Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN
"SCIENZE DELL'INGEGNERIA"

CICLO XXII

COORDINATORE Prof. Stefano Trillo

WEB DISTRIBUTED COMPUTING SYSTEMS

Settore Scientifico Disciplinare ING-INF/03

Dottorando
Dott. Boldrin Fabio

(firma)

Tutore
Prof. Mazzini Gianluca

(firma)

Anni 2007/2009

Contents

1 Intrduction 1

1.1 Definitions . . . 1

1.1.1 Distributed computing . . . 1

1.1.2 Web browser . . . 2

1.1.3 AJAX . . . 3

1.1.4 Adobe FLEX® . . . 6

1.1.5 AJAX and Adobe FLEX®comparison . . . 7

1.1.6 Web 2.0 . . . 7

1.2 Research project . . . 10

1.2.1 Web Distributed Computing Systems . . . 10

1.2.2 Mathematical model . . . 13

1.2.3 Implemented solution comparison . . . 13

2 Architecture 15

2.1 Client . . . 15

2.2 Server . . . 16

2.3 Network communication . . . 17

2.4 Data . . . 17

2.5 Reingegnerizzazione . . . 18

3 Implementation of the distributed solution 21

3.1 Problems . . . 22

3.1.1 RSA cryptosystem crack . . . 22

3.1.2 Pearson's correlation . . . 23

3.2 Computation client . . . 24

3.2.1 AJAX Client . . . 24

3.2.2 Flex Client . . . 25

3.3 Server . . . 26

3.3.1 Scheduling . . . 28

3.3.2 Recovering . . . 28

4 Metrics and measurement **31**
4.1 Definitions 31
4.2 Metrics 32
4.2.1 PTR 33
4.2.2 DER 33
4.2.3 TER 33
4.2.4 Metrics relationships 34
4.3 Data logging 35

5 Experimental results **39**
5.1 Testing environment 39
5.2 Metrics comparison 40
5.2.1 RSA Cracking 40
5.2.2 Pearson's correlation 41
5.2.3 Results 42
5.3 Implementation comparison 43
5.4 Results 47

6 Mathematical model **49**
6.1 Model parameters 50
6.2 Model description 51
6.3 Examples 55

7 Conclusions **59**
7.1 Summary of the research work 59
7.2 Ulteriori sviluppi 60

Bibliografia **65**

List of Figures

1.1 Comparison between the conceptual schemes of a standard web page and an AJAX application 4
1.2 Web 2.0 map - Web 2.0, fundamental concepts, applications e sites - from Wikipedia 9

5.1 Crunch selection from the database 44
5.2 Distribution of computation time for the AJAX client data . . 45
5.3 Distribution of computation time for the Flex client data . . . 46

List of Tables

1.1 AJAX vs Flash/Flex comparison table [10] 8
1.2 Web 1.0 e Web 2.0 key concept evolution. 10

5.1 Metrics: factorization and correlation 41
5.2 Average time detected 44
5.3 Metrics calculated by type of client 46

List of code

2.1 scheduled function 20
3.1 AJAX scheduled function 25
3.2 Flex scheduled function 27
3.3 getCrunch function 28
4.1 t_{get} java implementation 36
4.2 t_{get} php implementation 37
4.3 t_{post} java implementation 37
4.4 t_{post} php implementation 38

Chapter 1

Intrduction

This first chapter presents the problem studied during the research work concerning the usage of web systems to perform distributed computing, making some preliminarly remarks on some propaedeutics definitions before describing the project developed during the study course.

After the introduction it will be presented the research objectives, giving the details and the solutions developed for the implementation of the realized and employed system.

Then data collected for the studied solutions will be shown, together with the summary and comparison of different test cases in different situations and configurations, underlining measurements and convenience values, those values that make the distributed system usage convenient compared with the direct solution.

After the system definition and implementation the focus will move to the mathematical description of the studied problems. This part took to the development of a mathematical model to identify and describe different classes of problems, to discover feasibility, convenience and resolution performance.

The treatise, in conclusion, will present real applications and fields where the system could be conveniently emplyed with good performance of resolution.

1.1 Definitions and preliminar concepts

1.1.1 Distributed computing

The term *distributed computing* states for the resolution of a problem dividing it into a number of smaller parts, called sub-problems, each of them is solved separately by a different machine (or CPU) that compute the sub-problem solution in a parallel and independent way.

The results obtained by the computation, or in other words the sup-problem solutions, are then collected and combined together by another machine, typically a server machine, that first decomposed the original problem into the sub-problems.

The main field of application of this kind of approach is the solution of complex problems, those problems unsolvable in practical due to the amount of time required to find the solution with a single CPU, doing the computation monolitically.

By dividing the original problem into more smaller parts there is the possibility to make the computation of the single sub-problems parallelized, in fact singles machines (CPUs) partecipating to the computation can run independently from each other while a single machine has to compute the solution linearly, under the hypothesis of a single processor.

It comes out, obviously, that the computing power of a single machine is constraind by the available technology, while on the field of distributed computing the constraint is represented by the number of available clients, that is looser than the first.

Furthermore, on the other hand, it is not always possible to utilize distributed computing solutions to solve all kind of problems: there are classes of problems, in fact, that can't easily divided into separate sob-problems, or even not separable at all.

Finally, thare are classes of problems that give optimal performances with this approach and other classes that need fine tuning to be solved in a conveninet way, as exposed hearinafter.

1.1.2 Web browser

The *web browser* is well known as the main hypertext navigation tool. In recent years it has further developed to process a wide range of documents, written in a growing number of different languages and markup standards.

The role of the browser into this research project has a wider meaning: it is not only the browsing tool but also the limits and the container for the whole application. It is not possible for the application developed to overcome the boundary imposed by the browser process [1, 2].

This constraint is very important and has to be taken into account the fact that there are a number of limitations in using machine resources through browsers, in particular it is not possible to control external resources out of the browser sandboxes boundaries. It is not permitted to have control over the performances using cpu loading measurements or to access local machine resources not directly included into the current document.

To avoid excessive limitations it is necessary to exploit the available limited resources and design the system in order to let it to be as autonomic as possible so it does not need any external resources.

1.1.3 AJAX

AJAX is an acronim that states for Asynchronous Javascript and XML. It has been introduced by Jesse James Garrett of Adaptive Path [3]; it is not a true new technology in the proper term sense but rather an integration of existing technologies to obtain new functionalities.

The web application concept changes: the application is not only a set of web pages linked together by a hypertextual navigation logic following a predetermined path but instead something more similar to a stand-alone desktop application, with all the advantages in terms of usability and functionalities (see fig. 1.1).

The more characterizing concept of the old approach is the loss of the current view. When a hypertextual link is clicked, a form sent or any other action performed the effect is to be transferred to a new page, statically or dinamically generated, but always with the effect for the user of loosing the page view for a certain time, while the new resource is loaded. This effect is called flickering and mainly depends on the network speed and congestion: the slower the network is, more this effect is underling and evident.

AJAX reinvent this concept, exploiting in a new innovative way the integration of technologies to create clients that are not affected by this limitation in usability.

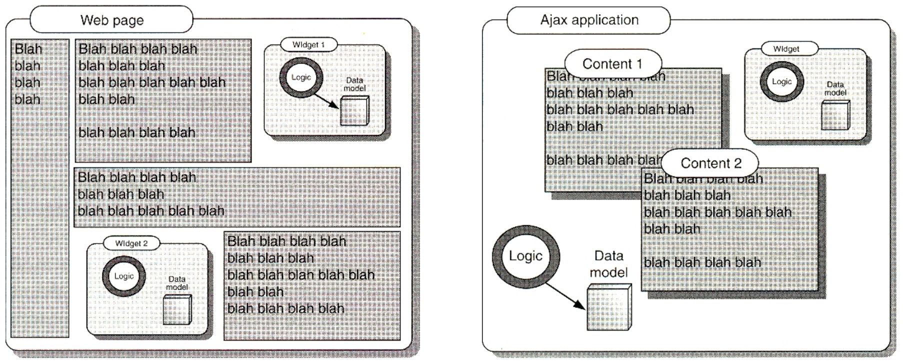
Using functionalities offered by Javascript language, combined together with DOM the *Document Object Model (DOM)* and the *Browser Object Model (BOM)* it is possible to control the document hierarchy and composition and also the browser behaviour programmatically.

In particular, accessing the DOM via javascript it is possible to modify the current document structure at runtime, using for example HTML information dynamically generated on user events, computation results or other conditions.

The flickering effect can be avoided and the user interface become *rich*, increasing interactivity in the direction of a desktop application user experience [4].

Ajax rich client

The richness of a client built with AJAX technoloies derives from the concept of*RIAs, rich internet applications*, that indicates a web page based applica-



(a) ***Web page*** **(b)** ***AJAX Application***

Figure 1.1: Comparison between the conceptual schemes of a standard web page and an AJAX application

tion that has an advanced interaction model with the end-user, sensitive to different kind of inputs and characterized by fast and intuitive responses to user actions [5, 6, 7].

To better give the idea the goal of a rich ajax client, and more generally of a rich internet application, is to resemble to an application such as a spreadsheet or a word processor, in terms of user action reactivity.

The important difference between a desktop application and a web application is on the fact that while the first typically executes into a closed environment, the second executes into a client-server environment, where users interacts with clie client-side part of the application.

The client-side components, in turn, communicate with other separated processes, generally executed in a completely different environment.

Focusing on the server side of tha application, the modern multi-tier architectures divide the server into many different components, usually not visible from the external side of the application itself.

AJAX application, acting as an *"empty"* client (thin client [8]), interacts with server side viewing it as a black-box, without any knowledge over the internal architecture, neither having the necessity to.

Considering the interaction between server and client it has to be considered also the network infrastructure: while a desktop application all the resources and processes used are local, in a web application the network plays an important role acting often as bottleneck in presence of slow and unreliable communications.

This is the typical issue to solve when designing and implementing interactive and application that have to be responsive for the end-users inputs.

The heart of AJAX is the possibility to create Javascript objects that represents asynchronous HTTP requests. Thanks to this kind of object it is possible to communicate with remote servers to recover many types of data.

The asynchronous characteristic permits moreover to avoid the current page refresh and flicker like in the usual hypertextual navigation style.

All the problems presented has to be faced when implementing a "rich" client to obtain fast responses to the user action and inputs and to model many different type of interaction, to build applications based on a model that is as mych similar as possible to a classical "monolitical" desktop application instead of the hypertextual browsing one.

All the concepts presented can be summarized into the so called 4 AJAX principles [9]:

Browser encapsulate application, not contents: a standard web applicationbased on hypertextual web-pages navigation acts as an empty terminal without any knowledge of the current session.

All the session information are hosted by the server that has the task to create session information when user logs-in on the first page, then it updates the session information according to the user actions and at the end, when the user logs-out or closes the browser (and consequently the connection), it destroys the session information freeing resources associated.

In this scenario the user has no active part in all the workflow, all the process is completely transparent to him.

Ajax adds a new element to the scene: the browser has an active part in the workflow hosting parts of the application logic using Javascript.

It is not necessary to reload the page during the various interactions because it is possible to update it without refresh through AJAX, so it is possible to keep tracking of user actions and choices during all the session time, like in a classic desktop application.

The server sends data, not contents: the code of the application logic of the client side is sent a single time at the beginning of the session in a single solution.

After the first client download, since all the necessary application code is locally present it is sufficient to transmit between client and server only the chages and the application data.

This can be done by transferring only portion of code for the client sub-functionalities or data fragments to insert into the document.

Continuous and fluid interaction: with classic web pages the user input is restricted to hypertextual links and form posting.

Those reduced possibilities limit the field of interactions between user and application.

AJAX, on the other hand, permits to exploit at best the interaction possibilities like keypresses, drag&drop, mouse movements.

All these option add at first interactivity, as stated before, and then also the option of a different management of the application logic.

For example, when browsing a page a form post breaks the user workflow because of the page refresh; using other events, such as browser and mouse ones, it is possible to avoid the workflow break and keep the user experience linear.

Programming discipline: AJAX is real programming, with rules to respect respect,patterns and best practices

Client has its specific own logic embedded into page that continue to work until user closes the window.

The application has to function without error for the whole operating time with no internal bugs but also without interferee with the host browser.

The same attention payed to the server application coding has to be employed to build up a robust and functional client that correspond to it. In particular it has to be noted that the code will not be on the server machine so it will be out of the server-side control.

1.1.4 Adobe FLEX®

Adobe Flex is a free of charge, high productivity opensource framework that can be used to build up interactive web application with uniform implementation over the principal web browser available, desktops and operative systems, based on the popular Adobe Flash Player runtime.

Usually the term Flash mean the well known player or the development IDE; Flash is actually a powerful application platform.

Flex, compared with the Flash IDE, is a more flexible development framework focused on the creation of application for the AIR and Flash runtimes.

Moreover, Flex opens up new design possibilities increasing common web browsers capabilities with the Adobe Flash technology that is at the base of Flex.

The Flex technology is now available on 2 different environments: Adobe Flash Player and Adobe AIR, who permits to Flex application to be executed on the user desktop, outside the common browser boundaries for the flash environment [10].

The framework capabilities permit to create powerful RIAs (rich internet application), a concept already intrduced before, that give a user experience similar to a desktop application.

The 2 main components of the development language are:

- MXML (Macromedia Flex Markup Language) – used to define the user interface elements and to define the data binding

- ActionScript (ECMA compliant scripting language) – this is used to define the business logic and to manage the interactions between the various components and the data.

1.1.5 AJAX and Adobe FLEX®comparison

In this section is presented a comparative table that shows the differences and the peculiarities of the 2 different languages used in the development (tab. 1.1)

1.1.6 Web 2.0

The term Web 2.0 comes out more or less in 2004, when it has been introduces at a O'Reilly Media conference [11, 12].

The terms states for a new web generation, with new applications and new services, linked together by a new web use philosophy.

It is not concerning about new programs or languages, nor architecture changings or upgrades of the present network in use: the focus this concept introduces is on the evolution of the way the network is consedered and used.

There are no new techlogy to apply but rather a new way to intend those already developed and widely used, integrating together to obtain new and more advanced functionalities, more powerful and easy to use.

The web becomes a platform, like a universe of principles and best practices to utilize in development and usage of the various applications (see fig. 1.2).

At the beginning the bases of this new kind of view the web environment have been defined, finding some comparison points between the "first" version of the network, called by analogy Web 1.0, and this new kind of approach.

	Description	AJAX	Flash/Flex
Animations	Animations support permits to developers to underline state changes, focus the user to the navigation path or simply to give a better navigation experience and entertain the user	Limited support to linear animations	Extended support
Bitmap manipulation	• Lets the user to modify bitmap graphics in a real-time way, directly on client side • Gives to the user the flexibility to create visual effects, distorsions and variation changes directly at runtime • Helps in supporting animation (moving effects and image distorsion)	Partial non-standard support for Opera, Firefox and Safari browsers. Achievable through server side processing	Natively supported
HTML Rendering	Under some circumstances and application it is useful to generate and present contents directly into che application. User can create a potentially high complexity HTML content also for other users	Full support	Limited support, it is not possible to utilize HTML tables, JavaScript, frames and other components
Video and Audio Streaming	Video contents are at present those of majar interest. There are a number of video contents on the web: tutorials, business communications, or simply entertainment.	Limited not native support, work in progres for the new HTML5 standard integration. Need to use external plugins, such Quicktime, Windows Media Player or Flash Player, and control them with AJAX	Supported with variable video quality. Video and audio capture also supported.
Development environments	An advanced development environment give the developers tools to create code quickly and so helps to diffuse the choosen techlogy widely.	• Google Toolkit - free • Echo 2 - free • jsLINB - free • Rico - free • Zapatec - 5400$ • many more, both free and commecial	• Flex Builder (Eclipse plugin) - 500$ • Flash CS3 - 700$
Runtime	The runtime is the environment that takes the code and convert it into actions. Inconsistencies, bugs and problems related to performances can considerably degrade the user navigation experience.	Each browser has a proper internal engine to interpret the code, under defined standards.	Unique platform, common to all different browsers that gives a plugin runtime to put into various implementations.

Table 1.1: AJAX vs Flash/Flex comparison table [10]

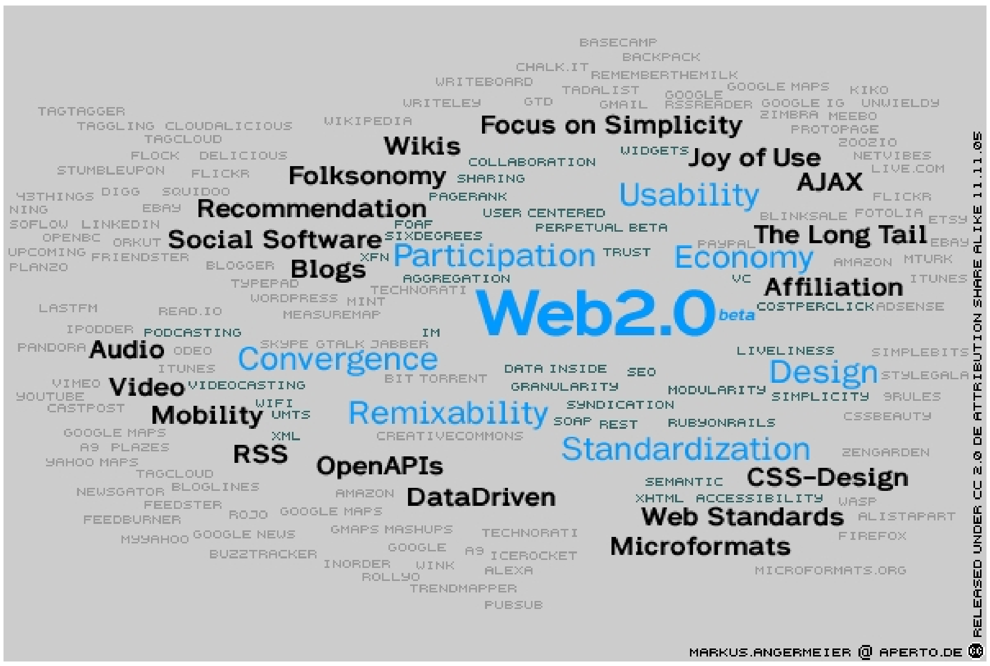


Figure 1.2: Web 2.0 map - Web 2.0, fundamental concepts, applications e sites - from Wikipedia

Web 1.0		***Web 2.0***
DoubleClick	$\Longrightarrow$	Google AdSense
Ofoto	$\Longrightarrow$	Flickr
Akamai	$\Longrightarrow$	BitTorrent
Britannica Online	$\Longrightarrow$	Wikipedia
Siti personali	$\Longrightarrow$	Blog
Speculazione sulla registrazione dei domini	$\Longrightarrow$	Ottimizzazione dei motori di ricerca
Pagine viste	$\Longrightarrow$	Costo per clic
Pubblicazione	$\Longrightarrow$	Partecipazione
Content management systems	$\Longrightarrow$	Wiki

Table 1.2: Web 1.0 e Web 2.0 key concept evolution.

The main elements of this 2 different version of the web have been identified with the most important sites and services, reference of the whole Internet community.

Those summarized on table 1.2 are only some of the differences found for the 2 different web versions.

The main aspect that come out from the analysis of the comparison of services of the same type is the way the service is intended when moving from the "old" Web 1.0 to the new Web 2.0 version: the focus is on the services and information integration to create a dynamic platform to navigate, different from the concept of static web page.

The debate over the Web 2.0 is still opened and heated, there is also the question if this concept really exist, since in fact there is no new technology and any new thing is introduced under the pratical aspect, this is only a new idea and a way of definition.

A remark point over the Web is the activity and the debate it generates: this stimulates the continuous innovation toward new ways of use the network to provide new services that are innovative, useful and expecially easy to use.

1.2 Research project

1.2.1 Web Distributed Computing Systems

The reasearch project comes out from the development of a distributed computing system that uses common web browsers as client, focusing on the fact that no specific client installation is necessary.

Literature on distributed computing offers multiple cues of study [13, 14], and many solutions have already been implemented, such as [15, 16].

Many works focus on some particular fields covered by the distributed system world: for example relating to the network performances, in terms of connectivity [17] or management of bandwidth [18].

There are solutions based on the usage of virtual machines [19], in combination with overlay networking and peer-to-peer techniques to create networks of virtual workstations for high-throughput computing.

Many systems based on client-server design pattern have grown up in recent years, for example the Folding@home project [20] of the Stanford University Chemistry Depart- ment, or the SETI@home [21] of the Space Sciences Laboratory at the University of California, Berkeley and also the LHC@home [22], a project by the CERN for simulation on the new Large Hadron Collider.

However, literature lacks on the field of web distributed computing systems, those based on web browser computation capabilities.

This is the reason that guided the PhD research: study new possibilities and in particular new ways to do distributed computing, relying on web browsers potentials on the client side to solve various kinds of problems.

The fact this field is relatively unexplorated has obviously pros and cons: cons are, as stated, mainly the lack of resources to base on to develop the study.

Pros are surely the possibility to explore a completely new field, with a wide range of paths to follow, starting from the beginning with definitions, designs and implementations.

This particular characteristic guided all the research work and also has required the development of new specific solutions, based on AJAX and Flex technologies.

The developed solutions have been then compared and analyzed, to discover the various implemented solution performances, to find if among this solutions there is one better than the other and in case which of them.

A mathematical model have been developed based on the research study and on the implemented solutions, to characterize different classes of problems.

Then identy if a particular problem can be efficiently and conveniently solved with good performance with the developed system before implementing it, but rather using results of the model characterization.

A brief description of the various themes developed follows, before describing with detail each single argument in subsequent chapters.

Web browser computing

The main argument of the research is, as mentioned, the development of a distributed computing system that could function embedded into the common web browsers.

Under this assumption it is not neccessary any installation of additional client and it is possible to exploit, to the contrary of other distributed solutions, a large number of potential client, in practice every machine connected to the Internet network and used to browse web sites.

The developed solutions, both in the AJAX case and using the Abode Flex technology, had to focus particularly on 2 fundamental aspects:

- limited computing ability provided by both AJAX and Flash/Flex virtual machines, overall the lack of multithreading.
- the constraints on the resources usage. The system can not interfere with the user operations. Computation runs during the user navigation, so not on the idle intervals but during the normal activity. The computation runs in backgroung and obviously can not occupy resources user is employing.

Those characteristics forces the adoption of some particular solutions in the definition of computing cycles, details over this reenginering has been implemented will be shown in next chapters.

The practical development studied the feasibility of the system over the problem of large integer number factorization, that is one of the fundamental steps for the RSA cryptosystem cracking algorithm.

This kind of problem, in fact really simple and trivial, started the implementation of the whole system.

The system has been then enhanced and tuned by solving another problem: the correlation computation over samples relative to a genetic database using the Pearson's correlation formula.

The choose has not been random: this kind of problem, the correlation computing, needs the transmission of larga data packet in front of a relatively limited amount of computation over this data.

Moreover it is worth to underline that this kind of solution could be inserted into the field of autonomic systems [23, 24], because of it has the ability to adapt its behaviour to the environment for both server and client components, increasing reliability and efficiency of the system itself.

1.2.2 Mathematical model

Together with the implementation of the distributed system it has been studied also the characterization and the classification of problems.

Using among the various input parameters also the resolution algorithm implementation it has been realized a mathematical model to describe performances and conveninence of a particular implementation, with the objectives:

(a) identify the convenience of a particular implementation prior to realize it, obviously with the intent to save time and resources, focusing on implementation that are guaranteed to give better results.

(b) optimize a particular algorithm acting on elements that can modify the convenience values of the system.

Measurements

A set of parameters has been defined to measure the overall system performances. These parameters are then been used into the mathematical description of the various problems and for its optimization.

1.2.3 Implemented solution comparison

After the implementation of the whole system with 2 different languages, the first using AJAX technology and the second with Adobe Flex language, the solutions have been analyzed and compared to each other.

The final research goals can be summarized as follow:

(a) Identify the best solution, or in other words which technology is better to employ to build up the system. Identify pros and cons of the technologies to exploit them to obtain better performances, under different aspetcs.

(b) Demonstrate that the choosen technology used for the client-side computation is completely independent from the technology used for the server side part that has the task to provide data, manage the persistence and the transmission.

The following chapters will show details about:

- Realized system architecture: client, server, network elements and data persistence.
- Studied implementation characteristics, with details about the 2 problems analyzed during the development and the tests of the system.

- Definition of metrics used to measure the various system parameters
- Experimental results obtained during test execution performed in an Internet environment.
- Mathematical model developed to describe and categorize problems, with details about the model construction and with a particular focus on the algorithm and the problem, used as input for the definition od the system dimensioning.
- Conclusions, with the summary of the results obtained during the research work and some guideline proposals on the implementation and problem dimensioning.

Chapter 2

Architecture

The general application architecture follows the classic client-server model: client, embedded into a web page, requests data, computes locally the solution and sends back to the server the result of the computation.

This process is iterative and continues as long as the client keeps the connection alive, or in other words, the browser opened on the pae hosting the client.

The server side has the task to coordinate clients work, distributes computation data to the various clients, keeps track of already computed parts and of the solutions and recovers lost data when errors or malfunctions happen.

It is worth to underline that this kind of approach does not rely on the compational power offered by the single client, instead it relies on the number of machines that take part in the computation process.

Every single processor is really barely used, to ensure that end-users do not have any kind of interference in their normal usage of computer while the computation takes place.

Therefore the whole performances are given by the extremely large number of potencial clients instead of the single machine compation power.

2.1 Client

The client part of the service, implemented in 2 different versions with both AJAX and Flex, is the object that has the task to solve the single subproblem, called “crunch”.

Exploiting the Javascript functionalities for the AJAX solution and using Actionscript in the Flex implementation an iterative dialog between client and server is established, based on timed scheduling.

At every iteration the client query the server side to obtain data to compute and process those data according to a cycle schema that permits to control the execution speed, to avoid to consume too much resources.

At the end of computation the client query again the server side to send back proccess results and repeat the iteration from the beginning.

This process is controlled by setting some temporal parameters for the cycle to keep low the whole time necessary to perform a complete request-response cycle.

In particular the whole cycle time has to be kept under the limit of a few seconds, so it is possible to ensure that at least 1 complete computation cycle is performed by every user that connects to the pages hosting che client.

2.2 Server

Server side of the application has the same importance, since it has the task to manage the partitioning process of the problem, called “crunching”.

The server also has to manage the solutions collecting process and their persistence, needed to build up the overall problem solution after all crunches have been solved.

The task of the server component of the application can be resumed with the “scheduling” of the problem, that has the following components, as introduced above:

- **crunching**: the problem has to be parted into subproblems (crunches) each of them is assigned to a different client that connects to the application.

  This process can be dynamic, so did at runtime when the client query for a cruch to solve, or static, did by preparing and saving crunches prior to start their distribution to clients.

- **results collection**: this process includes all the management of crunches assignement to clients, the sending operations, the management of communicatio errors both sending and receiving data, the management of lost crunches recovering, those for whom there is not a solution available, because of errors in client, server o transmission.

- **results consolidation**: once single crunches results are collected it is necessary to compute the origianl global problem solution. In fact a local (crunch) solution not necessary is a global (whole problem) one so single subsolutions has to be treated to obtain the final solution.

2.3 Network communication

All communications between browser hosted client and management components on the server side take place through absolutely standard HTTP requests sent by appropriate objects that allow to perform an asynchronous non-blocking communication between client and server.

The asynchronous non-blocking communication is another of the key aspects of the research, since one of the main goals of the whole system is the background execution of the processes and data tranfer is one of the components of them.

When usign Javascript the object that has the task to perform the asynchronous requests is the **XMLHttpRequest**: this kind of object create exactly asynchronous requests that allow to the client to communicate with the server side without the need to fully reload the displayed page.

The user can continue browsing without any intrusion in his operations while the embedded client can continue its tasks.

On the other hand, the analogous element in the Flex environment is the **URLLoader** object, which has the same functionalities and behaviour for Flex of the XMLHttpRequest for Javascript.

A XMLHttpRequest object is able to natively manage XML content but effectively can vehiculate any type of content, from HTML to Javascript code to various text format and more.

It has to be pointed out that with this kind of request it is possible to send and receive among client and server only portion of data, XML, HTML or other, limiting to the minimum the network traffic.

In fact only interesting data are sent without need to provide complete content at every request.

2.4 Data

Last aspect to consider about the application, but not the least important one, is the data management for the application: server side dat persistence has to be guaranteed to keep track of the problem, of the computation, of the found solutions and of all other complementary data needed to manage crunches.

The persistence is guaranteed by a RDBMS (Relational DataBase Management System) with the following tasks:

- sto the problem data;

- store current crunches data. The server side keeps track of the problems currently assigned to the various clients, to be able to effectively manage the different errors that could happen, the redundance and in general all the aspects necessary to guarantee the correct resolution of all the problem crunches;

- store data concerning crunches in error or without solution, to be able to reassign them to another client;

- store the solutions that have been found by the clients.

Obviously it has to be underlined also the other data component that is part of the system: the packets transmitted between server and client.

The choosen format is XML because of its natively support by both AJAX and Flex objects, it is standard and it is also quite easy to implement the serialization and deserialization processes.

2.5 Algorithm re-engineering

Virtual machines embedded into browsers, relatively both to the Javascript browser engine and to the Flash plugin environment do not have the ability to exploit multithreading.

This fact takes to another one more importat and more constraining for the application developed: it is not possible to rely on a classic standard loop, such as for and while loops, to solve the problem.

Using a standard loop, in fact, the browser would look hanged until the whole cycle completes.

This is obviously a not acceptable scenario for an application with background activity as keyword: the system has to perform all its operations without impede the machine performances for the foreground user activities, using first of all limited resources to perform its tasks.

These fact made necessary a re-engineerization of the algorithms that perform the computation of a single crunch solution client side.

The aim of the re-engineerization is to find a way to control the CPU loading and the execution speed of the client application, to ensure that no application hangs or malfunctions happen to the browser while the computation take place.

So the focus returns again to the main aspect of all the research work: the need that the client end-users do not have any feeling about the computation taking place because this activity has always background non-invasive characteristic and remain always completely transparent for the user.

Loop instructions are used with very rarely exceptions in every problem resolution due to the iterative nature of almost all solving algorithms.

The classic while loop has been reviewed so it executes only a small *"inner cycle"* in a scheduled iterative way, so between 2 different schedulings there is a certain time of pause in the computation and so the control over the resources is released.

This *"atomic"* cycle has a little size and is composed by few instructions.

Its size is dimensioned to be executed in fractions of second, so the CPU resources are occupied monolithically for a time not appreciable by the user, then released for a while before restarting the execution.

By using this expedient and releasing the CPU execution control to the browser after a very limited interval the user experience and feeling is not deteriorated since the browser is always responsive to the user inputs.

To better describe the system in the rest of treatment it is convenient to define in a more rigorous way some terms only introduced until now:

- **Crunch:** defines the individual subproblem, fundamental unit sent from the server to each client and resolved separately on each request.

  The division of the original problem into crunches can be done in many different ways, depending on the type of problem, the size of the problem and its termination complexity. All these aspects must be considered to divide the original problem into crunches in a timely manner, convenient for the resolution.

- **External loop:** in practice this is the whole resolution algorithm of a crunch. Exactly as stated by the definition, this element is the outermost component of the resolution algorithm, which is scheduled when releasing control between a cycle and the next.

- **Inner loop:** specular to the external loop, inner loop represents the atom of computation, or in other words the part of algorithm executed monolithically by the computation client.

- **Operation:** this term indicates the base instruction of an inner loop. Considering an operation one of the 4 fundamental one among addition, subtraction, multiplication and division, all operations can be implemented by these four, considering the number of base operations that compose more complex instructions.

Now that the concepts necessary to the treatment have been clarified it is possible to move on to detailed discussion of the re-engineerization applied to the algorithms.

This re-engineering consists in the resolution of single crunch through the use of a double loop which includes both external cycles and internal cycles.

It should be noted that this double loop does not increase the whole termination complexity of the algorithm because it represents only a breaking of the single cycle of the original resolution of the problem into external and internal cycles defined above.

Each execution of the external loop is composed of an inner loop and a break and is iterated until the termination of the crunch under examination.

In contrast, the inner loop is used to control the CPU load on the client host that is solving a particular crunch of problem.

As mentioned above the inner loop has to be considered as the computational atom executed by the client.

Carried out a number of loops, number that depends on the complexity of the cycle, the computation of inner loops stops and the control is released from the scripting engine (AJAX or Flex) to browser that contains it.

The performance of inner loops must be designed to comply with a relatively small execution time , time in the order of fractions of second, interrupted by a pause time, with subsequent release of control to the browser, so the user does not suffer the influence the progress of calculation.

With reference to the pseudo code listed in 2.1, the algorithm of the client application is structured by an external loop consisting of k_A iterations, where each iteration consists in turn of by an inner loop composed by k_A iterations followed by a break with consequent release ot the control lasting for t_p seconds.

Code 2.1 scheduled function

```
for(kB iterations) {
   for(kA iterations) {
      # solution code
   }
   pause tP
}
```

Chapter 3

Implementation of the distributed solution

This chapter presents details about the implementation of the distributed computing solutions studied.

Here are the details of implementation of the server side, of client side, about the components of the transmission and the persistence, as presented in a more theoretical way in the introductory chapters.

The focal point certainly regards the client component architecture, as this part integrates the major points of innovation, the concepts of the algorithms re-engineering and of the particular approach to solution of the problem presented in this research.

Before delving into the architectural and implementative details is appropriate to emphasize the concept of independence of the approach studied by the particular technology used to implementation: a demonstration of this feature is underlined by the two different implementations realized, the first using AJAX and the second with Adobe Flex.

This chapter will show 2 different practical solutions implemented during the research work to test the general feasibility first and then the practical behavior of the system.

The server component, also independent of the technology implementation, includes the scheduling logic, whose implementative space is extremely broad, as we will see from the simplest iterative assignment to adaptive logics aimed to achieve the best performance.

3.1 Problems

The architecture of the web distributed system has been implemented on two different problems, each with its own characteristics features:

- RSA encryption system crack
- Pearson correlation analysis on genetic samples

3.1.1 RSA cryptosystem crack

The breaking of the RSA encryption system [25, 26] can be reduced in essence to a factorization problem of a “ large”integer n, product of two primes p and q.

This encryption system, which fits into the category of asymmetric key systems, uses a public key that is not secret which can be used to encrypt information that can then be deciphered only by using the corresponding secret private key.

With appropriate simplifications of the case the public key is the number n, the product of the two primes p and q, while the couple (p, q) is the secret private key for decryption.

The security the encryption relies on is the difficulty of factorization of integers that are the product of two prime numbers, the more the 2 primes are great the more will be difficult to find the 2 prime factors that make it up, or in other words, the greater the time required to find them will be.

On the other hand, it is obvious that once factored the number n into its factors p and q the secret key to decrypt a message encrypted with your public key becoma available to discover the content of an encrypted message.

After this mandatory brief introduction, and then traced the problem of break the encryption algorithm to a problem of factorization, it should be noted as at present there are no algorithms that allow to decompose a number into its prime factors efficiently and this is another point the security of the RSA cryptosystem relies on.

The research work aim was not to find and implement sophisticated algorithms to solve this problem, since the theme of the study is not to find algorithms with specific particular performance.

The problem has been entered into the system using the most obvious solving algorithm, the brute force one, trying to factorize the public key with all possible odd divisors.

Each client participating in the resolution seeks to factor the number n trying all the odd divisors of a given range, which is precisely the assigned crunch.

3.1.2 Pearson's correlation

The second problem addressed is to calculate the correlation between pairs of genetic samples taken from a database with the aim to identify possible similarities in order to find for example patterns of disease or mutations of the DNA.

In this case the problem is to assess the value of correlation between all possible pairs of samples of the database, identifying those with a value into a specific range that indicates a certain similarity between samples.

The resolution algorithm implements the Pearson's correlation formula [27]

$$r = \frac{\sum X, Y - \frac{\sum x, \sum Y}{N}}{\sqrt{(\sum X^2 - \frac{(\sum X)^2}{N}), (\sum Y^2 - \frac{(\sum Y)^2}{N})}} \tag{3.1}$$

While the first problem, based on the factorization, has been used as the basis for development, taking advantage of its intrinsical simplicity, the problem of calculating the Pearson's correlation is certainly more interesting from the standpoint of the system.

The calculation of the correlation, as discussed in detail hereinafter, is an example of an application that stresses much the system and highlights any limitations in terms of calculation on one hand and of transfer data on the other.

It is quite obvious that the system limit is reached when the data transfer time is greater than to computation.

Spending more time in the transfer of information rather than computing the solution of subproblems makes system not convenient, in favor of a direct resolution.

In a first work [28] published on this subject was described in detail the implementation of the system with AJAX.

Both solutions of two problems have been implemented using the re-engineering cycles of the algorithm, using the iterative client re-scheduling into the crunch resolution components.

In order to compare the two implementations with the differentl chosen languages, AJAX and Flex, the solutions implemented have been translated from one language to the other in order to obtain objects comparable in the measurements.

The two environments, Javascript and Flash/Flex, offer solutions based on approaches that are very similar, so it was easy to achieve essentially the same implementation with the two different languages, of course unless the specific characteristics of individual language which however, did not create particular problems in the implementation process.

3.2 Computation client

In this section is described one of the most important components of the whole system designed, maybe the most important one at all: the computing client embedded into the web pages that executes the background computation activities during the user navigation.

Details of the 2 different solutions developed follows, with the description of expedient used, common points and differences.

3.2.1 AJAX Client

Ajax technology is certainly one of the most prominent among those counted in the world of Web2.0, although not in itself something new but rather a union of other web technologies existing use with a much more developed.

Keeping fixed the idea to maintain and follow the pivotal point of Web2.0 philosophy, that is the *integration*, and of course the basic principles that have moved all the research work, namely the *lightness* and *non-invasiveness*, the appearance of the client is that of a common banner, inserted in different pages very easily.

The main features of the client can be highlighted as follows:

- the client occupies only a small fraction of the available space, for this reason it was chosen to show it as a banner, that can be plugged in so as not to occupy the foreground areas reserved for the main contents [29].

- The basic scheme of the pages is not distorted, the client is inserted into appropriate areas for the layout chosen, then without the need to provide adequate locations in which embed the computing client.

- visibility on the client is maintained in any case, details on the crunch resolution process are always available to the user and are also available controls that allows user to change the settings for the calculation.

- The client is extremely lightweight due to the background characteristic of the computation, is essential that the client commitments minimizing the CPU usage.

  This feature has the double advantage, beyond the limited use of the CPU, also that client could be transferred very quickly because of its size, and thus affects very marginal on the timing of data transfer, both for information contained on these pages, and with regard to the crunch data.

The code listed in 3.1 presents the basic skeleton of AJAX client for the resolution of a crunch. The computation end conditions are:

- *finished*: the solution of the crunch has been completed, the function ends and it is possible to send back to server results and then a new request, which in turn will call the function *f* again, starting the solution process for the next crunch.
- *ended*: the computation can always be interrupted acting on the user controls on the banner of the client. This operation should always be permitted so end-users have always freedom of choice whether or not to participate to the resolution of the problem.

Code 3.1 AJAX scheduled function

```
function f(params) {
    ...
    ** crunch inner cycle algorithm **
    ...
    if(!finished && !ended) {
        setTimeout(f, timeout); //last instruction
                                //timeout (int [ms])
    }
}
```

3.2.2 Flex Client

The implementation realized with Adobe Flex traces in good measure that achieved in AJAX.

It is also used in this case the re-engineering cycles resolution, using a timer for scheduling continuous inner loops, as presented.

Since the building architecture of the two solutions has no significant difference, it is worth noting the main differences between the two implementations:

- The execution environment of the Flex client is the Adobe Flash Plugin. The plugin is usually a browser add-on running properly compiled code.

- The plugin is browser-independent, in fact, Adobe distributes versions of the Flash runtime for all browsers and operating systems.
- The applications, because of they are compiled, are also independent of browser and operating system, since they are executed within the runtime, in a separate process but bound to program settings that hosts the plugin.
- The compiled code of the Flex client is considerably larger, in size, compared to the AJAX version. In fact, while the Flex client is just compiledd, the version of AJAX client is made of plain text files, which are then interpreted by the Javascript engine of the browser. The compiled code contains, in addition to the application logic (the algorithm of crunch resolution), even the components of the interface and the various Flash animations.

Other features of the client made in the language from Adobe are quite closely related to the AJAX universe, the ActionScript language has in fact many similarities to Javascript used in the first implementation.

The code listed in 3.2 shows the similarities in the implementation of Flex client compared to the AJAX version, despite the syntax and APIs used differ from each other.

3.3 Server

The server side of the application is obviously independent from the client, since the response to a request must be completely transparent to the client, regardless of its implementation.

The behavior is then identical for both requests from AJAX type clients and from those coded in Adobe Flex: to the request for a crunch the server will always respond by sending a data packet for a subproblem and will expect back a packet with data of the computed solution.

It should be noted that independence between the two sides of the system is mediated by the communication packets format, which obviously must comply with an appropriate standard to be defined a priori.

Back to the description of the server side of the system, its task is to schedule the division of the problem, distribute the crunch, manage the recovery of those subproblems that have been lost by errors of any type.

Because the system is client-server is not possible for the server control the behavior of clients, particularly it is not manageable the connection status between the two sides of the system.

Code 3.2 Flex scheduled function

```
public function f(event:TimerEvent):void {
    while( !inner_cycle_ended && !finished && !ended) {
         ...
         ** crunch inner cycle algorithm **
         ...
    }
    if(ended)
         return;
    if(!finisched) {
        // new inner_cycle loop
        timer = new Timer(timeinterval, 1);
        timer.addEventListener(TimerEvent.TIMER, f);
        timer.start( );
    } else {
        // crunch ended
        postresults( );
        timer = new Timer(5000, 1);
        timer.addEventListener(TimerEvent.TIMER, crunchrequest);
        timer.start( );
    }
}
```

At any time the connection between server and client can be interrupted, for reasons due to errors or even more trivially to the user disconnection.

Under these conditions the most convenient way to handle errors due to data loss is to consider a crunch as incorrect when the response is not received within a certain time, identified considering settings and configuration of the system.

Two different implementations have been realized also for the server system, mostly to show the actual independence of the two components (server and client) and to comply with the configurations of the environments in which the tests were conducted.

A first prototype has been built on Apache Tomcat, using components created in Java language. Then the same components were re-implemented in php scripting language, which best suited for publication in a public environment in the network of the University of Ferrara.

3.3.1 Scheduling

The scheduling is effectively the entry point for the various clients; the scheduling, implemented first by servlets and then with php scripting language, is responsible for receiving requests for a new crunch by the clients and for collecting the responses of the same clients containing the results of the computation. The scheduling algorithm is summarized in the pseudo-code listed in 3.3.

Code 3.3 getCrunch function

```
getCrunch() {
   if( lost crunches ) {
      crunch = recoverCrunch();
   } else {
      crunch = getNewCrunch();
   }
   saveCrunchInfo();
   return crunch;
}
```

3.3.2 Recovering

The second fundamental component of the server side is the recovering of crunch considered lost, or more generally in error.

The evaluation of the loss of a crunch is pretty delicate, as it should avoid false positives cases, which cause performance degradation, and of course it should also avoid not recognize subproblems actually in error.

This second part of the problem is quite simple to identify, in fact a subproblem is in error when the server does not receive receive back a response for a crunch, neither positive or negative

The first part is instead more delicate, because it must consider when evaluate a subproblem actually in error, or lost.

The simplest solution is to consider a crunch lost when a reply for that subproblem is not received within a certain time.

This time can be set as twice the total time normally necessary for the resolution of a crunch. Even assuming slow client or slow network is plausible that if a particular client does not send back results within this time it could disconnected or may have gone wrong for other reasons, causing the loss of the response.

However, this does not solve the problem entirely but moves it in determination of the "normal" time of calculation of a crunch.

This topic will be analyzed further in the definition of mathematical model, for now we just consider the fact that this time, that should not be too high, can however, vary depending on the problem and the algorithm implemented.

Returning to the recovering methods, after having established the conditions of recoverable crunches, these are essentially two:

- timed

- upon request.

With regard to the first mode, the timed one, recovering is done through a daemon that periodically queries the database containing the data of the current crunch assigned to any client and evaluate if any of these has exceeded the maximum time to be considered in error.

If the recovery conditions are satisfied for a cruch, this crunch is removed from the crunch queue of subproblems allocated and added to that of subproblems recovered to be resent to another client.

On the other hand, if recovery process is done at runtime, the scheduler evaluates the age of crunches when clients request new ones, recovering them when the crunch age is over a certain limit and by updating the age to now.

The two approaches can be considered equivalent, again discriminant depends on the particular configuration of a problem and scalability to be achieved.

Hitting too often reading and editing the same database table, in the case of the runtime approach, while certainly it makes easier to manage the application because it is simpler, on the the other hand may degrade system performances due to frequent accesses.

Chapter 4

Metrics and measurement

During the development of the computing system, regardless of problem faced some measures have been collected in order to quantify the overall performance and system peculiarities, evaluating various parameters that come into play during the resolution.

These measurements also are a prerequisite for the development of mathematical model that describes the system, detailed later in the discussion (see chapter 6).

In particular, measurements involving the transmission times, the computation time and the size of packets containing subproblems and solutions exchanged between clients and servers have been defined and collected.

The aim is to derive relations expressing a trade-off between time and size of packages and the various times in the game from the analysis of all these parameters.

The measures collected and analyzed, using the mathematical description of the solving algorithms for the problems, can be used to develop adaptation and optimization solutions to extend the class of problems that can be solved, meaning the problems which solution is convenient by the analysis of measures obtained.

4.1 Definitions

Before defining the sizes of the system that come into play in determine system performance, it is necessary to precede the definition of secondary variables that contribute to compose the metrics.

The components that come into play in the metrics used by system are the size of data packets both in input and output, that determine the network traffic, the time required to transmission of data over the network, either to or from the server, and finally those relating to the computation process.

The measurement of transmission time over the network can be done both from the standpoint of the client than the server.

It is reasonable to expect that there are no macroscopic differences since the time spent on the passage through the various levels of the stack is negligible compared to the real-time transmission on the network.

The computation time is instead purely relative to the client and to the processing necessary to calculate the single crunch solution.

Data used by the metrics we are going to define are the following:

- D_{in}: is the size of the client incoming packet, sent by the server in response to individual GET request the client sends to get a new crunch to compute.
- D_{out}: specular to the previous one, this is the size of the outbound packet from the client, containing the results of the crunch camputation. While for sizing the package data in a crunch it is possible to act server side, providing crunch with greater or smaller size, relative to response packet this is highly dependent on the way the client is implemented, and on the type and number of solutions found during the computation.
- t_{get}: GET time, is the time needed to client to receive (or similar to the server to send) the data of a crunch.
- t_{post}: POST time, is time required to client to send to the server the packet data with the results of a processed crunch.
- t_c: computation time, the total time needed to solve a subproblem. It should be stressed that this value is not the actual time of calculation, because it includes also the pauses between inner cycle necessary to release control and resources that are actually part of the computation process.

4.2 Metrics

After the necessary preliminary definitions the details on metrics that describe the system now follow.

Next will be also presented details of the relations that exist between different metrics, which permits the convertion from the different measurements to each other.

4.2.1 PTR

Using the values just defined we introduce the first of the system metrics:

$$PTR = \frac{D_{in} + D_{out}}{t_g + t_p} \tag{4.1}$$

Packet-to-transfertime ratio, the ratio between the size of packets sent and the time needed to send them.

This measure is in fact the actual speed of transfer, calculated taking into account all the variables that come into play during data transfer, such as network transfer errors, congestion and reconnections.

A high value of the metric indicates of course a fast network, where many data is transferred in a short time. Similarly, a small value indicates a slow network, where more time is needed to transfer the data.

The value of the PTR is useful for data normalization where measurements are carried out in different environments with different network speeds.

4.2.2 DER

$$DER = \frac{D_{in} + D_{o}ut}{t_c} \tag{4.2}$$

Data-to-evaluation ratio, ratio betweend size of packet exchanged and the time required to calculate the solution.

This parameter indicates the *usage* of data packets. This information is an indication about the quality of the algorithm. An algorithm that exploits deeply the crunch data will obviously be better than another that does not do it.

Optimizations in the solution algorithms can be measured through this metric.

It is assumed that this metric can not be controlled in an arbitrary manner because it depends on the particular characteristics of the problem faced.

Every problem has particular peculiarities which determine the order of magnitude of this metric.

The result can then be refined to improve performance, this is in fact the significant value of the measure.

4.2.3 TER

$$TER = \frac{t_g + t_p}{t_c} \tag{4.3}$$

> Transfer-to-evaluation ratio, the ratio between the time of data transfer and the time needed to solve the subproblem.

This third and final metric is the true measure of system performance subject of research.

The value of this metric provides information about the "convenience" to use the distributed web system developed to solve a problem.

An high value of this parameter indicates that the transmission time is too higher than the computation time, so most of the time available is used for network transmission rather than the calculation of the solution.

Obviously, this indicates that the proposed method is not convenient since the network transmission time can be considered "wasted" and should be limited as much as possible.

In a case like this it will be more convenient to solve the problem by running it on a single machine, without distributing it.

A low value, in particular less than the unity, indicates instead that the solution is cost-effective: the time spent to compute solutions of the crunches is higher than the correspondant time "lost" in transferring data between the client and the server.

The unit value of the ratio can be considered as the value of reference of *"threshold" of convenience.*

To have this approach method actually effective in practice, the TER value should be significantly less than unity: for example, a value of 0.5 means for the TER index means that the $\frac{2}{3}$ of the total time is spent in computing the solution, while the remaining $\frac{1}{3}$ is the time lost in data transfers on the network.

4.2.4 Metrics relationships

Looking at the equations that define the metrics in previous sections (see eq. 4.1, eq. 4.2, eq. 4.3) the relationships between the different measures can be easily identified. Each of these can be obtained by knowing the other two:

$$DER = PTR \cdot TER \tag{4.4}$$

$$PTR = \frac{DER}{TER} \tag{4.5}$$

$$TER = \frac{DER}{PTR} \tag{4.6}$$

4.3 Data logging

The measurements of the values used for the calculation of metrics of the system are stored and later retrieved from the server side problem database.

As stated, the server side of the application persists into the database not only the information about the problem but also all other accessory useful data of the problem.

The server-side entry points queried by clients deal to identify the time required to transfer the data, identifying opening and closing connections after sending or receiving, according to the operations of the client.

These data, which the server records autonomously relatively its operations are complemented by time measurements that client side banner componentscollect and send as ancillary information of the results of processing.

The information needed to measure different values are then been identified in the following, independent of the particular problem:

- *GET Start*: records the call entry point, when the service method to send the crunch is entered.
- *GET End*: records the closing of the data stream after sending the package containing the subproblem, immediately before closing the process and exit (see code 4.1 and 4.2).
- *POST Start*: record the call entry point of sending the solution, when you enter the service function of receipt of the results of the crunch.
- *End POST*: records the closing of the stream of data after receiving the package containing the calculated results of the crunch, just before close of the process and exit (code 4.3 and 4.4).
- *GetTime*: This value is the information that is recorded as the value of recovery time from the client, which corresponds to the time difference between the time *GET End* and *GET Start*. The same time was also measured from the perspective of the client, evaluating the value as the difference between the start of the request for a new crunch and the completion receiving data, just before starting the cycle resolution.
- *CalcTime*: This information is sent by the client as as accessory to the calculation results. This time is included among the time difference between *GET End* and *POST Start*, which serves as the upper limit.

  Is preferable to collect this value directly into the client instead use the measurements of time on the server side, since client-side a measure more effective and therefore more significant of the computation can be obtained.

- *PostTime*: this information records the amount of time spent in sending data from the client, similar to what was seen for the *GetTime*.

  This time corresponds to the difference between the times *POST End* and *POST Start*.

  This value is not measurable by the client side as it would be necessary to complete the send operation, close the connection, collect the total time spent and then send a new data packet to server that contains only this information.

Code 4.1 t_{get} java implementation

```
protected void processGetRequest(
                HttpServletRequest request,
                HttpServletResponse response)
                    throws ServletException, IOException {
    LogRecord lr = null;
    lr = new LogRecord(Level.INFO, "Start GET: "+new Date());
    logger.log(lr);
    // ... open response output stream ...
    // ... get crunch data ...
    // ... prepare xml packet ...
    // ... send data ...
    // ... close stream
    lr = new LogRecord(Level.INFO, "End GET: "+new Date());
    logger.log(lr);
}
```

Code 4.2 t_{get} php implementation

```
<?php
header('Content-Type: text/xml');
try{
    logGetStartTime();
    echo(getCrunchDataPacket());
    logGetEndTime();
}
catch(Exception $ex) {
    echo 'Caught exception: ',  $ex->getMessage(), "\n";
}
?>
```

Code 4.3 t_{post} java implementation

```
protected void processPostRequest(
                HttpServletRequest request,
                HttpServletResponse response)
                     throws ServletException, IOException {
    LogRecord lr = null;
    lr = new LogRecord(Level.INFO, "Start POST: "+new Date());
    logger.log(lr);
    // ... open response input stream ...
    // ... read crunch result data ...
    // ... update database ...
    // ... close stream ...
    lr = new LogRecord(Level.INFO,
                       "CalcTime: "+result.getCalctime());
    logger.log(lr);
    lr = new LogRecord(Level.INFO, "End POST: "+new Date());
    logger.log(lr);
}
```

Code 4.4 t_{post} php implementation

```
<?php
try{
    logStartPostTime();
    saveCrunchResult();
    updatePendings();
    logEndPostTime();
}catch (Exception $ex){
    echo 'Caught exception: ',  $ex->getMessage(), "\n";
}
?>
```

Chapter 5

Experimental results

This chapter presents the results obtained during the various tests carried out during the system and solutions development.

Results and details relating to mutual comparison of these solutions are also presented.

Collected data, then use with the metrics defined in earlier chapters, are fundamentals for the subsequent the mathematical model description of the system presented later in treatment.

5.1 Testing environment

The tests were conducted on the implementations of the problems made during the development of the system, detailed in previous chapters (see chapter 2 and chapter 3.

After the first design implementation of the the server-side published on a Java Apache Tomcat server environment for the test was decided to use php to re-implement the problem scheduler server side component.

The client components of the application were made available both without changes due to the independence from the server implementation, and have not required any change moving from the first environment to the new one.

The aspect that deserves attention in this context and need to be underline concerns the network component of the system: the tests are were carried out at the beginning on a 100MB LAN environment, at the Department of Engineering, University of Ferrara.

Then the same system has been published to the external Internet environment and tests continued accessing the distributed system by outside the University network, to collect data in a context more aderent to the real usage.

Obviously when the system is used in an Internet environment, communications are significantly slower and more unreliable than those observed in a LAN.

The main effect that can be observed is the shift of the values of measurements from the surveys carried out when we will move into real Internet environment from an high-performance Intranet one.

The tests can be divided into two categories:

- *Comparison of metrics*: data are collected solving the two problems studied, RSA crack (par. 5.2.1) and genetic correlation (par. 3.1.2), by measuring the values of parameters defined in the previous chapter (see section 4.2.1, 4.2.2, 4.2.2), with the aim of identifying a possible classification of problems.

- *Comparison of implementations*: data were collected related to the solution of a single problem, the genetic correlation (par. 3.1.2) using the two different client implementations developed, with the aim of assess whether a particular language may offer some kind of benefits compared to the another.

5.2 Metrics comparison

The metrics defined in chapter 4 have been applied to the 2 problems implemented, the breaking of RSA encryption and calculating the Pearson correlation.

5.2.1 RSA Cracking

The problem has been set to the resolution of the factorization of a number of 72 bits, 12 bits of crunch, corresponding to 4096 values for each crunch.

From the measurements made on resolving the problem of the RSA cryptosystem crack the following values for the crunch solving have been obtained:

- average time of calculation, calculated by the client: **3499** ms.

- Average time to serve a GET request from measured by the client: **221** ms.

- Average time to serve a GET request from measured by the server: **182** ms.

Measure\Problem	Factorization	Correlation
PTR	0.694 Byte/ms	314.982 B/ms (300KB/s)
TER	0.133	0.041
DER	0.092	12.898

Table 5.1: Metrics: factorization and correlation

- Average time to serve a POST request, measured exclusively from the server: **243** ms.
- size of a GET packet containing crunch data: **125** Byte
- size of a POST packet containing the computed crunch solution: **197** Byte

5.2.2 Pearson's correlation

The system calculates the correlation over a database composed of 20000 samples, packaging 50 + 50 samples a time, calculating the correlation between all possible pairs, considering the correlation result as significant when the absolute value is greater than 0.4.

From the logs for the computation of the correlation over the genetic samples database were extracted the following values:

- average time of calculation, calculated by the client: **19293** ms.
- Average time for serve a GET request measured by the client: **580** ms.
- Average time for serve a GET request measured by the server: **542** ms.
- Average time for serve a POST request, measured exclusively from the server: **210** ms.
- size of a GET packet containing crunch data: **196946** Byte (about 192 KB)
- size of a POST packet containing the computed crunch solution: **51890** Byte (about 51 KB, it depends on the number of significant correlations detected).

The results of the metrics computed using the collected data are presented in Table 5.1.

5.2.3 Results

The PTR index for the first problem, the crack of the RSA encryption, has showed a relatively low transfer rate, considering the LAN environment where the first tests have been led.

In reality, this measure was partially distorted by the fact that the data packet is extremely small and therefore does not use efficiently the network connection.

In this case assumes a significant overhead the path along the network stack both server and client side.

In the second problem, about the Pearson's correlation, the network provides a more significant data rate considering the environment in which it operates.

The TER index is probably the most significant for our system, since it indicates the ratio between the time spent in data transfer and the the corresponding time spent in client side computation and gives the convenience value of the system.

Considering the unit value as a threshold for the convenience index it results that when the value is above the unity value the system is not convenient while if the index has a value below the unity there is convenience in the use of distributed system to solve problem (see section 4.2.3).

The index result for both the problems faced during test was very good, well below the the unity value, indicating that both problems can be conveniently dealt with the designed distributed system.

Regarding the problem of the correlation, where the data transfer is critical respect to the computation component, resolution algorithm has been properly calibrated to get a good result for the TER index convenience value.

The latest index, the DER, indicates the "usage" of data comparing the packet size and computation time (par. 4.2.2): this measure has good values when the index is closer to 0.

In the case of breaking the RSA we have a value very close to 0 as the descriptive parameters of a crunch include very little data.

With regard to the Pearson's correlation problem, instead, the index shows an high value due to the particular characteristics of the problem that is very data oriented.

From the data collected the system performance measured during the test have been more than good, although we must stress the fact that bringing the system on the Internet with wide usage rather than in a controlled testing environment the parameters that control the execution speed and size of packets should be modified to maintain the same performance level.

5.3 Implementation comparison

The second part of the tests conducted involved the comparison of the 2 different solutions implemented for the client side: the realization with AJAX and the one with Flex.

In this case, the tests are based on the problem more stressful for the system, that of the Pearson's correlation, to better focus on performance characteristics of the different implementations.

The tests were performed both in the same Intranet environment of the Previous comparison of metrics (section 5.2 and on a more realistic Internet environment, accessing from outside the ENDIF network to the published application, in accordance with the following features:

- The database contains $S = 20000$ samples, each consisting of 100 measurements made on genetic sequences.
- Each crunch is composed of a $50x50$ set of samples on which perform calculations of correlation, taken from the collection SxS, organized as can be seen in fig. 5.1.
- The algorithm considers the calculation of a positive result when correlation in absolute value exceeds 0.4.
- data were collected over 12700 crunches, which correspond to the calculation of $50 * 50 * 12700 = 31750000$ Pearson's correlations.
- The collected results showed just over 60000 significant correlations, corresponding approximately to 2‰of the total calculated.

In this case, in addition to the previously defined data information on the type of client used have also been collected, to permit analysis for the 2 implementations in a separate way.

Data relative to lost crunches,for errors of various kinds, have not been considered in this case.

Since the testing environment was quite controlled both in the Intranet and the Internet tests the number of lost crunches can be considered negligible for treatment.

The table 5.2 presents the most important data detected by analyzing the information collected during the calculation of Pearson's correlation.

The table shows statistics on the execution with AJAX and Flex clients and also gives the consolidated value, regardless of the client used.

It should be emphasized that the results obtained with two different clients are quite similar and therefore close enough to general data that do not distinguish the client used.

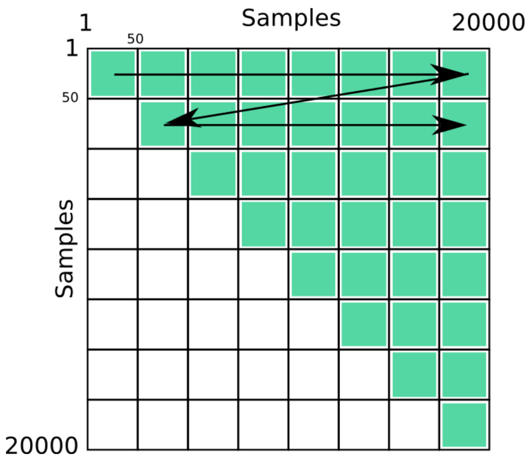


Figure 5.1: Crunch selection from the database

Measure	Ajax	Flex	Total
#sample	5718	6989	12707
Average get time [ms]	1301	2334	1766
Median [ms]	754	1702	834
Average computation time [ms]	8591	7052	7821
Median [ms]	8008	4671	5288
Average post time [ms]	1655	725	1237
Median [ms]	1281	509	1231

Table 5.2: Average time detected

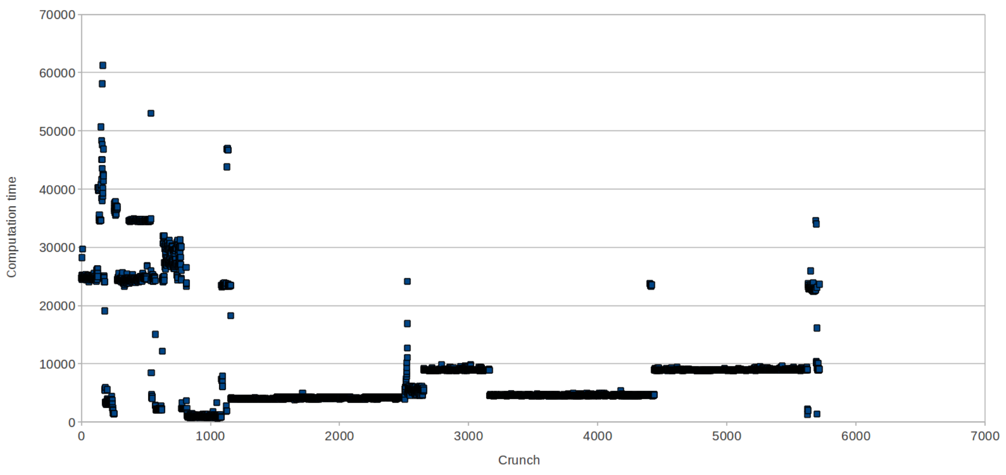


Figure 5.2: Distribution of computation time for the AJAX client data

Another interesting fact to be noted is the distribution of computing time, visible in figs. 5.2 for the AJAX client and 5.3 for the Flex client: there is a fairly even distribution of time, distributed according to number of "steps".

This is due, in part, to the environment in which the tests were conducted, but more prominently in the construction of the crunch, sized to keep the calculation within a certain range of times, in order to minimize the probability of crunch loss as a result of the disconnection by the client.

The segmentation of time, finally, is due to the user control of computing power provided:

- the lower layers, corresponding to a faster computation, are obtained by increasing the number of inner loops performed before a break and/or decreasing the pause time between two successive cycles.

- the intermediate layers are those that correspond to standard, which takes into account an average time per visit that minimize loss of crunch, and at the same time limiting as much as possible the resources involved in the calculation.

- the upper layers, which correspond to the user action of slowing computation and thus reducing the power provided or to moments of congestion of the client, leading to a lower response in the browser and then to a general slowdown int the computation process.

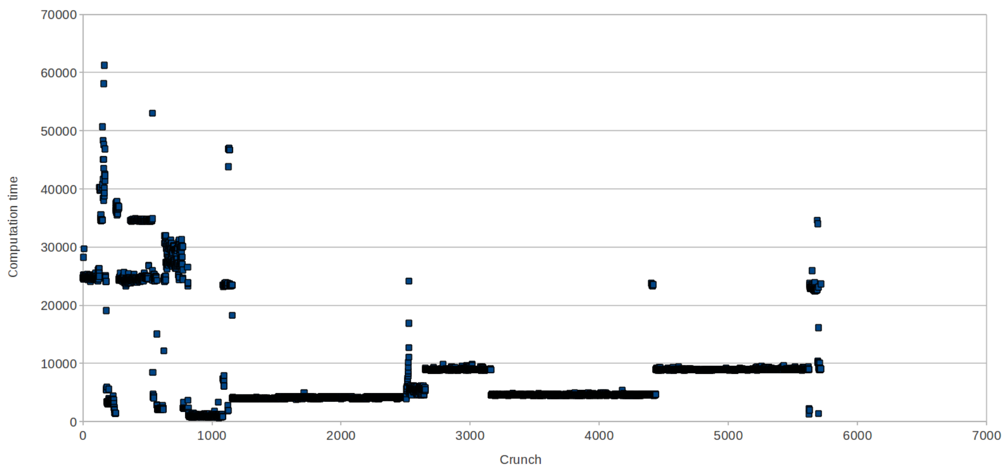


Figure 5.3: Distribution of computation time for the Flex client data

Index	Ajax	Flex	Total
TER	0.686	0.692	0.688
PTR [KB/s]	0.027	0.033	0.030
DER [KB/s]	0.018	0.023	0.020

Table 5.3: Metrics calculated by type of client

5.4 Results

Considering the characteristics of the two implementations realized the expectation was that the results obtained by the Flex implementation should be significantly better than those obtained with the AJAX implementation of the client.

This is mainly because the Flex client is natively compiled and runs in a separate process, although under the control of the browser that hosts the plugin.

In fact the results confirmed this expectation, the test showed a difference in the mean time for the 2 types of clients by about 20% between results of the AJAX client an the results of the Flex implementation.

Considering the overall average the difference is instead of ±10%, due to the fact that totals are in practice the average of the 2 solutions.

The difference, however, is not so marked as to say that the Flex client is much better than the AJAX version for the distributed system designed.

Here at last, the table 5.3 complete the comparison between the two different implementation of the client showing the index values for the metrics defined in chapter 4, evaluated in two separate cases involving the single type of client and in the comprehensive way.

Chapter 6

Mathematical model

The ultimate objective of development of the web distributed computing web studied and developed as described so far is produce a mathematical model that is able to describe the system, in order to determine the limits, constraints and optimal size for convenient use of the system.

In other words, the model must determine the optimal size for the crunch sent to clients and minimize the probability of crunch loss identifying appropriate client-side parameters for the re-engineered algorithm execution.

Typically, the data packet size is strongly dependent on the problem addressed, as already noted in chapter 4.

It is therefore rather difficult to develop a model that can be applied to problems different with each other in terms of description of the problem itself, since this description determines then the crunch data packet sent to the various client for computation.

Trying to overcome this obstacle the problem itself has been used as one of the inputs to the model developed: the problem description is thus one of the input parameters and is used to derive results, so it is possible to control and then modify these results according to the problem description given as input.

To characterize the problem and introduce it as a parameter of the model developed the number of elementary operations required to solve a crunch have been used.

This allows to generalize the model unlinking it from a particular class of problems because it is usually always possible to determine the number of operations required to solve an algorithm, in our case that for the resolution of the individual crunch on the client side.

Another advantage of this generalization is the possibility to apply the same model to the system, with the same parameters and measurements, without requiring any changes to introduce and describe new kind of problems.

6.1 Model parameters

This section starts with the introduction of the main parameters of the model that describes the web distributed computing system exposed, strictly defined:

- t_{op}: operation time. This parameter is one of the most difficult to measure and usually must be identified experimentally by running the code on different machines with different performance.

  Based on the rated speed of the CPU is possible to approximate a value close enough to the real value.

  Another aspect to take into account is the fact that transactions are often complex in terms of implementation, as for example, function calls or composed operations.

  In this case, complex operations count for the number of elementary operations of which they are composed.

- t_p: pause time. It is the time between the execution of two consecutive inner loops.

  This time must be long enough to allow the release of CPU control and therefore not cause delays for the user.

  On the other hand should be short enough to ensure that there is not too much idle time between two inner, as this would cause a loss of time useful for calculation.

- N_{op}: number of operations required to solve a crunch. This is the total number of elementary operations (in reference to the first point) needed to solve a subproblem completely.

- n_{op}: number of operations required to solve an inner cycle. This number represents the total of elementary operations performed between 2 successive pauses of the computation, without interruption.

- k_A: number of cycles for the resolution of an inner loop, executed monolitically before releasing control to the browser.

- k_B: number of cycles required for a complete crunch solution. In practice this is the the number of external loops performed to solve a subproblem.

- t_n: time between two successive breaks, or in other words the computation time needed to execute k_A inner loops.
- t_c: computation time. This time, already defined previously, is the total time needed to completly solve a crunch.
- T_v: time to visit. This value is the average time that a user spends visiting a single web page. This value can vary greatly depending on the type of information the page provides.

  An average time of visit can be easily statistically obtained from the visits.

  Typically the value is settles between 20 and 30 seconds. Another significant value, the median, could also be used, derived also from visit statistics.
- v_{net}: network speed. Means in this case rated speed of the network. For a LAN this value is usually 100Mbit/s or even 1Gbit/s, depending on the infrastructure available.

  In an Internet environment this value can vary greatly, according to the infrastructure and to the service provided.

  Moreover, while in a local area network the nominal speed can be quite significant, in an Internet environment often the service provider has no guaranteed minimum bandwidth, then the rated speed is purely indicative.

  Under these conditions the rated speed must be determined with analysis of the average bandwidth available, then considering how actual speed will vary around the nominal set.

6.2 Model description

After defining the parameters needed to define the model describing the web distributed computing system presented, now follow the details of the construction of the model step by step presented.

The first constraint of the model concerns the time to visit T_v: to have some assurance, or in a statistical sense high probability, that the computation completes in the time the user spend in a given page that hosts the computing client, the total time needed to complete the resolution of a crunch has to be kept below this value.

If the process of solving a subproblem, including retrieving data from the server, the calculation and the delivery back of result of processing, takes longer than the visit time, the crunch will be lost due to interruption of resolution process.

The loss of crunches obviously worsens the system performance because if requires the computation of a greater number of crunch due to the recovery of those that are lost.

In addition, the time used for the resolution of a crunch which solution is not collected on the server side is being lost in the calculating the total time required to solve a particular problem.

The constraint appears to be the following:

$$t_{get} + t_c + t_{post} < T_v \tag{6.1}$$

The expression 6.1 is the minimum and also the main requirement to ensure that at least one crunch is resolved completely, so received from the server, processed and sent to the server for the solution storage.

Obviously if the user keeps the client connection opened for a sufficiently long time than the time needed to solve a crunch (including time of receipt and dispatch) it is possible to solve more than a crunch since the resolution process is iterative and continuously rescheduler, improving overall performance.

The computation time t_c is, as mentioned above, strongly dependent from the algorithm, this time can be defined as the product of number of operations multiplied by the time of execution time of a single operation [1]

$$t_c = N_{op} * t_{op} \tag{6.2}$$

The equation 6.2 does not consider, however, the re-engineerization of algorithms of the web distributed system and in particular does not show the pauses between two subsequent inner cycles necessary to release control to the browser and keep it usable and not slow.

Considering the revised components of the resolution algorithm the following expression is obtained for the computation time t_c:

$$t_c = (n_{op} * t_{op} * k_A + t_p) * k_B \tag{6.3}$$

A very important aspect to be highlighted at this point is that the running time of an inner loop must be maintained sufficiently low, this allows the client machine avoid congestion and overloading of the CPU, which would translate in a slow response to user input, particularly relatively to the

[1]remains to be stressed that this time is dependent on the performance of the host machine and the load of machine during execution

browser, which is supposed to be used as foreground application during its processing.

This defines a new constraint for the model: the time execution of an inner loop must be maintained below a specific time t_A.

This is the maximum fixed time of continuous computation after then the the computation process must pause and release the control.

To keep the browser, and the machine in general, always interactive to user input this time must be maintained in the order of magnitude of 10^{-1}.

The constraint for the inner loop is therefore:

$$n_{op} * t_{op} * k_A < t_A \tag{6.4}$$

By relaxing the constraint in an equation:

$$n_{op} * t_{op} * k_A = t_A \tag{6.5}$$

it is possible to find the value of k_A, ie, the number of inner cycles that can be completed maintaining the computation time under the limit imposed by t_A:

$$k_A = \frac{t_A}{n_{op} * t_{op} *} \tag{6.6}$$

From which follows, since the inner loop is atomic and therefore not divisible:

$$k_A = \left\lfloor \frac{t_A}{n_{op} * t_{op} *} \right\rfloor \tag{6.7}$$

The value of k_A is the greatest integer that keeps the single computation atom within the time limit t_A

From the concept of convenience of the system, already introduced during system development and related metrics, an additional constraint of the model is defined as follows:

$$TER < 1 \tag{6.8}$$

Considering that the TER index formula can be solved in the following expression:

$$TER = \frac{t_{get} + t_{post}}{t_c} < 1 \tag{6.9}$$

which is immediately derived:

$$t_{get} + t_{post} < t_c \tag{6.10}$$

At this point the computation time can be explained in its components, as seen above in equation 6.3), obtaining:

$$t_{get} + t_{post} = t_t < (n_{op} * t_{op} * k_A + t_p) * k_B = t_c \tag{6.11}$$

In the equation 6.12 was introduced the total time as the sum of transfer times, both get and post, for 2 fundamental reasons:

- the distributed system is convenient when the *total* transfer time is less than the computing time;

- typically the size of the response is a function of the size of the request.

  The response will include the solutions of a problem specified in the packet sent from the server after the request.

  Intuitively, except in special cases, larger will be the get packet for a crunch, the greater the size of response packet will be too, since it is relative to a greater size subproblem.

By combining the constraint equations 6.1 and 6.10 yields the following:

$$t_{get} + t_c + t_{post} = t_t + t_c < T_v \tag{6.12}$$

Once defined the time constraints, it remain to determine those relative to sizing, the other key aspect of the model, aimed at establish limits and optimizations about the size of packages containing the crunch to be processed.

therefore the size of data packets of subproblems is introduced as function of the timing values that have been introduced so far:

$$\begin{aligned} D &= D_{in} + D_{out} \\ &= t_{get} * v_{net} + t_{post} * v_{net} \\ &= (t_{get} + t_{post}) * v_{net} = t_t * v_{net} \end{aligned}$$

From which it follows, by combining the equations 6.10 and 4.3 with the appropriate substitutions:

$$D = t_t * v_{net} < t_c * v_{net} \tag{6.13}$$

Finally, still following, from the equation 6.13 the fundamental constraint for the model developed so far on the data packets size.

$$D < ((n_{op} * t_{op} * k_A + t_p) * k_B) * v_{net} \tag{6.14}$$

This last inequality 6.14 is the final result of the mathematical model describing the web distributed system and provides the limit on the size of data packets transferred.

Transferring a larger amount of data in a crunch makes the system not convenient in its use compared to the direct resolution, considering the amount of data transferred in both input and output from the client.

Keeping the total size of data transferred under this threshold makes the web distributed computing system developed during the PhD program convenient for the resolution.

In other words we can say that it is possible to create a scheduler to distribute to crunches to clients sized in a way that makes the problem provided as input solved conveniently.

This constraint of the model has another important meaning as well as to establish a limit of convenience given the algorithm resolution as input.

For the majority of the problems the size of k_A and k_B defines the amount of data analyzed for each crunch. This is strongly linked to the size of data packets.

If the constraints of the problem are all satisfied except the last 6.14 there are two possibilities:

- the resolution algorithm is not optimized in the context in which it is used, it is therefore necessary to review it and improve the format of packages used or algorithms, to meet all the constraints presented.

- The problem can not be solved conveniently with the system.

The second option is actually a rather rare case as it is generally always possible to re-engineer conveniently the algorithm for the system on the client side and/or the format of packets so that data transmitted satisfies all the constraints of the model.

In conclusion, the optimal size D_{opt} can be defined as the size of data packets that provides minimum value for the TER of the problem.

The lower the value of the TER ratio index is, the better the performance of the scheduler on the server side of the problem are: for a limited transfer time there is a correspondant greater time invested by the various clients to solve the problem.

6.3 Examples

To conclude the exposition of the model of the web distributed system here presented now follows the application of the equations and constraints of the model to a specific problem, showing how to derive the different sizing parameters.

The application of the model will use as a case example the problem of RSA encryption algorithm crack, presented in the previous chapter 3 and then in paragraph 5.2.1.

It will be shown how the application of the model to an actual concrete case takes to the identification of parameters necessary to a convenient implementation of the resolution through this web distributed computing system.

Starting with the definition of the average visiting time as $T_v = 20s$, the first constraint the model takes the form:

$$t_{get} + t_c + t_{post} < 20 \tag{6.15}$$

This value is indicative and may be derived from the average time the user spend on a single web page, calculated on a statistically sample of significant web resources.

Here are some values, established for convenience and used during the application development, that can be applied to evaluate the model constraints:

- $t_c = 15s$: it is a reasonable computation time referenced to equation 6.15.

  Considering furthermore the value of convenience for TER index it is obtained immediately that the system has sufficient performance when $t_c > 10s$.

  The chosen value is then consistent with the convenience of the system and with acceptable performance.

- $t_p = 2 * 10^{-3}s$: the pause time is 2 ms, sufficiently short to avoid running too long without processing problem data but at the same time sufficient to allow the release of control.

- k_A: the number of inner loops performed before releasing control.

  The number is limited to efficiently alternate computation and pauses with release of control.

- k_B: number of external cycles needed to solve a whole crunch, given the size of the range to process of 2^{12} integers.

- $t_A = 3.65 * 10^{-3}$: average time observed during tests for the resolution of a set of inner loops.

- $v_{net} = 2MBit/s$: network speed considering the execution in an Internet environment. This value has to be considered an average variable value affected by all the known network fluctuations.

Considering the constraint on the crunch size from equation 6.13 and paragraph 6.2 the crunch dimension equation gives:

$$D = t_t * 2 < 15 * 2 = 3.75MB < 20 * 2 = 5MB \tag{6.16}$$

As one can easily guess the result is not completely realistic, since it is rare to reach the rated speed of the network in a real environment on the Internet.

This relation, however, indicates that the client must make a strong processing on the data received, or in other words, it must add value to information processed so that the system could be used in a convenient way.

The example on this particular problem wants to underline a very important fact that characterizes the convenience of the approach presented: the properties of the problem of RSA cracking and its relative algorithm that has been implemented have a data packet of constant size.

The package consists of the RSA public key system (the product of two prime factors) and the extremes of the range try factors.

This information is totally independent of amplitude interval taken from a single crunch.

Consequently, it is possible to change the computation time of a crunch by using more or less wide intervals while maintaining the size package constant.

This same concept holds true for the package containing the calculation results.

Considering all these features, for this particular problem the optimum is determined simply by setting the size of a data packet as D^{fix} and its correspondant time to transfer as t_t^{fix} that only depends on the speed of the network and not on other characteristics of the problem.

It is obtained from equations 6.9 and 6.13 of the previous paragraph:

$$\frac{t_t^{fix}}{t_c} < 1 \tag{6.17}$$

$$D^{fix} < t_c * v_{net} < 20 * v_{net} \tag{6.18}$$

The optimal value is the one that makes the TER index as close as possible to 0; having t_t (and D) values constant the optimum is given by:

$$t_c = T_v - t_t \tag{6.19}$$

This equation is not a formula derived directly from the model, but derived from the considerations on the problem and the resolution algorithm which the model is applied to, stressing the fact that a one of the inputs of the model is exactly the problem.

In general, the optimal value is derived from reflections on these characteristical elements and is part of the optimization process for the system.

This process has to identify the values of k_A and k_B that minimize the value of the TER index of convenience.

Chapter 7

Conclusions

7.1 Summary of the research work

The research work developed and presented here, had as its objective the implementation of an innovative solution for distributed computing system by the use of a web infrastructure.

The results obtained in the development of the various components have shown that this system is feasible, obtaining also good performance with a simple but effective architecture, which makes convenient to use this destributed environment to solve problems.

The use of proven technologies, those enumerated in the universe Web 2.0, AJAX and Flex in particular, the cornerstones of the implemented system, together with the use of standards and specifications for protocols, makes the system usable by almost each of the multitude of clients connected to the World Wide Web.

This fact makes the number of possible clients for the system extremely high, thereby increasing the performance and potential of the system.

The mathematical model developed, moreover, is a very useful tool in the hands of the developers to determine a proper sizing of the various parameters in order to obtain applications with good performance and "convenient" under the definition here described (see 4.2.3).

Another important result is the overall system design: the model can be applied to almost any problem because the problem itself is an input.

It is sufficient to provide some parameters of the problem under examination, as the number and type of operations required to solve a crunch in order to apply the model even to problems very different between each other.

Finally, tests have shown the effectiveness of the system in a real Internet environment, which of course is the natural field for a web distributed computing system, the one that provides the greatest number of potential clients and therefore the most computational power.

The characteristics of the client components, both the one developed in with AJAX the other implement on Adobe Flah/Flex, makes them embeddable in many different web resources, thus differentiating the catchment area.

The current implementation is still in operation at the ENDIF department, at the University of Ferrara [30].

At the conclusion of the the PhD studies can then be summarized positively the achieved results and also evaluate interesting and commercial real applications, especially using the extent of the catchment area for capillary applications.

Finally, the work led to the publication of two different works, the first, introductory, at the *VTC conference* in *2007* [28], the second summarizing the whole system presented, with model and results, in the journal *International Journal of Adaptive, Resilient and Autonomic Systems* [31].

7.2 Ulteriori sviluppi

Among the possibilities for further development of the system, in order to improve the performance and applicability, there is surely the integration of model definitions directly into the server side scheduling process.

This implementation will make the adaptive system with respect to client behavior, in particular with a view of use it in other network environments, where the measured parameters were difficult to measure because of the differences among client adopted and also of the network infrastructure.

By evaluating the constraints of the model at runtime it is possible di dynamically size the crunch of the problem, getting from time to time packages that maximize performance and indexes of convenience, while limiting the crunch loss and errors.

Client components are also likely to improve, although it is pointed out that these depend on the particular problem, then the optimizations can only partially cover these components, mostly in the structural parts.

Finally, an aspect still under analysis, given the sensitivity and the possible impacts of the topic, is related to security, regarding the server and client components separately.

On the server side must have the assurance that the results received from clients are not corrupted by errors or tampering. A first step has already been implemented by using a control system over the responses on the server side, but still does not eliminate entirely the possibility of malicious actions by some clients.

On the client side the problem moves to the code executed, which in real application is potentially provided by third parties and may contain malicious instructions, invalidating the effectiveness of the system.

Bibliography

[1] Adam Barth, Collin Jackson, Reis Google, and Chrome Team. The security architecture of the chromium browser. *Web source*, 2008.

[2] Dieter Gollmann. Computer security. *Wiley Online Library*, 1999.

[3] Jesse James Garrett. Ajax: A new approach to web applications. *experiencezen.com*, 2005.

[4] Noriko Hanakawa and Nao Ikemiya. A web browser for ajax approach with asynchronous communication model. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, 2006.

[5] Vic Cekvinich. Rich internet applications. *TheServerSide.com*, 2004.

[6] Tom Noda and Shawn Helwig. Rich internet applications. *UW E-Business Consortium - Best Practices Report*, 2005.

[7] Jeremy Allaire. Macromedia flash mx—a next-generation rich client. *Macromedia.com whitepapers*, 2002.

[8] B.K. Schmidt, M.S. Lam, and J.D. Northcutt. The interactive performance of slim: a stateless, thin-client architecture. *Proceedings of the 17th ACM Symposium on Operating Systems and Principles. Kiawah Island, SC*, 1999.

[9] Dave Crane, Eric Pascarello, and Darren James. *Ajax in Action*. Manning, 2006.

[10] Anthony Franco. Flex vs. ajax friends or foes. *EffectiveUI White Paper*, 2008.

[11] Tim O'Reilly. What is web 2.0. *O'Reilly network*, 2005.

[12] Wikipedia. Web 2.0. *Wikipedia*, 2006.

[13] Gregory R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison–Wesley, 2000.

[14] Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach.* Cambridge, 2009.

[15] Sunderam V. Pvm: a framework for parallel distributed computing. *Concurrency: practice and experience, 2(4), pp215-319*, 1990.

[16] Wikipedia. List of distributed computing projects. *Wikipedia*, 2011.

[17] J. Maassen and H. E. Bal. Solving the connectivity problems in grid computing. *HPDC07, 2529*, 2007.

[18] L. Marchal, P. V.-B. Primet, Y. Robert, and J. Zeng. Optimal bandwidth sharing in grid en- vironments. *2006 15th IEEE International Conference on High Performance Distributed Computing (pp. 144-155)*, 2006.

[19] A. Ganguly, A. Agrawal, and R. Boykin, P. O.and Figueiredo. Wow: Self-organizing wide area overlay networks of virtual workstations. *2006 15th IEEE International Conference on High Performance Distributed Computing (pp. 30-42)*, 2006.

[20] Stanford University. Folding@home. *http://folding.stanford.edu/*.

[21] Berkeley University. Seti@home. *http://setiathome.berkeley.edu/*.

[22] CERN laboratories. Lhc@home. *http://lhcathome.cern.ch/*.

[23] Wikipedia. Autonomic system (computing). *Wikipedia*, 2011.

[24] Wikipedia (IBM). Macromedia flash mx—a next-generation rich client. *Wikipedia*, 2011.

[25] Menezes A., Van Oorschot P. C., and Vanstone S. A. *Handbook of Applied Cryptography.* CRC Press., 1996.

[26] Cormen T. H., Leiserson C. E., Rivest R. L., and Stein C. *Introduction to algorithms (2nd ed.).* MIT Press and McGraw-Hill, 2001.

[27] HyperStat Online. Hyperstat online contents: Pearson's correlation. *http://davidmlane.com/hyperstat/A34739.html*, 2006.

[28] Boldrin F., Taddia C., and Mazzini G. Distributed computing through web browser. *WTC Conference, Baltimore, USA*, 2007.

[29] Huizingh E.K.R.E. The content and design of web sites: an empirical study. *Information & Management, Volume 37, Number 3, 1 April 2000, pp. 123-134(12)*, 2000.

[30] Fabio Boldrin. Testing environment at endif dpt. - unifersity of ferrara. *http://www.tlc.unife.it/mycomputation*, 2009.

[31] Boldrin F., Taddia C., and Mazzini G. Web distributed computing systems, implementation and modeling. *International Journal of Adaptive, Resilient and Autonomic Systems*, 2010.