0.0002	0	0.0104	0.0002	0
bcdf	0.0242	0.0001	0	0.0036	0.0000	0
abcdf	0.1266	0.0025	0	-0.0012	0.0000	0
ef	-0.3623	0.0205	0	-0.1349	0.0284	0
aef	-0.0368	0.0002	0	-0.0280	0.0012	0
bef	0.0596	0.0006	0	-0.0341	0.0018	0
abef	-0.0113	0.0000	0	-0.0258	0.0010	0
cef	-0.3087	0.0149	0	-0.0682	0.0073	0
acef	0.0184	0.0001	0	0.0454	0.0032	0
bcef	0.1201	0.0023	0	0.0442	0.0031	0
abcef	0.0513	0.0004	0	0.0615	0.0059	0
def	0.0336	0.0002	0	-0.0304	0.0014	0
adef	-0.0028	0.0000	0	0.0165	0.0004	0
bdef	-0.1115	0.0019	0	-0.0621	0.0060	0
abdef	0.0619	0.0006	0	0.0167	0.0004	0

14.4 Client Deserialization

Factor	**AVG Cont. x10 4**	**SS x107**	**Red. SS**	**SDev Contrast x10 4**	**SS x10 6**	**Red. SS**
a	-0.1698	0.0045	0	-0.0352	0.0019	0
b	6.8217	7.2711	7.2711	1.4468	3.2707	3.2707
ab	-0.1010	0.0016	0	-0.0887	0.0123	0
c	-0.1359	0.0029	0	-0.4759	0.3539	0.3539
ac	0.1946	0.0059	0	0.0540	0.0046	0
bc	0.0803	0.0010	0	-0.1502	0.0353	0
abc	0.0561	0.0005	0	0.0576	0.0052	0
d	6.3211	6.2432	6.2432	1.4763	3.4054	3.4054
ad	0.0186	0.0001	0	0.0068	0.0001	0
bd	0.2136	0.0071	0	-0.4075	0.2595	0.2595
abd	0.0853	0.0011	0	-0.0571	0.0051	0
cd	0.1009	0.0016	0	-0.1198	0.0224	0
acd	0.0515	0.0004	0	0.1226	0.0235	0
bcd	0.3181	0.0158	0	0.1991	0.0620	0
abcd	-0.0845	0.0011	0	0.1379	0.0297	0
e	-0.1194	0.0022	0	-0.0743	0.0086	0
ae	0.0302	0.0001	0	-0.0300	0.0014	0
be	-0.1226	0.0023	0	-0.0920	0.0132	0
abe	-0.0011	0.0000	0	-0.0599	0.0056	0
ce	-0.0526	0.0004	0	0.0324	0.0016	0

ace	0.0897	0.0013	0	0.0156	0.0004	0
bce	-0.0114	0.0000	0	-0.0027	0.0000	0
abce	0.0842	0.0011	0	0.0286	0.0013	0
de	0.0958	0.0014	0	0.0375	0.0022	0
ade	0.0191	0.0001	0	0.0069	0.0001	0
bde	0.0937	0.0014	0	0.0246	0.0009	0
abde	-0.0113	0.0000	0	-0.0158	0.0004	0
cde	-0.2902	0.0132	0	-0.0867	0.0118	0
acde	-0.0033	0.0000	0	-0.0168	0.0004	0
bcde	-0.2483	0.0096	0	-0.1158	0.0209	0
abcde	-0.0092	0.0000	0	-0.0093	0.0001	0
f	4.6031	3.3107	3.3107	1.5134	3.5787	3.5787
af	-0.1970	0.0061	0	-0.0404	0.0025	0
bf	2.4893	0.9682	0.9682	0.7175	0.8044	0.8044
abf	-0.0911	0.0013	0	-0.0556	0.0048	0
cf	-0.1110	0.0019	0	-0.1914	0.0572	0
acf	0.1257	0.0025	0	0.0104	0.0002	0
bcf	0.0806	0.0010	0	-0.0415	0.0027	0
abcf	0.0411	0.0003	0	0.0173	0.0005	0
df	2.2823	0.8139	0.8139	0.6981	0.7614	0.7614
adf	0.0003	0.0000	0	0.0191	0.0006	0
bdf	0.1705	0.0045	0	-0.0989	0.0153	0
abdf	0.1052	0.0017	0	-0.0016	0.0000	0
cdf	0.0780	0.0010	0	-0.0210	0.0007	0
acdf	0.0608	0.0006	0	0.0373	0.0022	0
bcdf	0.2696	0.0114	0	0.1275	0.0254	0
abcdf	-0.0227	0.0001	0	0.0506	0.0040	0
ef	-0.3314	0.0172	0	-0.0827	0.0107	0
aef	-0.0308	0.0001	0	-0.0319	0.0016	0
bef	-0.2313	0.0084	0	-0.0764	0.0091	0
abef	0.0016	0.0000	0	-0.0256	0.0010	0
cef	-0.1305	0.0027	0	0.0112	0.0002	0
acef	0.0634	0.0006	0	-0.0103	0.0002	0
bcef	-0.0451	0.0003	0	-0.0188	0.0006	0
abcef	0.1046	0.0017	0	0.0257	0.0010	0
def	-0.0080	0.0000	0	0.0375	0.0022	0
adef	0.0124	0.0000	0	0.0208	0.0007	0
bdef	0.0913	0.0013	0	0.0395	0.0024	0
abdef	0.0443	0.0003	0	0.0244	0.0009	0

14.5 Server Serialization

Factor	AVG Cont. x10 4	SS x106	Red. SS	SDev Contrast x10 3	SS x10 4	Red. SS
a	0.1042	0.0170	0	0.8216	1.0548	1.0548
b	1.9092	5.6956	5.6956	2.5046	9.8019	9.8019
ab	0.0535	0.0045	0	0.7675	0.9204	0.9204
c	0.1039	0.0169	0	0.1019	0.0162	0
ac	0.0468	0.0034	0	1.2623	2.4897	2.4897
bc	0.0520	0.0042	0	0.2283	0.0814	0
abc	0.0396	0.0025	0	0.7084	0.7842	0.7842
d	2.3222	8.4261	8.4261	2.0342	6.4657	6.4657
ad	0.0505	0.0040	0	0.3925	0.2407	0
bd	0.0217	0.0007	0	0.1757	0.0482	0
abd	-0.0008	0.0000	0	0.2361	0.0871	0

cd	0.0722	0.0081	0	0.7977	0.9944	0.9944
acd	0.0401	0.0025	0	1.1975	2.2408	2.2408
bcd	0.0197	0.0006	0	0.8103	1.0260	1.0260
abcd	0.0338	0.0018	0	0.8014	1.0036	1.0036
e	0.7940	0.9851	0.9851	2.3445	8.5884	8.5884
ae	0.0775	0.0094	0	0.8416	1.1067	1.1067
be	0.3726	0.2169	0	1.1050	1.9079	1.9079
abe	0.0342	0.0018	0	0.3319	0.1721	0
ce	0.0757	0.0090	0	0.7250	0.8214	0.8214
ace	0.0013	0.0000	0	-0.0231	0.0008	0
bce	0.0379	0.0022	0	0.2122	0.0703	0
abce	0.0029	0.0000	0	-0.1398	0.0305	0
de	0.4414	0.3045	0	1.0382	1.6842	1.6842
ade	0.0352	0.0019	0	0.3900	0.2376	0
bde	0.0208	0.0007	0	-0.1164	0.0212	0
abde	-0.0081	0.0001	0	-0.1524	0.0363	0
cde	0.0412	0.0027	0	0.3139	0.1539	0
acde	0.0063	0.0001	0	-0.0826	0.0107	0
bcde	0.0035	0.0000	0	-0.2325	0.0844	0
abcde	0.0083	0.0001	0	-0.1109	0.0192	0
f	0.0122	0.0002	0	0.0188	0.0006	0
af	0.0080	0.0001	0	0.1130	0.0199	0
bf	0.0053	0.0000	0	0.0369	0.0021	0
abf	0.0139	0.0003	0	0.1580	0.0390	0
cf	0.0110	0.0002	0	0.1149	0.0206	0
acf	0.0098	0.0002	0	0.0462	0.0033	0
bcf	0.0091	0.0001	0	0.0853	0.0114	0
abcf	0.0100	0.0002	0	0.0104	0.0002	0
df	0.0066	0.0001	0	-0.0189	0.0006	0
adf	0.0006	0.0000	0	0.0835	0.0109	0
bdf	-0.0002	0.0000	0	0.0477	0.0036	0
abdf	0.0063	0.0001	0	0.0598	0.0056	0
cdf	0.0144	0.0003	0	0.2072	0.0671	0
acdf	0.0089	0.0001	0	0.0423	0.0028	0
bcdf	0.0124	0.0002	0	0.1016	0.0161	0
abcdf	0.0091	0.0001	0	0.0482	0.0036	0
ef	0.0043	0.0000	0	0.0285	0.0013	0
aef	0.0016	0.0000	0	-0.0489	0.0037	0
bef	-0.0048	0.0000	0	-0.1706	0.0455	0
abef	0.0037	0.0000	0	0.0650	0.0066	0
cef	0.0015	0.0000	0	-0.1691	0.0447	0
acef	0.0041	0.0000	0	0.0151	0.0004	0
bcef	0.0041	0.0000	0	0.0458	0.0033	0
abcef	0.0028	0.0000	0	-0.0473	0.0035	0
def	0.0039	0.0000	0	-0.0139	0.0003	0
adef	-0.0050	0.0000	0	-0.0321	0.0016	0
bdef	-0.0046	0.0000	0	-0.1013	0.0160	0
abdef	-0.0035	0.0000	0	-0.0527	0.0043	0

14.6 Server Deserialization

Factor	AVG Cont. x10 5	SS x108	Red. SS	SDev Contrast x10 4	SS x10 7	Red. SS
a	0.5586	0.4876	0.4876	2.5628	1.0262	1.0262
b	-0.0736	0.0085	0	-0.9093	0.1292	0.1292
ab	-0.0699	0.0076	0	-0.9141	0.1306	0.1306
c	1.2211	2.3298	2.3298	1.7963	0.5042	0.5042
ac	0.0511	0.0041	0	1.4397	0.3239	0.3239
bc	-0.0778	0.0095	0	-0.9901	0.1532	0.1532
abc	-0.0738	0.0085	0	-0.9717	0.1475	0.1475
d	0.0316	0.0016	0	0.5211	0.0424	0
ad	0.0412	0.0026	0	0.5255	0.0431	0
bd	-0.0122	0.0002	0	0.0484	0.0004	0
abd	-0.0091	0.0001	0	0.0392	0.0002	0
cd	0.0253	0.0010	0	0.3888	0.0236	0
acd	0.0349	0.0019	0	0.3925	0.0241	0
bcd	-0.0161	0.0004	0	-0.1070	0.0018	0
abcd	-0.0128	0.0003	0	-0.0943	0.0014	0
e	0.2717	0.1153	0	0.8728	0.1190	0.1190
ae	0.0900	0.0126	0	0.7662	0.0917	0.0917
be	0.0138	0.0003	0	-0.1027	0.0016	0
abe	0.0154	0.0004	0	-0.1110	0.0019	0
ce	0.1931	0.0583	0	0.7187	0.0807	0.0807
ace	0.0116	0.0002	0	0.6225	0.0606	0.0606
bce	0.0139	0.0003	0	-0.1103	0.0019	0
abce	0.0156	0.0004	0	-0.1089	0.0019	0
de	0.0163	0.0004	0	0.2754	0.0118	0
ade	0.0199	0.0006	0	0.2726	0.0116	0
bde	-0.0092	0.0001	0	-0.0194	0.0001	0
abde	-0.0076	0.0001	0	-0.0158	0.0000	0
cde	0.0153	0.0004	0	0.3061	0.0146	0
acde	0.0191	0.0006	0	0.3131	0.0153	0
bcde	-0.0092	0.0001	0	-0.1090	0.0019	0
abcde	-0.0075	0.0001	0	-0.0949	0.0014	0
f	0.0041	0.0000	0	-0.0932	0.0014	0
af	-0.0111	0.0002	0	-0.2246	0.0079	0
bf	0.0150	0.0004	0	0.3035	0.0144	0
abf	0.0154	0.0004	0	0.2953	0.0136	0
cf	-0.0031	0.0000	0	-0.1126	0.0020	0
acf	-0.0183	0.0005	0	-0.2305	0.0083	0
bcf	0.0118	0.0002	0	0.2710	0.0115	0
abcf	0.0123	0.0002	0	0.2831	0.0125	0
df	0.0410	0.0026	0	0.4285	0.0287	0
adf	0.0404	0.0025	0	0.4278	0.0286	0
bdf	-0.0374	0.0022	0	-0.3886	0.0236	0
abdf	-0.0379	0.0022	0	-0.3949	0.0244	0
cdf	0.0383	0.0023	0	0.3669	0.0210	0
acdf	0.0377	0.0022	0	0.3629	0.0206	0
bcdf	-0.0360	0.0020	0	-0.3230	0.0163	0
abcdf	-0.0366	0.0021	0	-0.3396	0.0180	0
ef	0.0090	0.0001	0	-0.0536	0.0004	0
aef	0.0130	0.0003	0	0.0234	0.0001	0

bef	-0.0132	0.0003	0	0.0427	0.0003	0
abef	-0.0111	0.0002	0	0.0499	0.0004	0
cef	0.0093	0.0001	0	0.0265	0.0001	0
acef	0.0132	0.0003	0	0.0892	0.0012	0
bcef	-0.0125	0.0002	0	-0.0110	0.0000	0
abcef	-0.0105	0.0002	0	-0.0044	0.0000	0
def	-0.0074	0.0001	0	0.0960	0.0014	0
adef	-0.0077	0.0001	0	0.0923	0.0013	0
bdef	0.0078	0.0001	0	-0.1302	0.0026	0
abdef	0.0065	0.0001	0	-0.1151	0.0021	0