A Java Toolkit for Distributed Evaluation of Hypergeometric Series

by

Fawad Chughtai

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2004

Abstract

Hypergoemetric Series are very important in mathematics and come up regularly when dealing with the precise definitions of constants such as e, π and Apery's constant $\varsigma(3)$.The evaluation of such series to high precision is traditionally done with multiple divisions, multiplications and factorials, which all takes a long time to compute, especially when the computation is done on a single machine.

The interest lies in performing this computation in parallel and in a distributed fashion. In this thesis, we present a simple distributed toolkit for doing such computations by splitting the problem into smaller sub-problems, solving these sub-problems in parallel on distributed machines and then combining the result at the end. Our toolkit takes care of all the networking for the user; connectivity, dropped connections, management of the Clients and the Server. All the user has to provide is the definition of the problem; how to split the problem into sub-problems, how to evaluate the sub-problems and finally how to combine the sub-problems and produce a result. The toolkit records timings for computation as well as for communication.

What is different about our application is that all the code is written in Java (which is completely machine independent) and all the Clients are Java Applets. This means that having a web browser in enough to take part in the computation when it is distributed over the internet. We are almost guaranteed that every computer on the internet has a web browser. The Java Plug-in (if unavailable) can easily be downloaded from Sun's web site.

We present a comparison between Java's native BigInteger library and an FFT based Integer Library written by R. Howell of University of Kansas. This study is important since we are doing computations with very large integers.

To test our system, we evaluate e to different number of digits of precision and show that our system truly works and is easy for anyone to use.

Acknowledgements

I wish to thank my supervisor, Eugene Zima, for his support for the duration of this work. Without his friendship, suggestions, financial support, motivation and absolute patience, it would not have been possible to complete this work. He kept the discussions interesting and fun and always motivated me in the time of need. He has helped me a great deal throughout my undergraduate and graduate career as well as in the preparation of this thesis.

I also wish to thank Douglas Stinson and Arne Storjohann for serving on the thesis committee. I would also like to thank all the members of the Symbolic Computation Group.

I would like to thank all my family members for their love, understanding, patience and support at all times. I would specially like to thank my parents for encouraging me to pursue my Masters in Computer Science.

I want to acknowledge all the help, support and guidance provided to me by Peter Van Beek, Charlie Clarke, and Justin Wan. I also would like to thank my colleagues and friends Abid Malik, Omar Nafees and Rosina Kharal for their continued support and guidance throughout my Graduate career.

Finally, my special thanks to my dear Friend and wife, Sadaf Jamal, for all her love, support, understanding, and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Background Information

1.1 Hypergeometric Series

We consider the evaluation of the hypergeometric series

$$S(N) = \sum_{n=0}^{N-1} \frac{a(n)}{b(n)} \prod_{i=0}^{n} \frac{p(i)}{q(i)} \tag{1.1}$$

to high precision, where a, b, p, and q are polynomials with integer coefficients, and $a(n)$, $b(n)$, $p(n)$, and $q(n)$ have bit length *O(log n).* We also assume that the series is linearly convergent, so that the nth term of (1.1) is $O(c^{-n})$ with$c > 1$. These series are commonly used in high precision evaluation of elementary functions and other constants, including the exponential function, logarithms, trigonometric functions, and constants such as Apery's constant $\varsigma(3)$[6, 7].

A widely used approach to the computation of (1.1) is binary splitting, which computes the numerator and denominator of the rational number $S(N)$. The decimal representation of $S(N)$ is computed by the floating point division of the numerator by the denominator.

In order to understand the properties of this type of series, let us take the computation of e. The Taylor Series expansion of e dictates that

$$e = \sum_{i=0}^{n} 1/i!$$

Alternatively, we can view this formula as

$$\frac{P}{Q} = \sum_{i=0}^{n} 1/i!$$

where performing the final division of P by Q would give us the exact value of e.

If we need n digits of precision, it is sufficient to evaluate m terms in (1.1) where $n = m$*$log\ m$. In order to evaluate this in a distributed fashion, let $m = p * s$ where p is the number of processors, sub-tasks or sub-problems, and s is the number of terms in a sub-task. In the following formula, k is the number of the sub-task (k = *0, 1,. . . , p-1*).

$$\begin{aligned} e &= 1 + \sum_{i=1}^{m} 1/i! \\ &= 1 + \sum_{i=1}^{m} \frac{1}{\prod_{j=1}^{i} j} \\ &= 1 + \sum_{k=0}^{p-1} (\frac{1}{(ks)!} \cdot \sum_{i=1}^{s} \frac{1}{\prod_{j=1}^{i} (k \cdot s + j)} (2) \end{aligned}$$

With this simplification, we have obtained a formula similar to the one in (1.1) and we have no factorials to perform, since the processor number takes into account the placement of the denominator and multiple factorials are not calculated more than once.

In this way, we can compute $\sum_{i=1}^{s} \frac{1}{\prod_{j=1}^{i} (k \cdot s + j)}$ as a sub-problem by distribution and collecting the results to combine them as in (2).Each sub-problem would have the result in the form of two rational integers P and Q. These are not divided until all P's and Q's from each processor are received.

In essence, the Clients perform the following computation

```
P = 1;
Q = (k - 1) * m + 1;
x = Q;
For i = 2 to m do
    x = x + 1;
    P = P * x + 1;
    Q = Q * x;
End
```

To combine the results, we apply the following code

```
Assuming partial sums are in P[i] and Q[i]
P=1;
Q=1;
For i =1 to k do
    P = P*Q[i] + P[i];
    Q = Q*Q[i];
End
```

Finally performing the division P/Q would yield the correct answer.

1.2 Distributed Computing

Distributed computing is a science which solves large problems by giving smaller subproblems to many computers to solve and then combines the results into a final solution for the original problem. With the advent of the internet, this form of computing has really flourished. Millions of users worldwide have access to the internet and can become a part of the computation. The types of computations have ranged from looking for extra terrestrial activity (SETI) to breaking secret key cryptography systems (distributed.net) to finding million digit prime numbers (Mersenne Prime Search). The most popular form of distributed computing that does not involve any direct mathematical computation is file sharing. Applications such as Napster, Kazaa and Systems such as the Torrent network have made full use of distributing computing for sharing files across networks.

On the computation side, the problems themselves are so large that it would be impossible for one single (even very powerful) machine to solve the problem by itself. With the growing power of the internet and all the computers working together, you have a force that is stronger than any single computer in the world.

We would like to start by describing the different models that can be applied to distributed computing and discussing what benefits each system presents. This will be followed by observing closely some of the popular distributed computing projects and how the researchers have been able to accomplish their goals. A detailed list of current distributed computing projects can be found at http://www.mersenne.org/projects.htm.

We also look at the usage of Java, Applets and its security concerns to discuss how viable Java is as a solution to implementing our System. We present a comparison between two Java Integer libraries and one C++ Integer library and see what type of solution best suits our needs for a fast and efficient Integer Library.

This is followed by a detailed description of the System that we have developed and are proposing for distribute computation of Hypergeometric Series. In this section we detail the architecture of the System, specify how the Server and Client interact with the user and with each other. We also show how a developer can use our System and make minor changes to evaluate a new problem. We then show some tests and results of those tests to show that our System truly delivers what it promises. Finally we have some concluding remarks, followed by some suggestions on some future work to enhance the System.

Chapter 2

Models of distributed computing

There are 2 types of distributed computing models [15]

1. Shared memory

2. Distributed memory

2.1 Shared memory

The shared memory architecture consists of a sharing a single memory/address space among multiple processors. There are two approaches for implementing such a model.

2.1.1 Uniform Memory Address

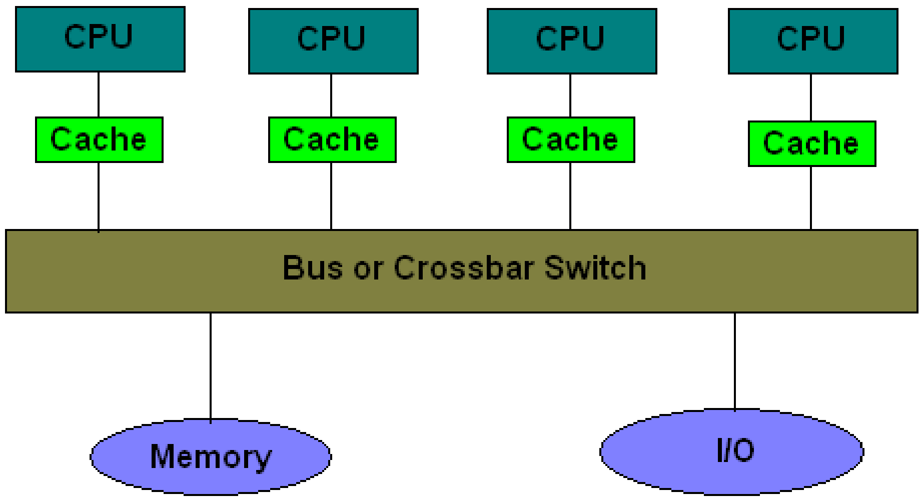


Figure 2.1: Uniform Memory Addressing

This model uses shared system resources such as Memory and I/O. These resources can be accessed by each processor without any added changes to programs. For example, a variable declared in the program can be accessed by each processor without any special changes.

Memory

The memory is centrally located and is shared by all the processing elements.

Cache

The coherence of the cache is maintained by the hardware.

Expandability

It is not simple to add more processors to the system and it is expensive to build in the first place.

Examples

An example of a System using the Uniform memory address model is the Compaq GS AlphaServer.

2.1.2 Non Uniform Memory Address

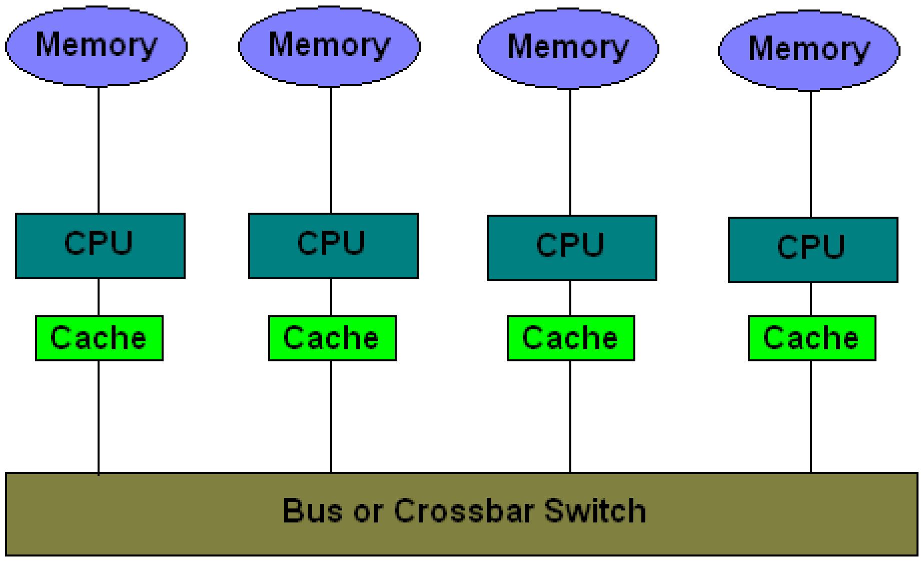


Figure 2.2: Non Uniform Memory Addressing

This model uses a shared address space, but there is the concept of local and remote memory access, which is different for different processors and therefore affects memory latency.

Memory

The address space is shared but memory latency varies with local or remote address access.

Cache

The cache coherence is maintained through a software or hardware protocol.

Expandability

More processors can be added to the System since they have local memory, but the cost of doing so is not economical.

Examples

Some examples of Non-Uniform Memory Addressing machines are SGI Origin 2000/3000 and Sun Ultra HPC Servers.

2.2 Distributed Memory

The distributed memory architecture is very general and therefore can be discussed as single machine architecture or multiple machine architecture.

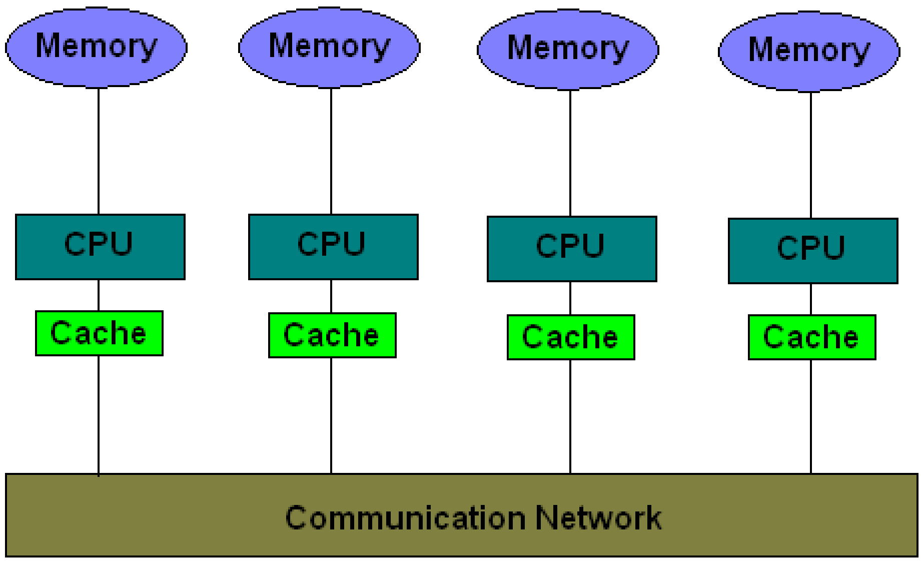


Figure 2.3: Distributed Memory

Although the diagram looks very similar to the Non Uniform Memory Address, this architecture can be either implemented within a single system or across a network or systems. The communication network can either be inside a large server or can be the Ethernet connection between networks of machines.

Memory

The memory manager uses local addressing to access local memory.

2.1.2.6 Cache

There is no issue of cache coherence in this System because the cache is maintained by each CPU locally.

Expandability

This System can easily be expanded by adding and making use of more processors

Examples

IBM SP is a System that has been developed by IBM, which uses distributed memory.

2.2.1 Multiple Machine Distributed Architecture

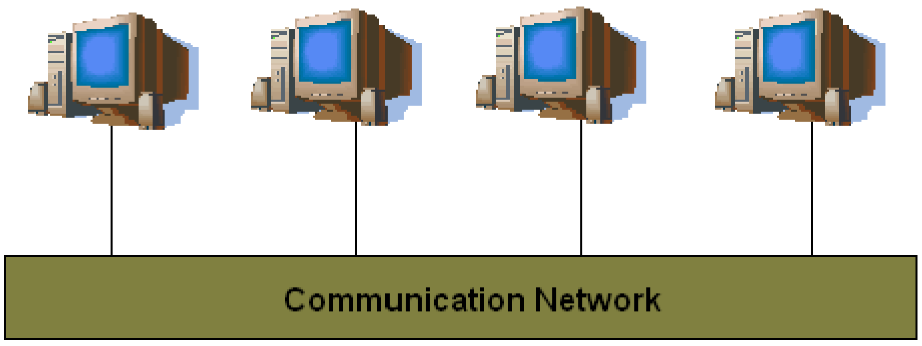


Figure 2.4: Multiple Machines, Distributed Architecture

From the previous discussion we saw that the Ethernet connection between multiple machines can serve as the communication network. The computers connected to the network can be the elements in the system. If we ignore communication costs, then from a distributed computing perspective, if it takes time t to solve a problem on a single machine, then it should take time t/n for the problem to be solved in a distributed fashion, if there are n machines of (roughly) the same computational power.

Memory

In this model, each system has its own local memory and manages it as well.

Cache

Each system deals with cache coherence on its own since the memory and the memory management is done locally.

Expandability

In this model, more systems can be easily added to the network. More systems mean more resources to be utilized.

Examples

Examples of such models are Local Area Networks (LANs).

2.2.2 Multiple Networked Machines Distributed Architecture

What we are really trying to utilize, is the following

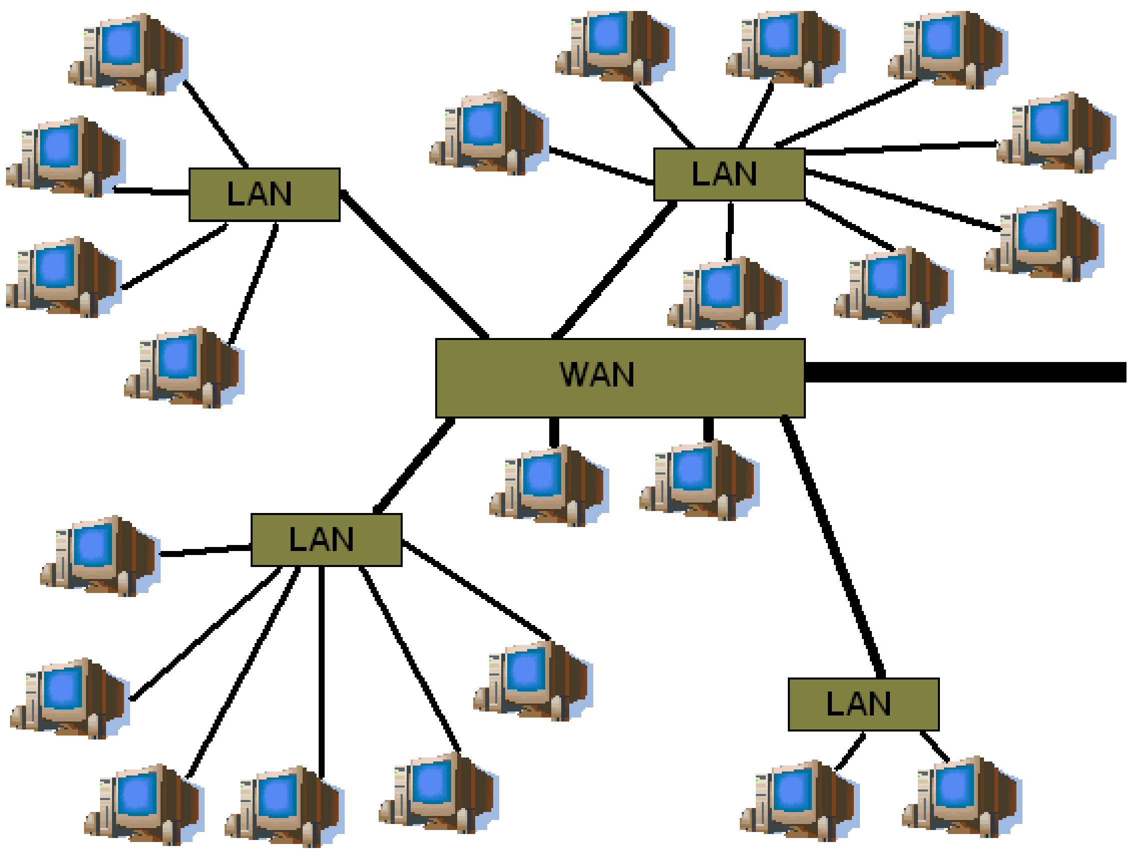


Figure 2.5: Multiple Networked Machines Distributed Architecture

The internet is sometimes called the WAN (Wide Area Network) but the WAN can be used to refer also to set of LANs. What we want to do is to utilize all possible LANs that are connected to the internet in our project. The more computers we have, the more problems we can solve by giving each computer a small chunk of work to do.

Memory

Each system has and manages its own memory. It is possible that each LAN supports several architectures of memory.

Cache

Each system or sub system manages cache on its own through the hierarchical systems.

Expandability

It is very easy to add more systems to the network.

Examples

The Internet is the best example for this type of model.

Chapter 3

Sample Distributed Projects

3.1 SETI@HOME

SETI stands for Search for Extra Terrestrial Intelligence [12]. This project was originally started at UC Berkley to analyze data from telescopes for strong signals in the skies. The main idea was to search for new life or new civilization in the galaxy. The problem was that the Super Computers could look for strong signals but did not have enough resources to look for weaker signals. They needed more powerful computers to be able to accomplish this. The decision came down to two things: Spend a lot of money on an expensive Super Computer or use smaller computers, but how many and how? The realization that a regular desktop computer spends more cycles displaying screen savers than doing something useful inspired the researches. And they did exactly that: replace the cartoon screen saver by one that displays “cool” graphs showing the analysis of data while actually using the CPU resources to analyze the data.

3.1.1 Setup

The telescope, “Arecibo”, records about 35 Giga Bytes of data on tape per day. The data is then sent (via regular mail) to UC Berkley where it is divided into 256 Kilo Byte chunks, with additional data about the “work-unit” to make a total of 340 Kilo Byes and sent to the internet to be analyzed by SETI@HOME users. After results from one work-unit are

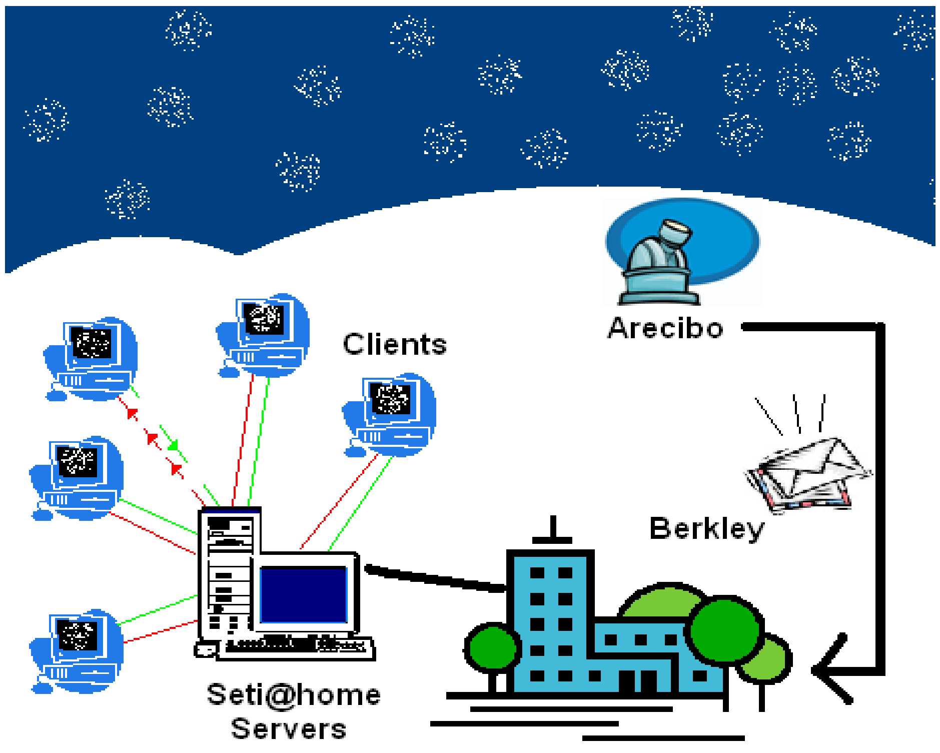


Figure 3.1: SETI@HOME System Architecture

submitted, the database at Berkley merges them into their central database and sends another work-unit, and this process continues. Several copies of the same work-unit are sent to different users to make sure there are no mistakes.

3.1.2 Client setup

Users can go to `http://setiathome.ssl.berkeley.edu` and download an installer based on their operating system. Once the software is installed, the user is allowed to set some options. Before a user can actually use the program, they have to create an account with SETI@HOME. Communication is only done when the data is sent and received, which is done only with permission from the user. The work is done only when the SETI@HOME screen saver is active. These are just the basics of the project; the details can be found at `http://setiathome.ssl.berkeley.edu`

3.1.3 Setup Parameters

Connectivity

Internet connection is only required when submitting results and getting new work.

Reliability

Several copies of the same work unit are sent to multiple users to avoid mistakes

Verification

By distributing several copies of the same problems, the chances of errors are minimized

Communication method

Communication is done through special ports using TCP/IP. Requires some tweaking to work behind a firewall

Expandability

New client version has to be downloaded for newer code or a different problem to solve

Fault Tolerance

The servers and results are backed up regularly to make sure the system is safe against any potential problems.

3.2 Distributed.net

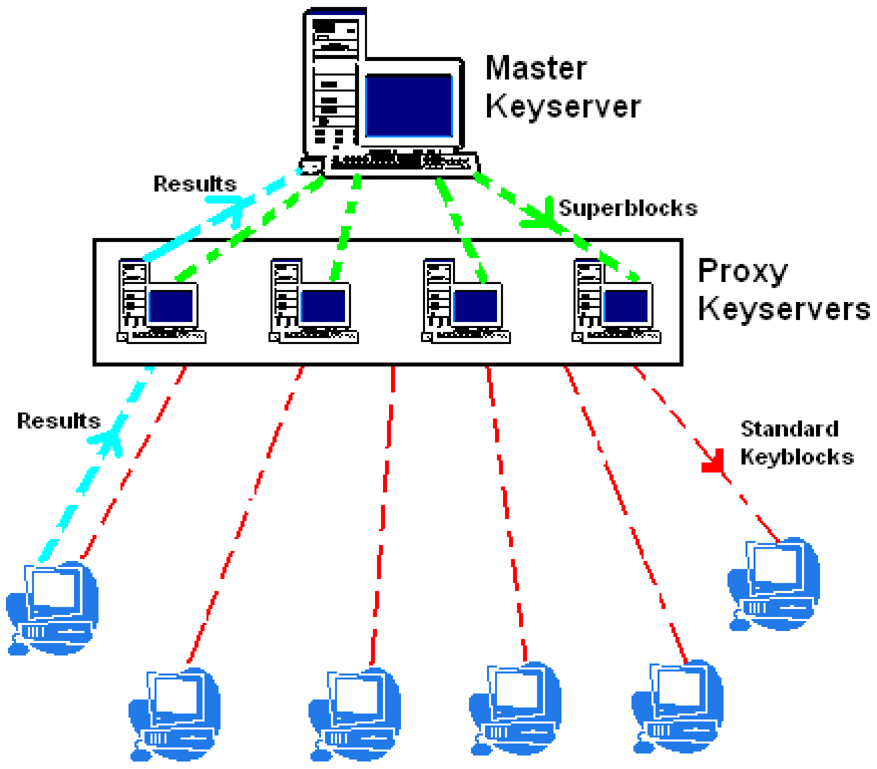


Figure 3.2: Distributed.net System Architecture

The RSA Security group is a major cryptography firm in the United States, selling cryptography, security and related products. There has been a lot of research done in the cryptography area since almost all levels of Government and non-Government institutions need to protect their data as well as transactions. RSA is very confident about its security algorithms and issues challenges to people (with a reward) to break its encryption by finding the correct secret key. Distributed.net [4] is a site that uses distributed computation to try and achieve this goal. In 1997, in coalition with the Electronic Frontier Foundation, it found the solution to RSA's RC5 56-bit key challenge, which took 250 days. Recently, in

2002, distributed.net cracked the RC5 64-bit secret key challenge, in 1,757 days by utilizing computers all over the world. Currently, distributed.net and its users are working to try and find the 72-bit key in the RC5-72 project. The prize money is split among the site and the user finding the winning key.

3.2.1 Setup

The site hosts several central servers that keep track of the keys. These servers specify the range of keys that each client works on. The results are then collected from the clients and stored on these servers.

3.2.2 Client Setup

The client can be downloaded from their website `http://www.distributed.net`. It can either be installed by windows or the files can be copied to any folder and run from there. It has a telnet like interface for setting up common options. It can either run in the foreground or through a screen saver. It communicates with the server to get the data for RC5 project as well as another project named OGR. There are a lot of options that can be tweaked by the user. There is no need to set up an account, the user can use a general account for taking part in the computation, but they have to sign up if they want to be considered for the reward.

3.2.3 Setup Parameters

Connectivity

Internet connection is only required when submitting results and getting new work.

Reliability

From the description available, only one client works on one key sub block.

Verification

The hierarchical structure of servers and clients exists so that the same block is not checked twice, therefore increasing the chances of possible error. The handling of such errors is not clearly laid out in the documentation.

Communication method

Communication is done through special ports using TCP/IP. It requires some tweaking to work behind a firewall

Expandability

A new client version has to be downloaded for newer code or a different problem to solve (RC5-64 bit clients are not compatible with the RC5-72 bit challenge)

Fault Tolerance

The proxy Key Servers serve as a Round Robin DNS, so if one of the them fails, the client will automatically switch to other available servers

3.3 Mersenne Prime Search

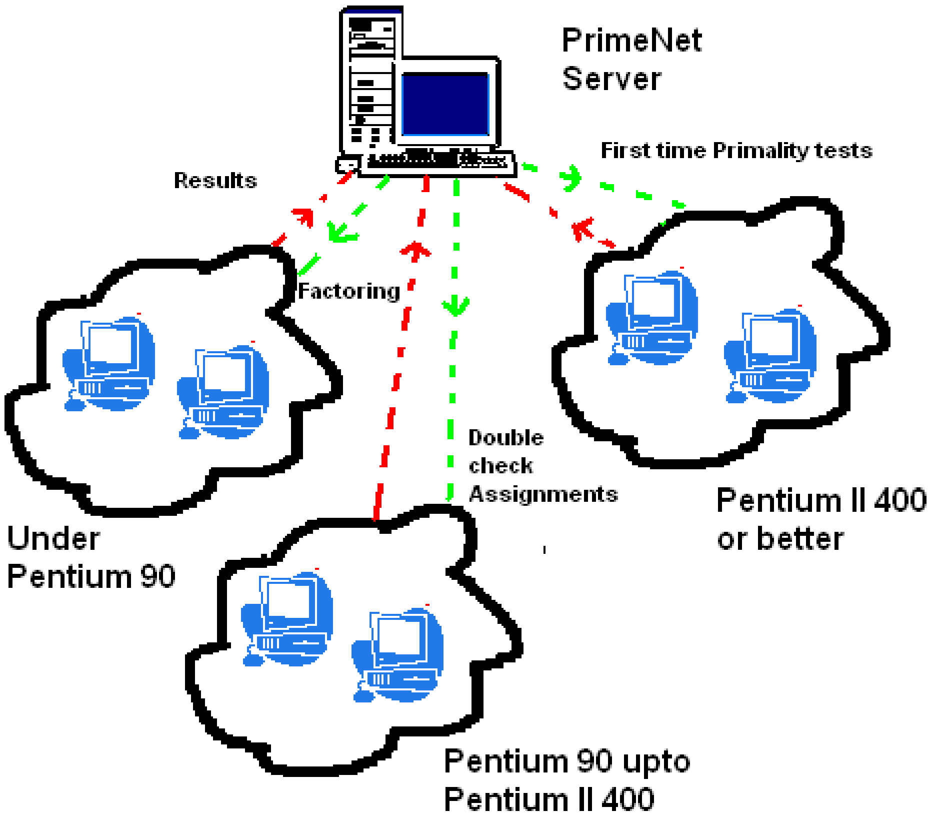


Figure 3.3: Mersenne Prime Search System Architecture

Known as GIMPS [16], the Great Internet Mersenne Prime Search, this project is designed to find Mersenne prime numbers. A Mersenne Prime number is of the form $2^p - 1$. This project was started in 1996 and users computers across the internet to find the prime numbers. So far, only 40 Mersenne prime numbers have been discovered.

3.3.1 Setup

A central server known as the PrimeNet server distributes exponents to the clients. All the communication is done via the HTTP Protocol. The work is sent to the client and results are collected by the PrimeNet server.

3.3.2 Client Setup

The client can be downloaded from `http://www.mersenne.org`. The installation program is a one Mega Byte file that can be downloaded and installed in Windows. There are two uses for their client software. For the user who wants to take part in the GIMPR project, you can either participate anonymously or by creating a user id with the client. Another option made available by the program is stress testing. This usage is meant for testing the consistency of the CPU. This is used typically to test CPU's, or in some cases to test the stability of over clocked CPU's (CPU's that are configured by the end user to run at a higher frequency than the CPU's rating, for example, running a Pentium 4 2.6Ghz processor at 3GHz by changing settings in the computer's BIOS).

The client communicates with the server every few weeks and sends only a few hundred bytes per connection. It lets the user specify how long the program should run, how much memory should be used etc. It also predicts the time at which a certain computation will complete. A computation "tag" or identifier is sent by the Server and displayed to the user.

3.3.3 Setup Parameters

Connectivity

Internet connection is only required when submitting results and getting new work. It is possible to use the client software only for testing the CPU, without participating in the research.

Reliability

As mentioned in the figure, different type of work is given to different class of machines. The program saves its internal state every half hour to avoid losing data, in case a crash occurs.

Verification

From the figure, it can be seen that a certain class of computers get work to double check the previous results. This way, the results from one machine can be verified through another machine.

Communication method

Communication is done through the HTTP Protocol and requires some tweaking to work behind a firewall

Expandability

A new client version has to be downloaded for newer code or a different problem to solve

Fault Tolerance

Internal state is saved so that data is not lost. It is not apparent but is likely the case that the Prime Net servers are backed up and have proxies in case one goes offline, another can take its place.

3.4 PiHex

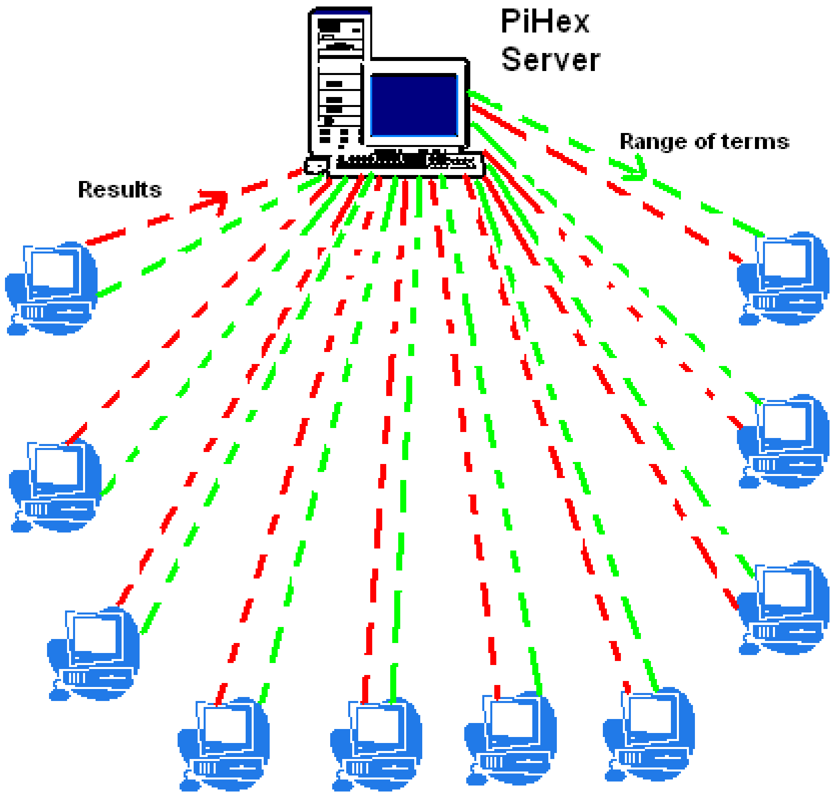


Figure 3.4: PiHex System Architecture

PiHex [10] was a project developed by Colin Percival at Simon Frasier University, BC. The goal was to compute millions of bits of π. The project has now been completed.

3.4.1 Setup

The idea is to give each computer a portion of bits to solve without solving for all the previous bits in π. This way it achieves efficiency by not doing previous redundant calculations, saving computing time as well as using only a small amount of memory on a client computer. A range of terms is assigned to each computer and results are collected.

3.4.2 Client setup

A client for PiHex can be downloaded in the form of a zipped file from `http://www.cecm.sfu.ca/projects/pihex/download.html`.

3.4.3 Setup Parameters

Connectivity

Internet connection is only required when submitting results and getting new work.

Reliability

If a machine does not communicate with the server after a fixed amount of time has passed, the work is automatically assigned to a different machine.

Verification

There is no information available regarding verification of results.

Communication method

All communication is done via TCP/IP using specific ports. It is fairly easy to configure the client to work behind a firewall.

Expandability

A new version of the software needs to be installed to start a new project.

Fault Tolerance

Since the work is reassigned if a client does not communicate, there is fault tolerance on the client side. It is not clear if there are multiple servers on the server side to provide fault tolerance in case of a Server crash.

3.5 Napster

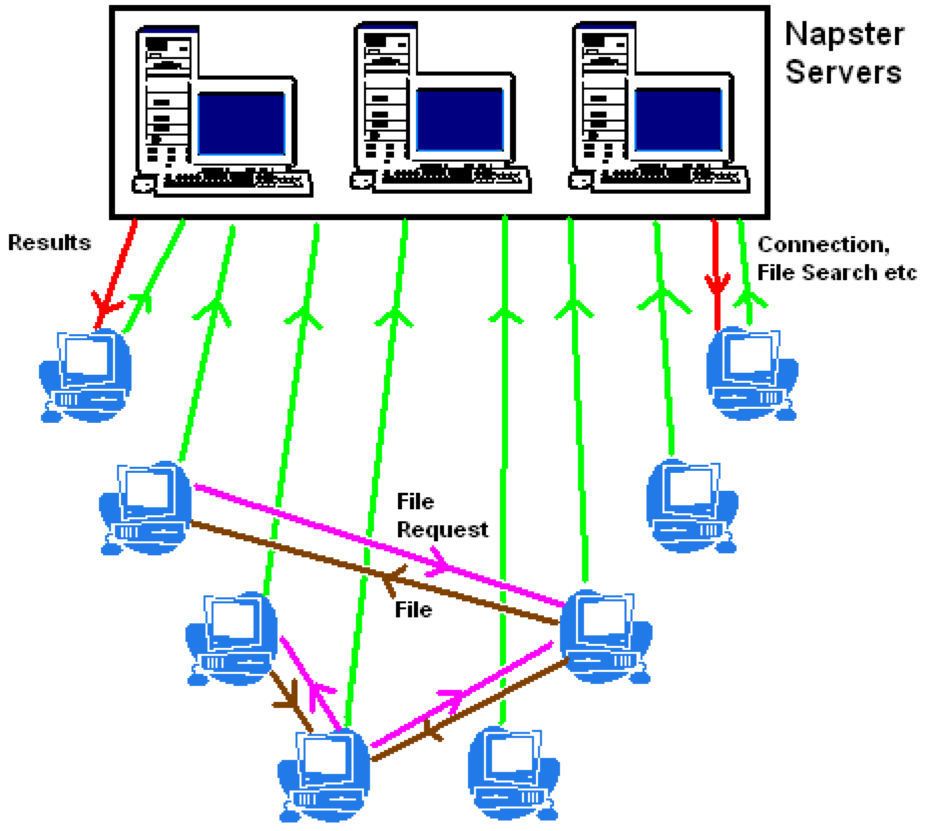


Figure 3.5: Napster System Architecture

Even though Napster [13] in its true form does not exist anymore, it started the evolution for making file sharing very easy across networks, countries and millions of users. The requirement was simple: every user needs some files and every user has some files to offer, Napster offered the public link between them. Though not a complete Peer to Peer network, Napster was probably the most popular distributed internet application and spawned quite a few other applications as well.

3.5.1 Setup

The central server is responsible for account and file list management. Users are connected to the central server that keeps track of the files and the users.

3.5.2 Client setup

The client can be downloaded from `http://www.napster.com`. After installing the program, you have to create a user name and password, which is a very simple process. You are presented the options of chatting with people or searching for files. When a search is done, a query is sent to the central Server, which searches through its list of files and returns the usernames that have the files. Queries can be complicated with additional options, such as bit rate, bandwidth of the users etc. Napster was designed specifically for songs in mp3 format.

3.5.3 Setup Parameters

Connectivity

Research has gathered that there is a constant connection to the Napster servers. This is concluded from the fact that there is a status of “on-line” and “off-line” within the client software.

Reliability

From experience, there is little to no reliability when it comes to downloading programs. This is because users in the past have liked to misguide searchers by changing file names. This does not seem to be the case with music files but is always possible due to the anonymity of the person who has the file.

Verification

The only possible way to verify if a file is what it is supposed to be is to open the downloaded files.

Communication method

The file lists are indexed by the Napster servers. The files are copied directly from the other clients, so there is a direct connection between the peers. Fixed port numbers are used for smooth operation behind firewalls.

Expandability

Only a new version would allow any different behaviour by the application. There were quite a few versions released of the client software, adding different features.

Fault Tolerance

If two users have similar file names with same file size in bytes, it is considered to be the same file and a file is downloaded in parts through different peers. This makes it possible to obtain high data rates if different peers are sending parts of the file.

3.6 Kazaa

Kazaa [9] took the ideas of Napster and expanded them further. Working on the same principles as Napster, Kazaa allowed users to share every kind of file; music, software, images, movies etc. It started becoming very popular due to the shutdown of Napster. This software however hosted a lot of spy ware, so people developed other versions of Kazaa that still work today.

3.6.1 Setup

The server is setup in a similar way to the Napster setup.

3.6.2 Client Setup

The client is pretty much setup the same way as in Napster, with a few changes: Kazaa allows multiple people to have the same user name, which means they keep track of users based on their IP addresses, and searches are not limited to music files as explained previously.

All other parameters are similar to Napster.

3.7 BitTorrent Network

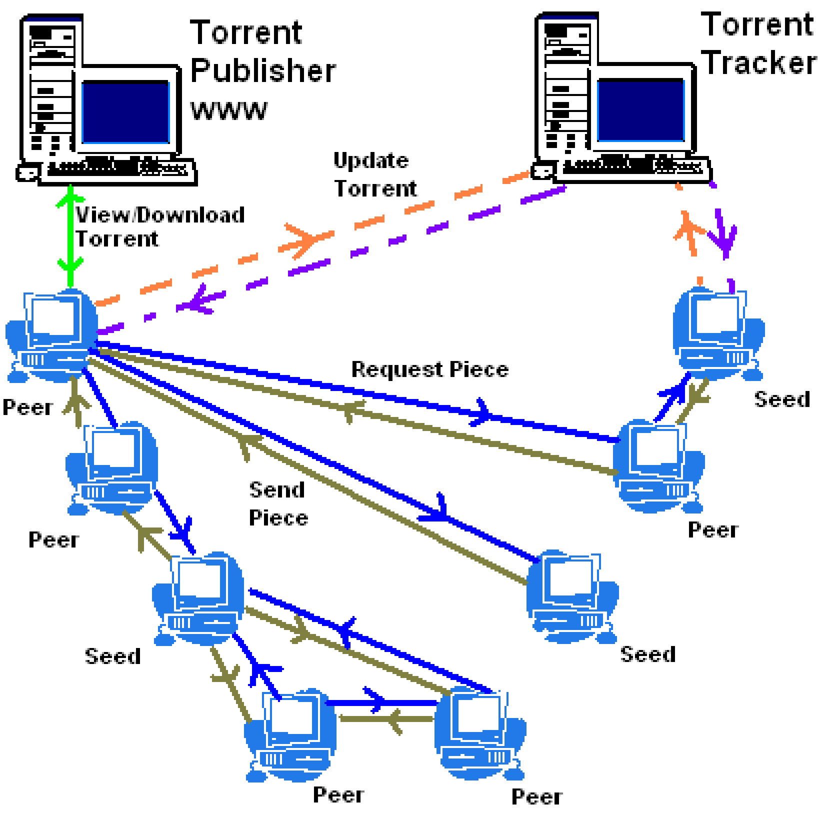


Figure 3.6: BitTorrent Network Architecture

With Kazaa's spy ware and the fact that it was becoming more commercial just like Napster, another alternative was developed by the community. This network [3] is not as popular as Napster or Kazaa, but with the passage of time it is growing very rapidly. This is a truly distributed network since there are no central servers. Technically there are no servers; there are only trackers (explained in detail in the next section), and anyone can become a tracker.

The main idea is that the files that are shared are published, rather than kept by someone and searched for. A lot of websites are setup that list "torrents". Torrents are files of small size that list details of the entity being downloaded. We call it entity because the download is not limited to just one file, it could be a set of files. A “Seed” is the user who has a complete copy of the torrent, and a “Peer” is a user who is trying to download the torrent. As the number of seeds increases, the chances of having a complete download increases as well. Even if there are no seeds, peers can still keep the download going, but the torrent will not be complete until there is at least one seed taking part in the download. It is also possible that a few users, together, be one seed. An example to illustrate this is that if user a has the first half of the torrent and user b has the second half, then there is one complete copy of the torrent available for download.

3.7.1 Setup

As we explained earlier, there are no central servers, just trackers. A tracker is a site that keeps track of the torrent. As users start to download the torrent, they remain in communication with the tracker and the tracker gets updated with information regarding the Seeds and the Peers. A lot of people have setup trackers that are tracking hundreds of torrents. The tracker then publishes the torrents either on its own website or some other website that lists torrents from different trackers.

3.7.2 Client setup

As of right now, there are about 20 different Bit Torrent clients. This is due to the fact that most clients are open source and therefore modified versions are easily available. The client that we have used is called “Azureus”, which is completely written in Java. After installing the software, a user can go to websites that lists torrents and start downloading. A very popular site, which forwards the users to available mirrors, is `http://www.suprnova.org`. A user can browse through the list and click on the torrent file, which is opened by the client. The client contacts the tracker and gets information about the download. In the same connection the client also finds out how many users are seeding the file, how many peers etc. Immediately, download as well as upload to and from other clients begins. A ratio is kept to show downloaded versus uploaded data. If there are no seeds available, it means that no client has a complete copy of the file, which means that possibilities of getting a complete copy of the file are very slim. At any point, someone can choose to seed a file and the chances to obtain the file increase. Some torrent sites monitor user usage based on login information (if required) or IP address and can ban users if their share ratio is low.

3.7.3 Setup Parameters

Connectivity

There is a constant connection to the peers/seeds while downloading files. There is a scheduled connection to the tracker to update the client's status at the tracker as well as to get status of seeds/peers. If this connection is not available later on in the download, the connection to the known seeds/peers remains open.

Reliability

The pieces are broken up into parts and downloaded one by one. The hash value of each piece is calculated as well as that of the whole torrent. These values are matched with the individual hash values of the pieces downloaded and pieces that don't match are discarded.

Verification

Until the files are viewed/opened/executed, one cannot tell if the torrent is what it is claimed to be. File misnaming has not been a huge issue, since most files are posted by users who are trying to be recognized for their efforts and would not want file misnaming associated with their aliases.

Communication method

All communication is done via TCP/IP using specific ports. The website states that the software does not work very well behind a firewall unless the ports are configured manually, but we have yet to run into that problem with our installation.

Expandability

Due to its open source nature, quite a few clients and new versions are available all the time. "Azureus" checks with its own websites for updates from time to time and upgrades itself if required.

Fault Tolerance

The chances of getting a complete torrent increase if the number of Peers and Seeds increase. The only issue of fault comes into play if the torrent file is available but the tracker is offline for the very first connection. This means that the client cannot get information regarding seeds/peers and therefore renders the torrent useless. If during the download, the client cannot communicate with the tracker, as long as it is connected to some Seeds and Peers, the torrent continues to download.

3.8 Problem with the approach

The problem with all these approaches is the requirement of the user to download and install the client program. As we see, this step involves going to the web site of the project, downloading and installing the software, setting up a user account in some cases, tweaking settings and then loading the client to start participation in the project. This step requires a lot of input from the user. With the advantage of Java Applet technology, we would like to show that this step is not necessary for the user. The amount of work the user has to do in order to join should be minimal, and our approach will show how it can be done via Java Applets.

Chapter 4

Java & Applets

In our project, we investigate the use of Java [14] for doing mathematical computation and distributed networking on a large scale. The main reason we need Java is for Applets, as will be explained later in this chapter.

Java also offers very easy use of its Socket classes, as well as to build a large Graphical User Interface (GUI) based application. Java also lets users Serialize Objects, which means that Objects can easily be transferred from the Server, over the sockets via Serialization, and rebuilt on the Client side with only a few lines of code. Any new Object can be Serialized with ease by just implementing the Java.io.Serializable Interface and including the methods to read the Object from a stream and write an Object to a Stream. Details of this are available in section 6.9.

Applets are “mini” applications, or light weight applications that run within the environment of a web browser. An applet is integrated within the HTML web page. When a browser parses a web page and finds the `<APPLET>` tag, it immediately knows that there is an applet that is embedded within the source. If the computer has JRE (Java Run Time Environment), the applet is handed by the JRE. If the computer doesn’t have JRE, then the user cannot see the applet. Most browsers come with a version of the JRE that is able to run most applets.

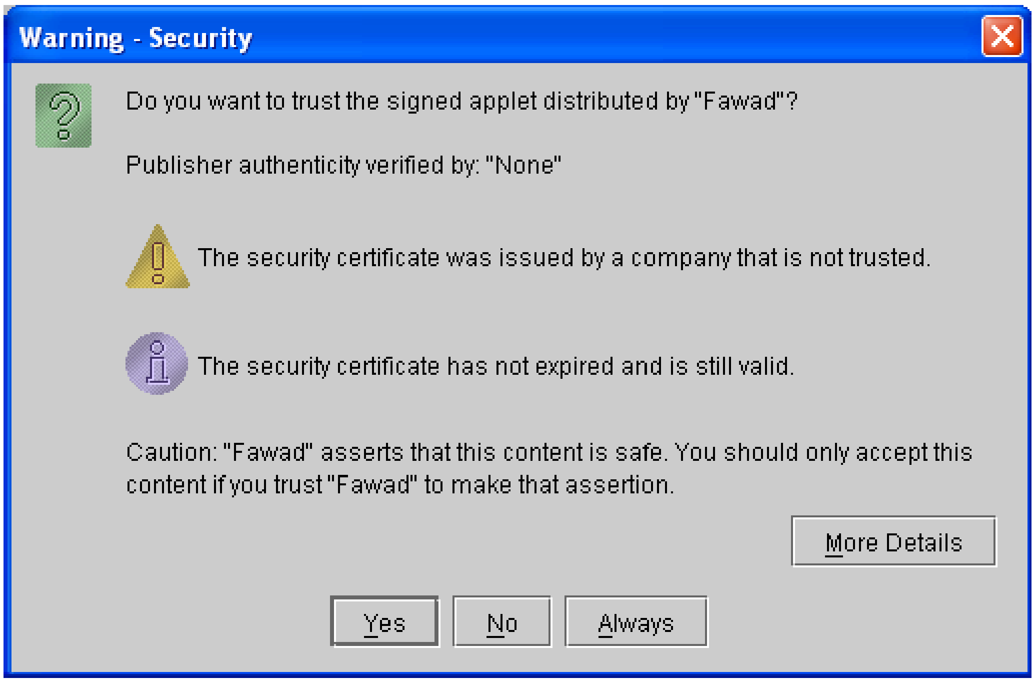
The applet is a small piece of executable code that runs within the JRE. The JRE has a built in “Virtual Machine” that runs the code for the applet. The Applet is downloaded by the browser and run locally on the machine. An Applet is not trusted by default and therefore not allowed to access most resources of the computer. We will see a comparison between unsigned applets (most applets available on the web are unsigned) and signed applets, to see what restriction each class presents.

4.1 Signed vs. Unsigned Applets

By default, the Java Virtual Machine does not allow applets to do anything more than accept keyboard input, display something on the screen and capture mouse movements. In order to let the developers benefit from the potential of Applets, in a proper application usage environment, which would open network connections, save files to disk and various other applications like behaviour, Sun introduced the notion of Applet Signing.

A digital Signature can be obtained from RSA for $400. The JAR (Java Archive, a file that stores the compiled code for the applet) can be signed with this Signature. When a user opens a Signed Applet that is signed by RSA, and their computer's root certificate trusts RSA for security, there is no difference to the user when they load the applet.

An applet can also be signed by anyone wishing to sign the applet, with the help of tools provided by Java. This of course means that by default the user's browser will not trust the applet and therefore a box similar to the one shown in figure 4.1 will alert them of the situation.


Warning - Security

Do you want to trust the signed applet distributed by "Fawad"?

Publisher authenticity verified by: "None"

The security certificate was issued by a company that is not trusted.

The security certificate has not expired and is still valid.

Caution: "Fawad" asserts that this content is safe. You should only accept this content if you trust "Fawad" to make that assertion.

Figure 4.1: Security Warning for Signed Applets

The user can choose to grant permission for one session, not grant permission, or always grant permission. Even if permission is not granted, the applet runs as if it was an unsigned applet: not allowed to use the resources that are available for signed applets. For a project that is along the magnitude and importance as ours, it is a good idea to get a code signing certificate from a trusted authority such as Verisign or RSA. The benefit of this is that when the applet is signed by trusted authorities, these are the same authorities that are (typically) a part of the browser's trusted sources and therefore the user trusts them as well.

The JRE allows different security levels under which signed and unsigned applets are executed. Following is a brief comparison of the two:

Category	**Unsigned Applets**	**Signed Applet**
Open a socket connection	Only to the applet's host	Open socket to any host
Reading/Writing files	Not allowed	Allowed
Read System properties	Only some can be read	All properties can be read
Invoke other applications	Not allowed	Allowed

Table 4.1: Applet Security

Even though table 4.1 is complete, for our purposes we are only interested in the first and second categories for the time being.

4.2 Applet Security Concerns

4.2.1 Opening Socket Connections

This is important to us because we want some Client machines to be able to execute as super nodes. A super node can be thought of as 2^{nd} level server, but the levels can vary arbitrarily. This would allow Clients to spread their work load onto other Clients. These Clients can be delegated by the main Server. This can be explained by the following diagram

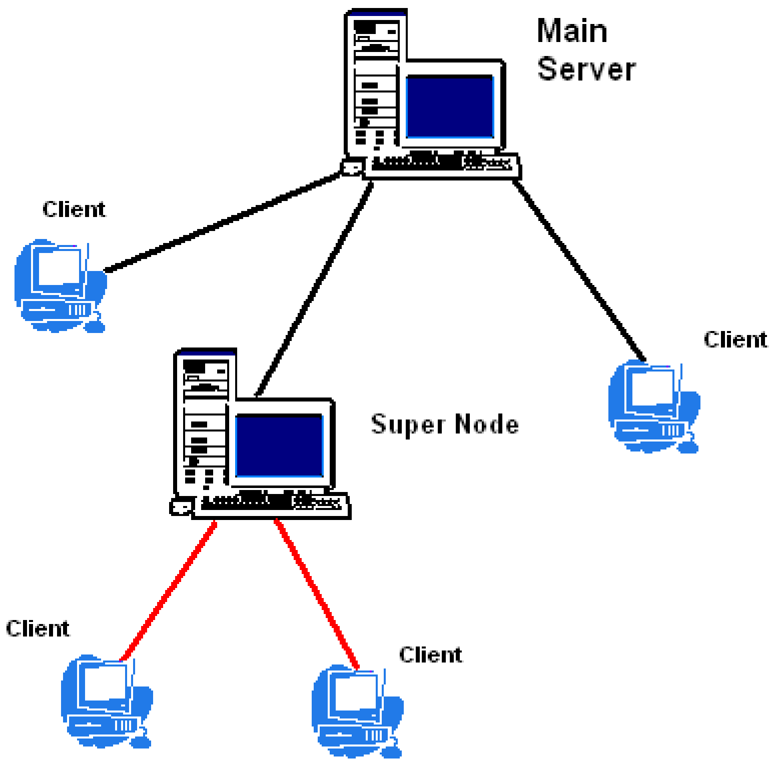


Figure 4.2: Super Node Diagram

In a very simplified form, figure 4.2 shows that we have the main server with some Clients connected to it, and then one Client acts like a super node, and it distributes the work further to other Clients, creating a hierarchical structure. In this way, we can have multiple Super Nodes that are capable of spreading the work to others and can get results even faster. For all this to happen, we need Clients to have permission to open sockets to other Clients than the main server for sending/receiving data.

4.2.2 Reading and Writing Files

For almost every application, writing data and reading data is very important. It helps in backing up the data to disk in case of a system crash. An unsigned applet does not allow you to write anything to the user's disk, or read anything from it. If we have this capability, we can write intermediate results to disk so that the next time we start up the applet, we can check previous results and continue on. This will obviously save a lot of time and can only be possible if we can read or write files.

Signed Applets enable writing to the local file system, which means it is possible to have a file sharing application written as an Applet, which would enable us to have a file sharing tool which does not need to be installed and updated all the time.

Chapter 5

Comparison of Integer Libraries

In order to facilitate us with choosing the best library to use in our distributed computation application, we ran the following tests that benchmark the computation time of 3 libraries: Java native Java.Math.BigInteger, an FFT based Java library LargeInteger written by R. Howell of Kansas State University [8] and a FFT based C++ library BigInt written by Xavier Gourdon [6].

The tests were conducted on a Intel®Pentium$^{\text{TM}}$4 running at 3124MHz with HyperThreading® and 512MB of RAM. Most non essential processes were killed before the tests were conducted.

5.1 Multiplying Integers of same size

The first test was conducted by generating two random integers a and b of the same length (the number of decimal digits being the same). Since there is no easy way of generating random integers of fixed length in java, this was done in the following way (testing only done on positive integers)

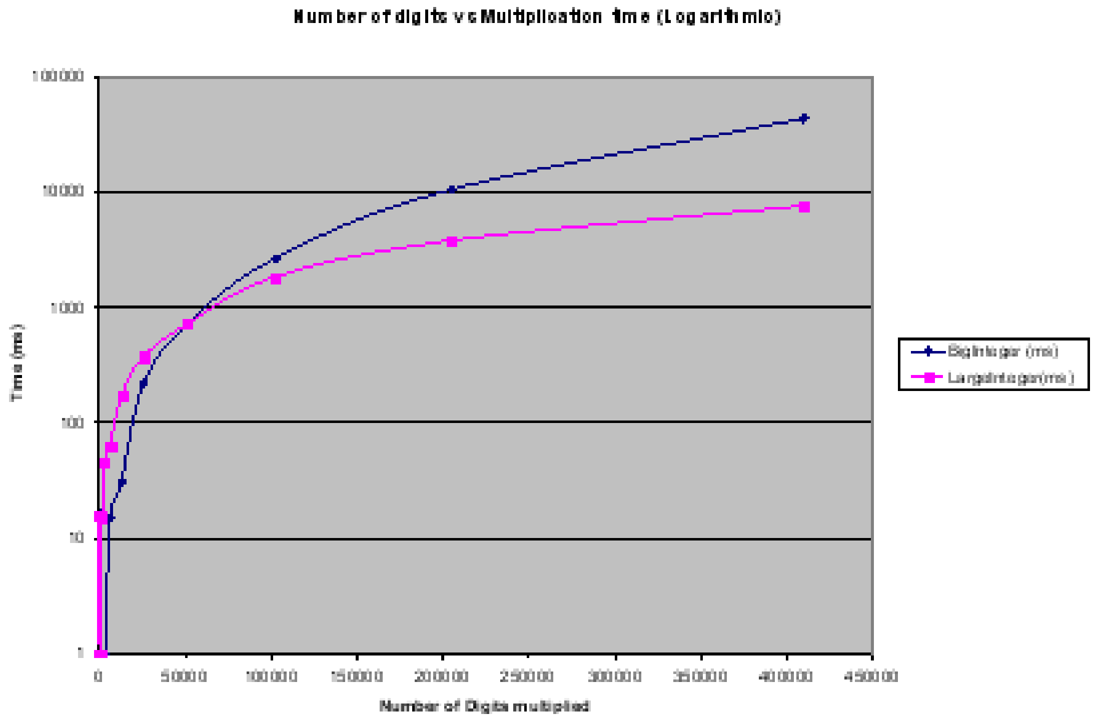
```
Repeat twice
    Let n be the number of digits required
    Create an empty String s
    Generate a random number x between 1 and 9 inclusive
    Append x to s
    While not the desired number of digits (length(s)! = n)
        Generate a random number x between 0 and 9 inclusive
        Append x to s
    End
    X1 = First integer created
    X2 = Second integer created
End repeat
Start timer
    Y = X1 * X2
End timer
End
```

Since the constructors of BigInteger and LargeInteger take a String as an argument, converting the Strings to integers was trivial. In order to speed up testing, the same random String was used to initialize the BigInteger object and the LargeInteger object. BigInt was not used in this test because it didn't seem possible to generate a String of digits and then initializing a BigInt from it.

Table 5.1 shows the timing results obtained from our testing. Please note that the results are taken over the average of 2 or more executions.

Number of Digits	**BigInteger (ms)**	**LargeInteger (ms)**
100	1	16
200	1	1
400	16	15
800	16	16
1600	1	15
3200	1	47
6400	15	63
12800	31	172
25600	219	375
51200	719	718
102400	2657	1844
204800	10485	3797
409600	43265	7578

Table 5.1: Same Size Integer Multiplication Results



.

Figure 5.1: Logarithmic graph of same decimal digit Multiplication vs. Time

Figure 5.1 shows a graphical representation of the logarithmic results. Most of the small computations took close to zero seconds; they were changed to 1 for simplicity.

We can see in figure 5.1 that for integers up to size 100,000, both libraries take about the same time for multiplication computation. However, with integers of much bigger size (400,000 decimal digits); LargeInteger is much faster than BigInteger due to its FFT based multiplication algorithm. This means that on integers of equal or almost equal size, using LargeInteger (on average) is faster and therefore more efficient than using BigInteger.

5.2 Factorial Calculation

In this test, we calculate the factorial of a number n. In this way, we will test how both libraries deal with multiplying small numbers with large numbers. This is to be done in the following way

```
Start
End = 64000
Accum = 1
For (i=2 to End)
    Accum = Accum * i
End loop
```

In this way, we are multiplying the accumulator each time with a relatively small number i. We refer to this computation as local computation, because if we think about calculating a factorial in a distributive fashion, each machine would do a similar type of calculation (with different starting and end points) locally and send back the results to the Server. The accumulator keeps track of the current value of the factorial and is multiplied by the next number on each iteration of the loop.

The result are shown in table 5.2 and the graphical representation appears in figure 5.2

Factorial	**BigInteger (ms)**	**LargeInteger (ms)**	**C++ BigInt (ms)**
50	1	16	1
100	1	31	1
200	1	141	16
400	1	734	78
800	16	3125	2672
1600	16	15797	1812
3200	62	80016	11203
6400	234	387703	41703
12800	1000	1819657	201206

Table 5.2: Factorial Calculation Results

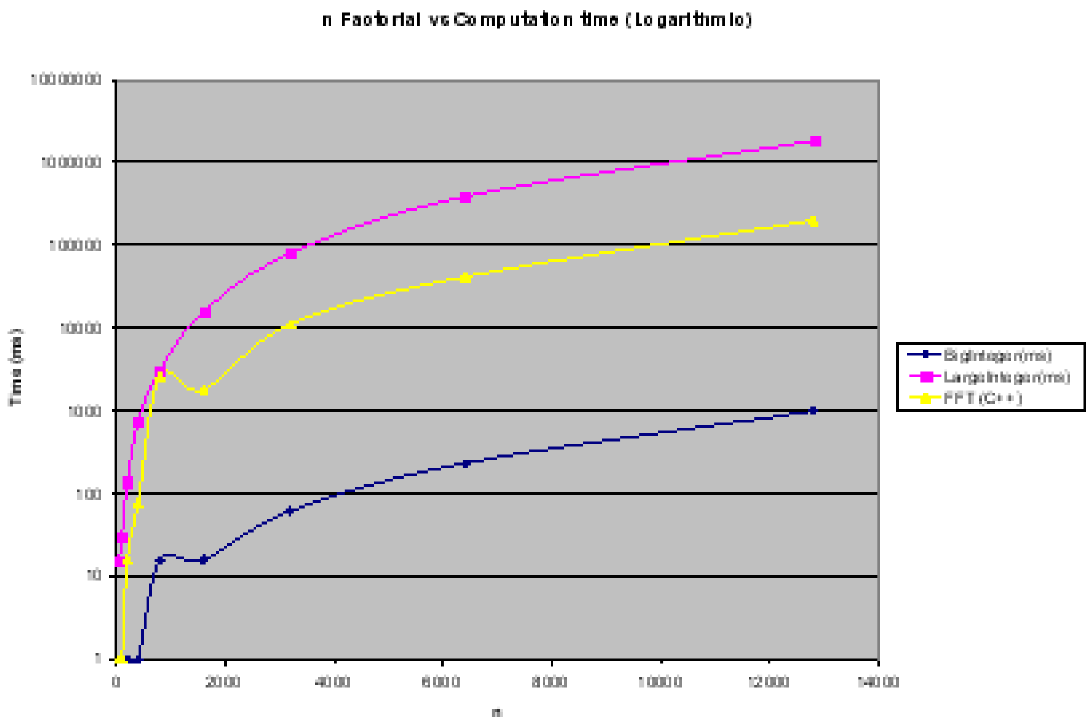


Figure 5.2: Logarithmic graph of n-Factorial Multiplications vs. Time

We can clearly see from figure 5.2 that the BigInteger library performs much better than any of the FFT based libraries. This allows us to conclude that for these types of calculations, the BigInteger library should be used.

5.3 Partial Factorial Multiplication

In this test, we test the amount of time it takes to multiply a partial factorial. We refer to this as global computation, because in our application, after the machines have returned their partial results, the Server will perform this computation over all the intermediate results. This test is performed in the following way

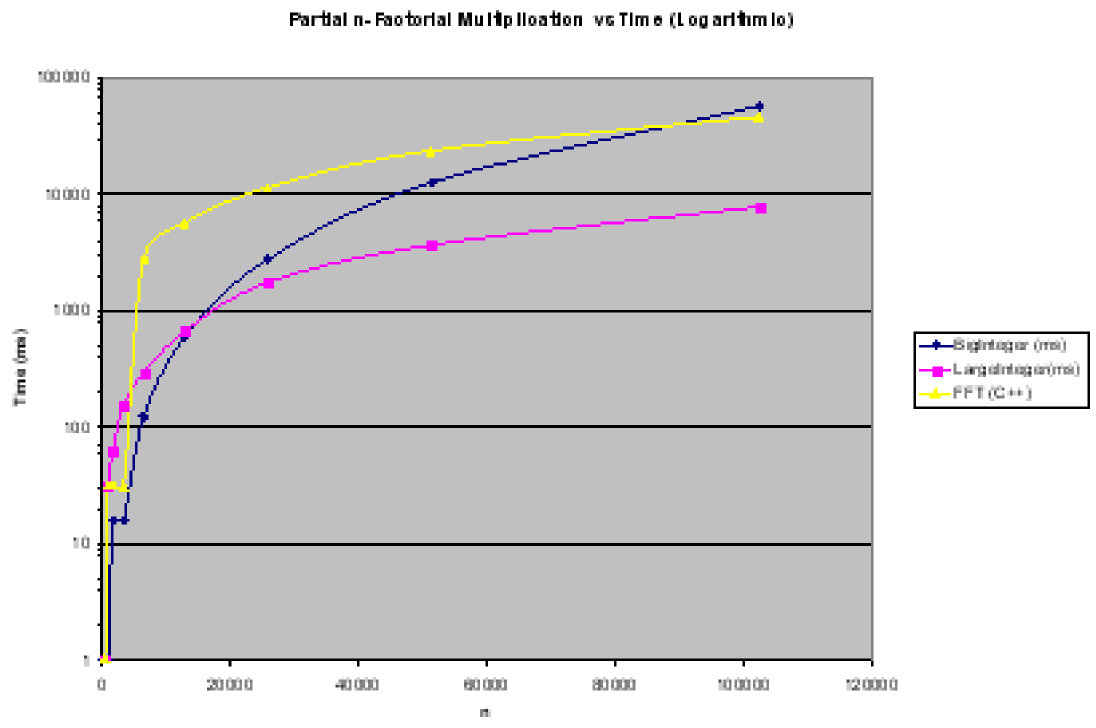
```
Start
// calculate factorial from 1 to n/2
End = 12800 (Sample End)
Mid = End / 2;
Accum1 = 1;
For (i=2 to Mid)
    Accum1 = Accum1 * i;
End loop
// calculate factorial from n/2+1 to n
Accum2 = Mid;
For (j=Mid+1 to End)
    Accum2 = Accum2 * j;
End loop
Time1 = get current time in ms
Result = Accum1 * Accum2;
Time2 = get current time in ms
Total Time = Time2 -- Time1;
```

Think of the problem of calculating the factorial of n=12800 distributed over 2 machines. One machine calculates the first half of the factorial and the second machine calculates the second half of the factorial and both return the results to the Server. The Server then combines both of the results into one result and we time this calculation from the server.

The results are shown in table 5.3 and the graph can be seen in figure 5.3

Partial Factorial	**BigInteger (ms)**	**LargeInteger (ms)**	**C++ BigInt (ms)**
50	1	1	1
100	1	1	1
200	1	1	1
400	1	1	1
800	1	31	32
1600	16	62	31
3200	16	156	31
6400	125	297	2880
12800	594	672	5720
25600	2750	1750	11440
51200	12641	3687	23842
102400	57906	7859	46724

Table 5.3: Partial Factorial Multiplication Results



.

Figure 5.3: Logarithmic graph of Partial n-Factorial Multiplications vs. Time

From the graph, we can see that initially all 3 libraries take almost the same time to multiply the partial factorials. As the size of n increases, BigInteger takes much longer than LargeInteger, but BigInt takes much more time than both and looks like it will gradually execute faster, but the tests were getting very long and CPU intensive. Some of this slow speed can be accounted to the memory overhead incurred by BigInt. From this information, we can deduce that when multiplying larger integers, the FFT based LargeInteger performs much better than BigInteger, but it is possible that in the long run, BigInt will execute about the same speed or a little faster.

5.4 Conclusion

From the results shown here, the task of choosing a suitable library for computation is not a very trivial one. Bear in mind that we have only really considered multiplication as the computation we are interested and not looked at division which is also a CPU intensive operation. Each library has presented its own merits and good/bad behavioural areas. The LargeInteger library is good for multiplying really large integers, and integers of different but very large sizes. The BigInteger library is good for multiplying smaller integers (even with one of the numbers being very big). It seems pertinent that we use the good features of both of these libraries. We take the fast small computation capabilities of BigInteger and fast multiplication behaviour of LargeInteger and fuse them together to form a faster system. This means that local computations should be done using BigInteger and global computations should be done using LargeInteger.

Chapter 6

Our Project: A Java based approach

Our implementation for a distributed project is a simple Client-Server architecture System. The Server and Client are both written in Java. We are aiming to achieve two goals with this project. Firstly, to develop an API for programmers to easily be able to write a distributed application quickly, defining a few derived classes and not worrying about how the distribution of problems, how their combination and how their management will take place. Secondly, to show that there are some real world gains by using a Java approach with distribution over the internet using Applets.

The software is written in Java, and thus requires that all developer-defined software components also be written in Java. Furthermore, the Java Runtime Environment (JRE) version 1.4 (or later) must be installed on each machine which is expected to run the software.

We can define the main sub systems of the System with the following diagram

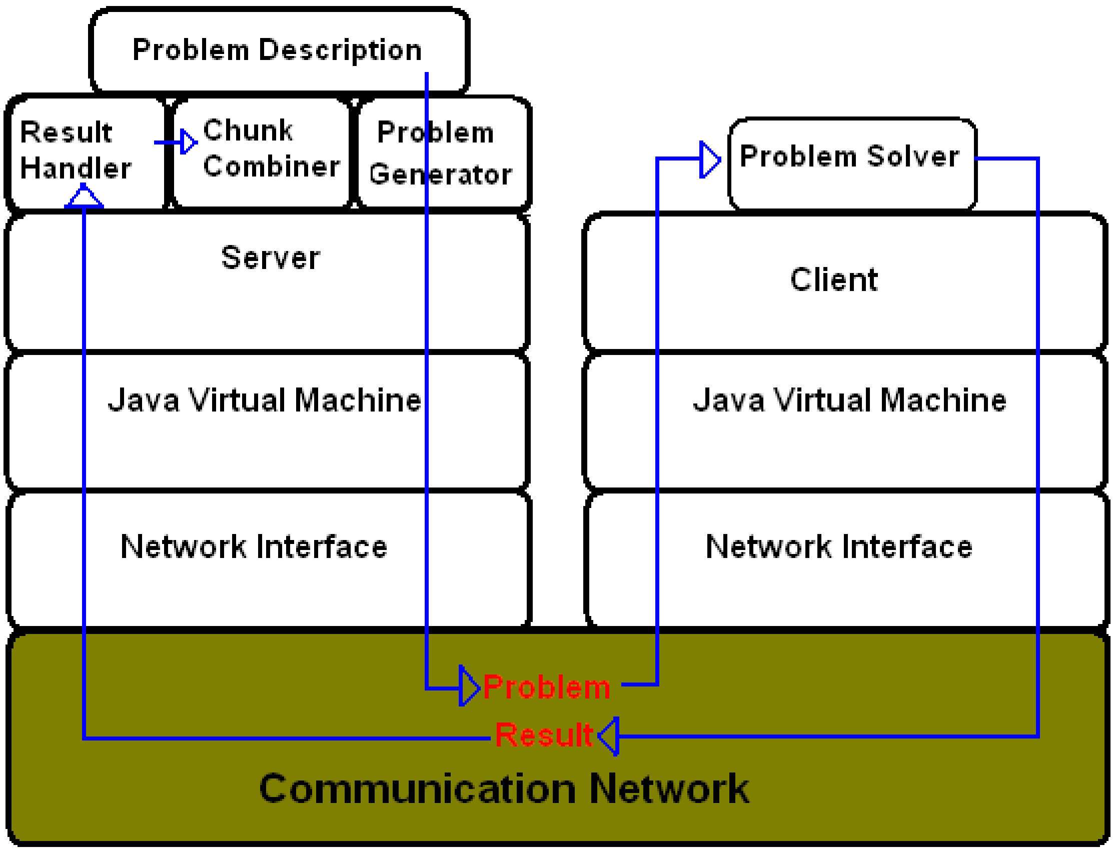


Figure 6.1: Main Sub-Systems of our Project

Figure 6.1 shows the main Sub Systems of our Project. Below is a brief description of the responsibilities and functions of these sub-systems.

6.1 Java Virtual Machine

The JVM must run on both the Client and the server sides. The JVM provides the interface between the Client and Server sub-systems, and the hardware and networking platforms on which they run. The JVM is also responsible for providing the platform- independent features of the system.

6.2 Server

The most important sub-system on the server-side is the Server. This sub-system runs on top of the JVM, and is able to communicate with Clients via the network interface and the network. Its principle responsibility is to manage various server-side data structures and components, as well as interacting with the three developer-defined components, namely the ResultHandler, ProblemGenerator and Problem (described below). The Server also manages the communication mechanisms through which transactions between Clients and the server will be conducted.

6.3 ProblemGenerator

The ProblemGenerator sub-system is responsible for generating Problems. Such problems are split into sub-problems by the Server (described later), which are sent to Clients on request, via the network. The sub-problems are received by the Client and processed. The ProblemGenerator is developer-defined.

6.4 ResultHandler

The ResultHandler is a sub-system that runs within the Server and takes care of the incoming Results. It notifies the Server that the partial results have arrived and keeps running until all the Chunks are evaluated.

6.5 ChunkCombiner

This sub-system runs within the Server and is responsible for combining the Chunks as results are made available. This gives us the advantage of combining the results as they arrive and not having to wait till all the Chunks are evaluated; we have more parallel computations this way.

6.6 Client

The Client is the most important sub-system on the Client side, and is responsible for maintaining various Client-side data structures necessary for its runtime functionality. The Client must request a Problem from the ServerEngine, which it processes. Once the processing is complete, the Client is responsible for sending the Result set back to the ServerEngine, where it is passed to the ResultHandler, as described above.

6.7 Network Interfaces and Network

The network interfaces and the network are the medium across which all Client-server communication takes place, and are thus critical in determining the overall performance of the system. The network must support the TCP/IP protocols.

6.8 Server Setup

For the setup of the Server, we require that the machine have Java Runtime Environment 1.4 or better installed. This will enable the pre-compiled classes to run the Server. For the user defined classes to be compiled, Java Development Kit (JDK) 1.4 or better is required.

Since we are trying to make it easy for developers to define quickly their applications by inheriting certain classes, we will first give a brief overview of what a developer needs to do in order for their application to run. This will involve showing which classes need to be sub classed and which methods should be defined and what their behaviour requirement will be.

6.9 Integrating developer defined application into the System

The following is a list of classes and methods of importance when a developer is adding their application to the System for distributed computation.

Class to inherit: DiCom.Problem

This is the class that actually defines a problem. For example, if you want to define a problem that calculates the factorial, you will create a sub class of this class and define the problem. It is important to follow the techniques for thread synchronization or method/object mutual exclusion by using the synchronize keyword wherever applicable. The following are the methods of importance to define in the sub classes.

```
 Result solve()
```

This method is defined so that the Problem can be evaluated without splitting it into any smaller sub-problems **(Here on referred to as Chunks)**. This needs to be over ridden so that the developer can define how to evaluate a sub-problem. The Result object needs to be retuned; a Result Object can be a user defined Object which sub classes Result class, or it can be any of the predefined classes available.

```
void split(int n, ProblemArray Problems)
     throws SplitFailedException
```

This method splits the Problem into n Chunks and returns the set of Chunks in a ProblemArray called Problems. If the Problem cannot be successfully split into Chunks, a SplitFailedException is thrown. It is possible that this method just calls on the constructor of its own class to create sub-Problems with different starting parameters. An example will be given later on.

```
int getRequiredChunksNumber()
```

This method is defined so that the developer can define the suitable number of Chunks for their particular application. This information is used when splitting the problem into Chunks.

```
Result combineChunks()
       throws BrokenProblemException
```

This method defines the behaviour to combine all the Chunks and come up with a Result. A BrokenProblemException is thrown if there is an error in combining the chunks. The developer has to define this method for their application to combine the Chunk's Results that were sent back from the Client.

```
void combineWithResult (Result _rResult)
```

This method is used to combine Chunks as the results arrive. We combine the current Result (current P & Q for the example of evaluating e) with the incoming Result. CombineChunks can be used for doing combination of results at one time, if required.

```
Result doFinalComputation()
```

When using combineWithResult (Result _rResult), if there is need for a computation after all Chunks all combined (for example, doing the final division$\frac{P}{Q}$), this is the method where such behaviour would be defined.

```
long getCombineTimeMs()
```

A method that will get the time it took for the Results to be combined. This helps in keeping track of how effective the System is.

```
void writeObject(ObjectOutputStream _rStream)
     throws IOException
```

Since the Objects are sent via the internet to the Client, they have to be written as Serialized Objects. Therefore, any important data members must be written via the writeObject method to the Stream.

```
abstract void readObject(ObjectInputStream _rStream)
              throws IOException,ClassNotFoundException
```

Since the Objects are sent via the internet to the Client, they have to be read as Serialized Objects. Therefore, any important data members must be read via the writeObject method to the Stream. The read must occur in the exact sequence the write has taken place.

There are 3 available Result classes: TwoBigIntegersResult (to be used when a value of P & Q is required), DecimalResult (when a decimal result is required, such as the rational value of P divided by Q) and BigIntResult, which used Java.math.BigInteger (this could be used for problems such as

$$\prod_{i=0}^{N-1} i$$

which will only have one Integer Result). Any of these classes can be used for obtaining Results.

Class to inherit: DiCom.Result

Since each application is different, the developer might want to define the types of Results it wants to generate. By inheriting this class, the developer has to define the following methods

```
abstract String toString()
```

This method is used for printing the Object. The developer can define how the Result can be printed.

```
void readObject(ObjectInputStream _rStream)
     throws IOException, ClassNotFoundException
```

Since the Result has to be sent via the internet to the Server from the Client, it has to be Serialized and the developer must write the data members to the Stream.

```
void writeObject(ObjectOutputStream _rStream)
     throws IOException
```

Since the Result has to be sent via the internet to the Server from the Client, it has to be Serialized and the developer must read the data members from the Stream. The read must occur in the exact sequence the write has taken place.

Interface to implement: ProblemGenerator

This class is just used to generate Problems. It has a single method that needs definition

```
Problem getNextProblem()
```

This method must return a Problem to be evaluated. Here a developer can just create and return an instance of the Problem they are trying to evaluate.

Class to Modify: Server

The Server must be modified so that it knows which ProblemGenerator to use. It is possible to parameterize this modification but as of right now, the following line of code must be modified

MyProblemGenerator m_rProblemGen = new MyProblemGenerator ();

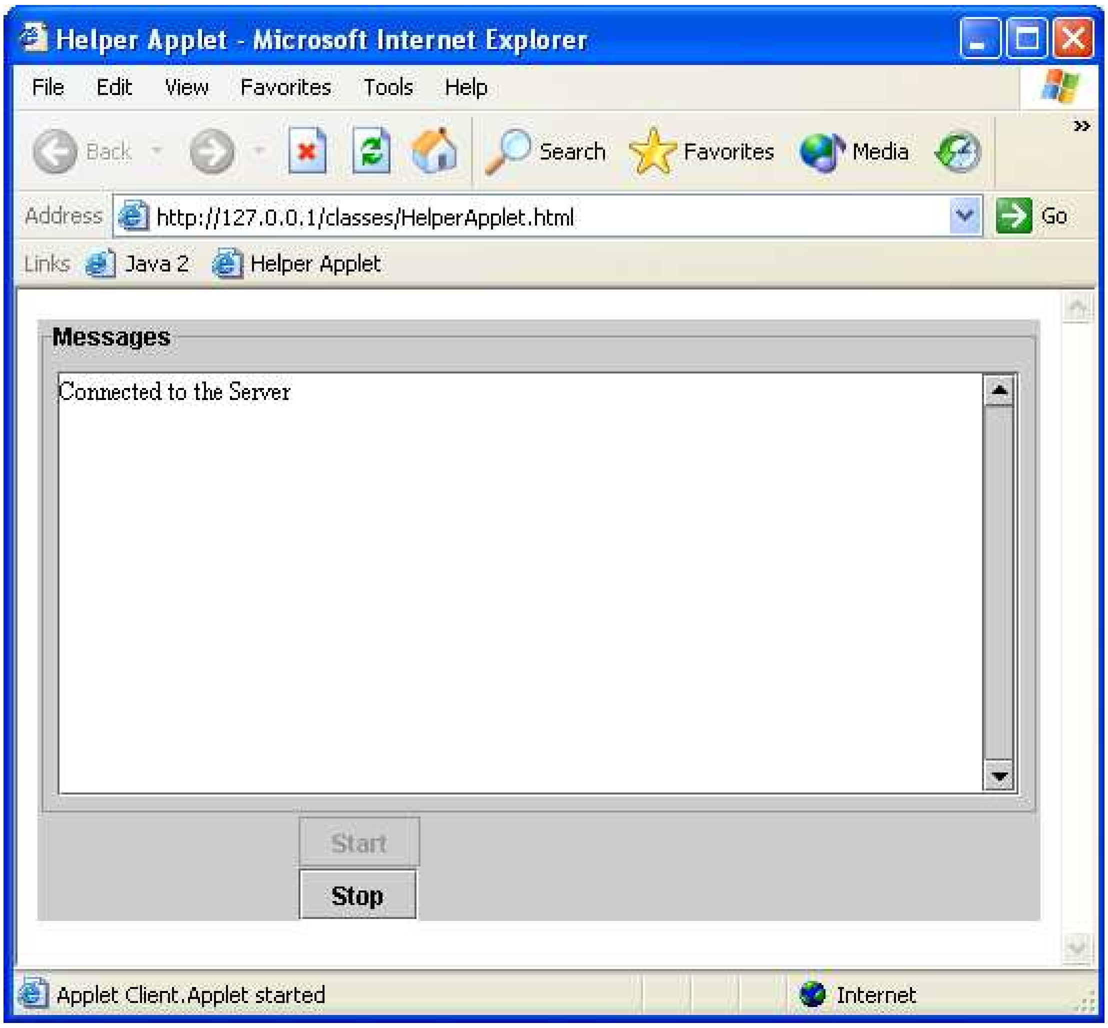
MyProblemGenerator needs to be changed to whatever the developer's class name is for the *ProblemGenerator.*

The combined definition of all these classes is all you need to have a working distributed application for solving Hypergeometric Series.

6.10 Client Setup

Since the Clients are connected via Applet, a developer can either make the applet available offline or run via Sun's Applet Viewer, or a web server can be used to host the Applet. The Applet is always downloaded and run locally on the Client's machines. The web server is a more popular solution, since a developer can just distribute the links to the site to the prospective Clients.

A user visit's our website (for example http://www.fawad.ca/research/) and clicks on the "I want to participate" link. They will be taken to a web page that will load a simple Java Applet. Since we are not yet doing any Security Certificates, the Applet will not be signed and therefore will simply open connections only to the website it is hosted on. The applet has a very simple user interface like the following


Helper Applet - Microsoft Internet Explorer

Address http://127.0.0.1/classes/HelperApplet.html

Messages

Connected to the Server

Applet Client.Applet started

Figure 6.2: Client Interface

6.11 System Architecture

The overall System Architecture of our project is very simple. It can be understood with the aid of the following diagram

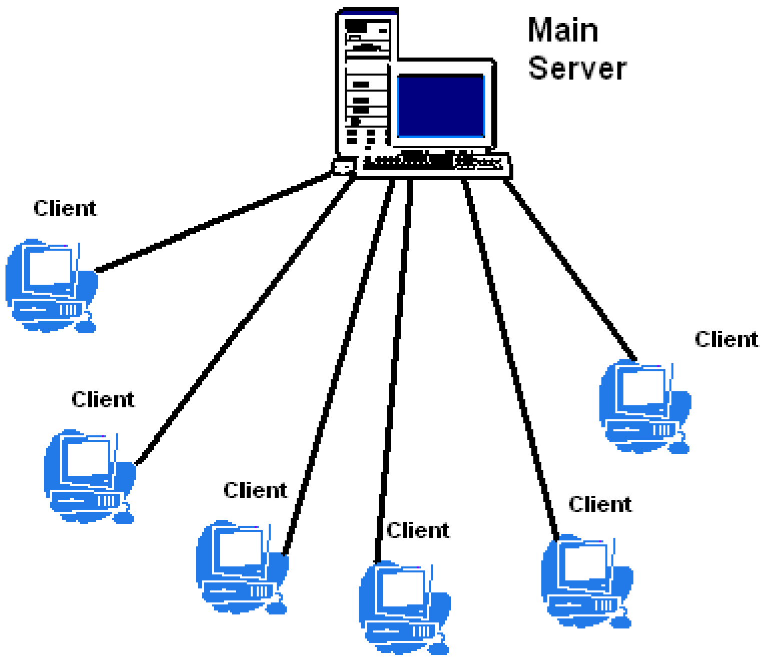


Figure 6.3: System Architecture

As can be seen in the figure above, our System is a very simple Client-Server type System. The Server is a Java Application running on our servers and the Clients are Java Applets running remotely via the internet on computers all over the world. This is a very simplistic view of how we are setting up our System. To get a detailed idea, let us look at the simple scenario of the interaction between the Server and only one Client.

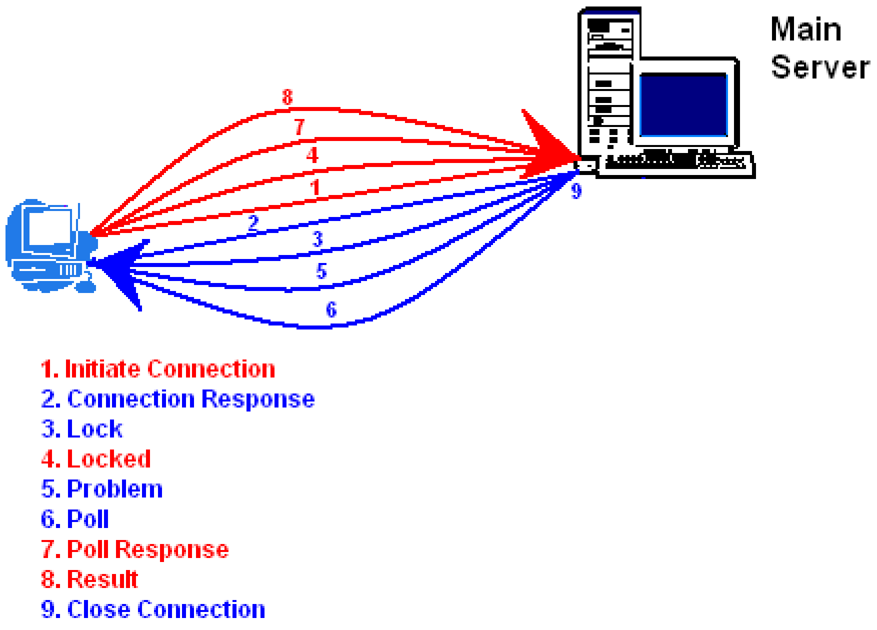
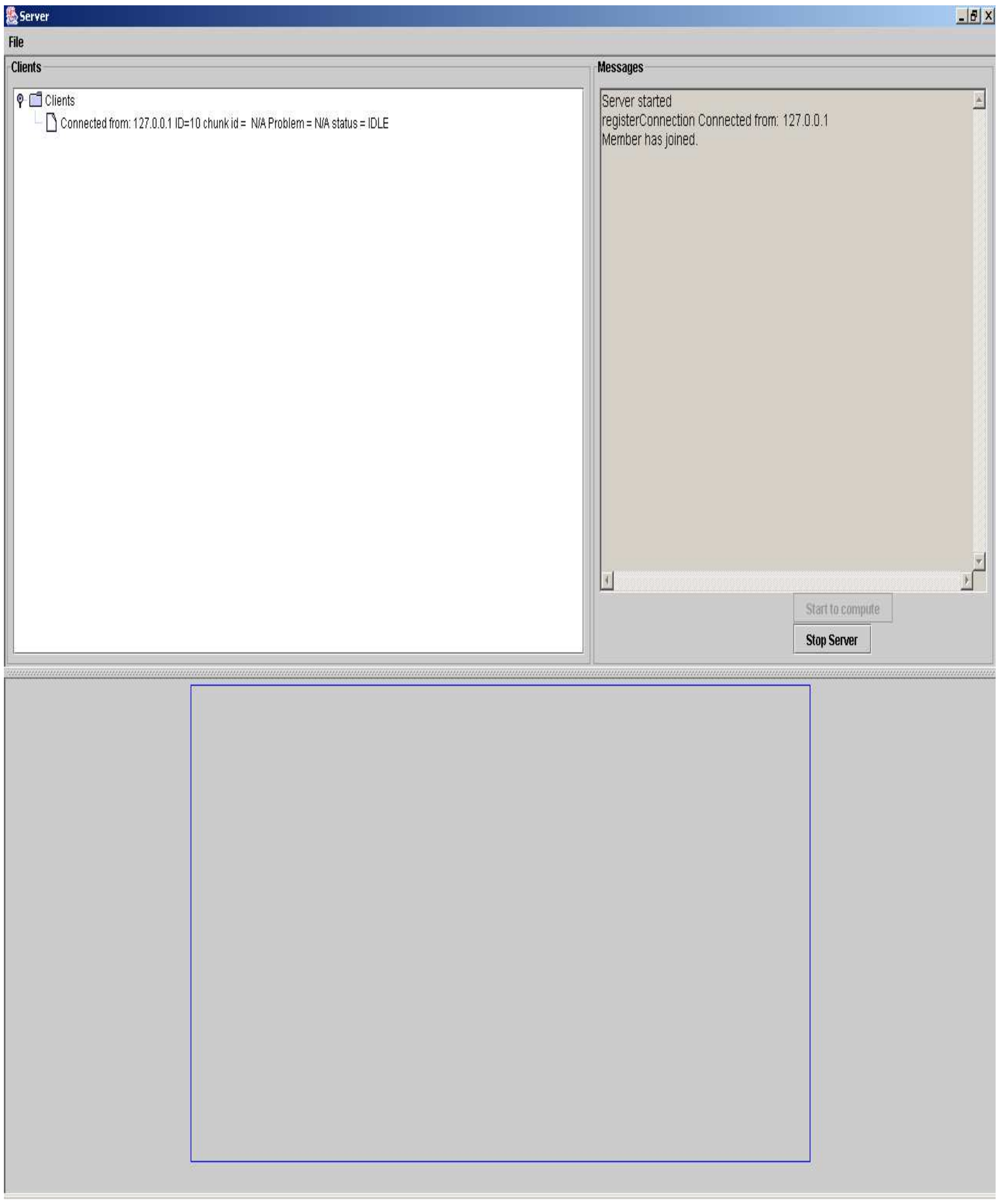


Figure 6.4: Single Client-Server interaction

As we can see from the diagram, we have a scenario in which there is only one Client and only one Server. Let us walk through a typical user and their connection to our System.

The Client is loaded in a web browser and a connection is initiated by the Client to the Server. On the Server side, the Server recognizes that there is a Client connected to the System; it sends a connection response to the Client and updates the User Interface as shown in figure 6.5.



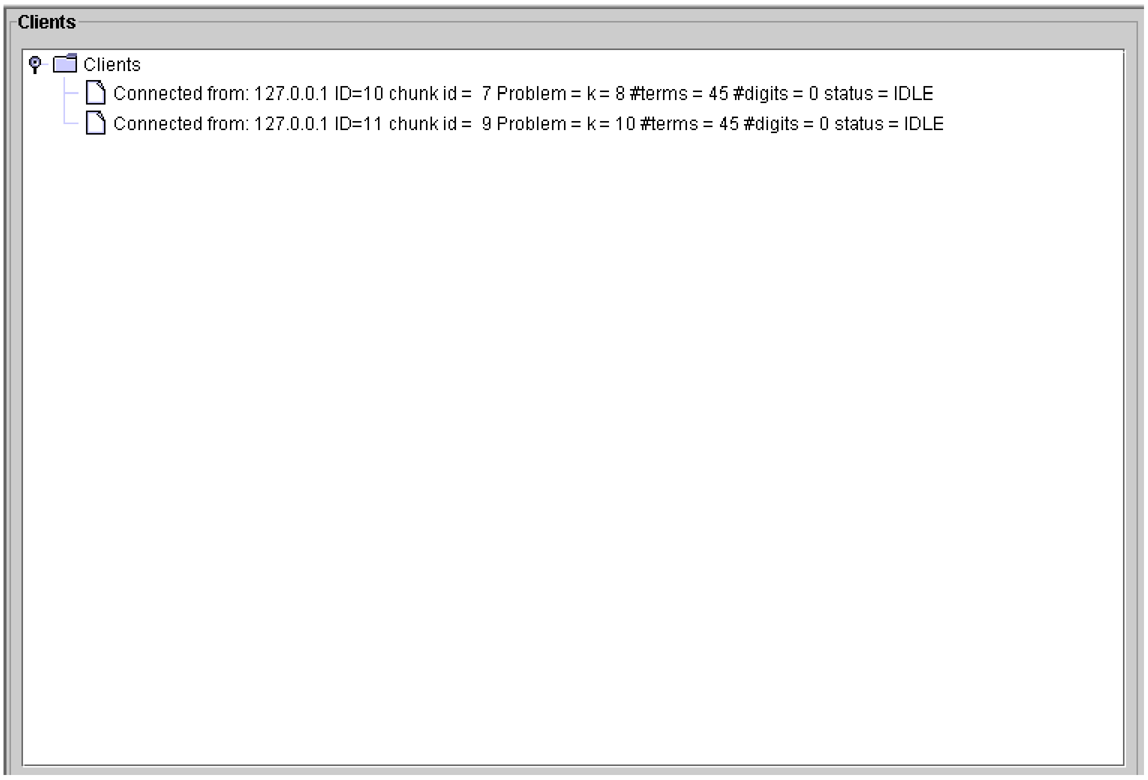
Connected from: 127.0.0.1 ID=10 chunk id = N/A Problem = N/A status = IDLE

```
Server started
registerConnection Connected from: 127.0.0.1
Member has joined.
```

Figure 6.5: Server Interface showing connection by one client

For the simplicity of the System, let us assume that we actually have 2 Clients (so that we are doing some shared work). When the second Client connects, the screen is updated similar to the one shown above.

The Problem is split into however many chunks are required (stated by the Problem itself). If we have that many Clients, that is the optimal situation, but in a situation that we do not have as many Clients as Chunks, we can either reduce the Chunks to match the number of Clients, or we can just have multiple Chunks to be given once the Clients finishes one computation (this approach is better since in the event of another Client connecting, we can give it some work as well). The Server then sends a lock message to the Client and waits for a response within 10 seconds. If there is no response from a particular Client, it is assumed that the Client is no longer available for computation. Once the lock responses are received, the Server sends the Chunk to each of the Client and updates the screen, as can be seen in the following series of diagrams which are just zoomed in for clarity

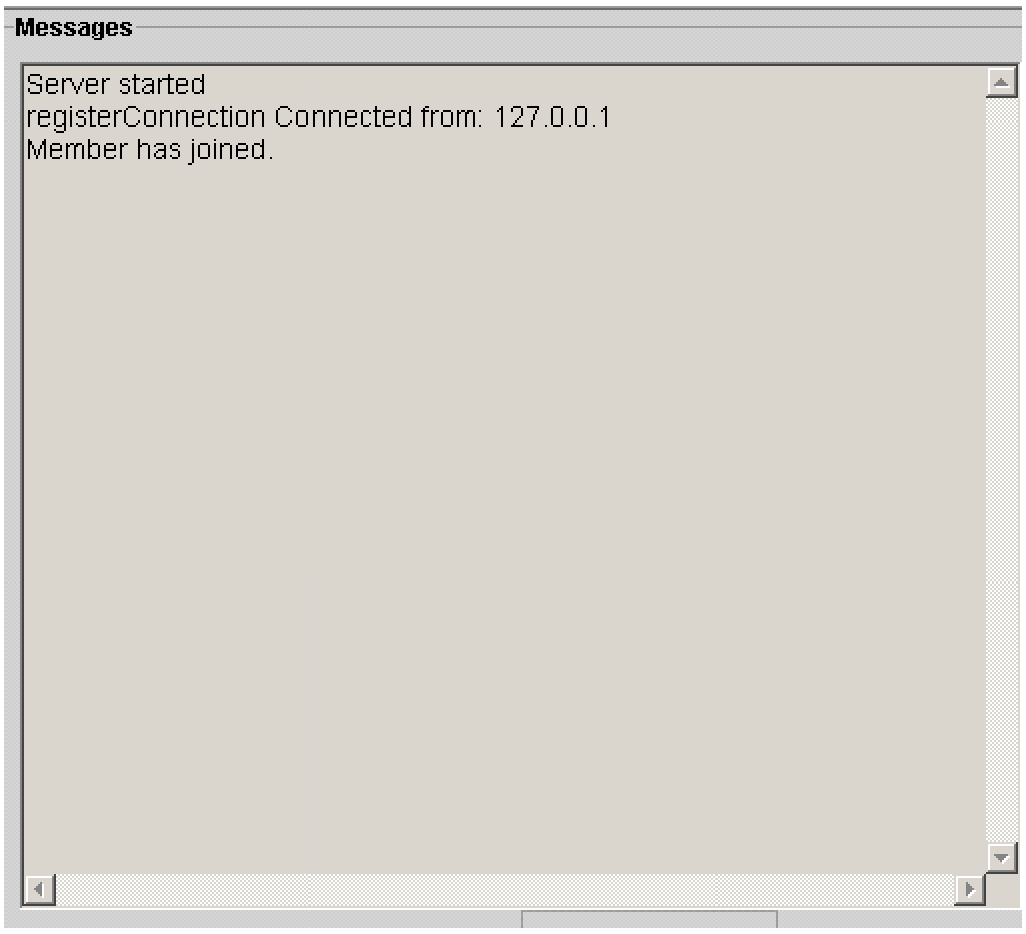


```
Connected from: 127.0.0.1 ID=10 chunk id =  7 Problem = k = 8 #terms = 45 #digits = 0 status = IDLE
Connected from: 127.0.0.1 ID=11 chunk id =  9 Problem = k = 10 #terms = 45 #digits = 0 status = IDLE
```

Figure 6.6: Clients List

In this list we see that we have 2 users, their IP addresses (both Clients are running on local machines), the chunk ID of the chunk they are working on, the details of the problem (this is user defined, for this particular problem, the value of k is used, and the number of terms for the computation), and finally we see the status of the Client.

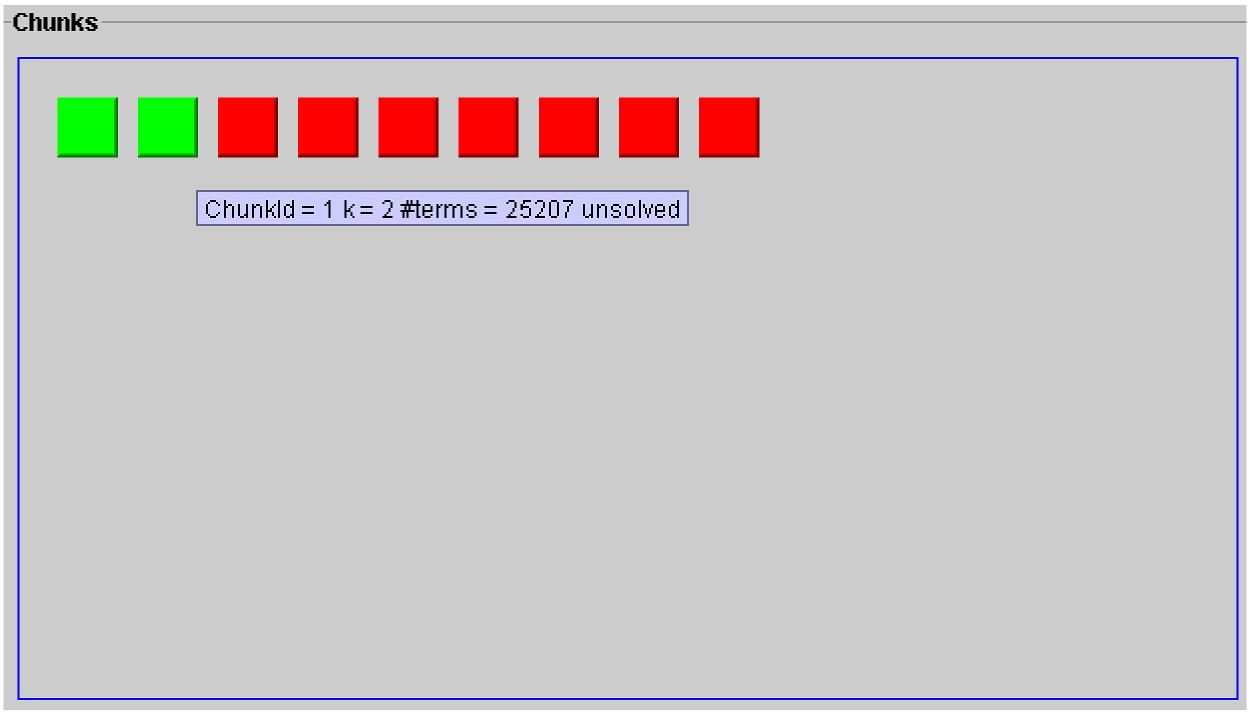
There is also a messages window, which outputs any messages, logs or print statement that the user or the Server has programmed to be printed.



```
Server started
registerConnection Connected from: 127.0.0.1
Member has joined.
```

Figure 6.7: Messages Window

The diagram that follows shows the completion of Chunks so that the user can follow the progress of the computation



ChunkId = 1 k = 2 #terms = 25207 unsolved

Figure 6.8: Chunk Completion diagram

In this diagram we see a visual display of the Chunks. The “alt” text shows detailed status of the Chunk, while the Chunk color defines its immediate status

- Red means the Chunk has not been evaluated and there are no members solving the chunk
- Green means the Chunk is being currently evaluated
- Blue means the Chunk is already evaluated

Once the chunks are distributed, the Client immediately starts to work on them. In order to make sure that the user has not disconnected (gone offline, crashed, rebooted etc); the Server sends a Poll message to the Clients every 10 seconds and waits for a response. If there is a response, there is no change made to the System. If there is no response from a certain user, the user is considered KILLED, removed from the list of Members, the Chunk status is changed back to unevaluated and unassigned.

If another user joins during the computation, the first available chunk that is unsolved and unassigned is assigned to them. If there are no available chunks then the Client just waits for a problem to be sent from the System.

When the Client is finished doing the computation, it sends back the Result to the Server. At this point, the Server will either assign an unsolved chunk to this Client if there are any; otherwise it will disconnect the Client. Once all Clients are disconnected, it performs the final calculation i.e. combining the results specified by the user defined Problem. This result is printed to the screen along with the time it took for the whole computation as well as the time each Chunk took to be evaluated.

In the setting of multiple Clients, we have a very similar picture as before but with multiple Clients

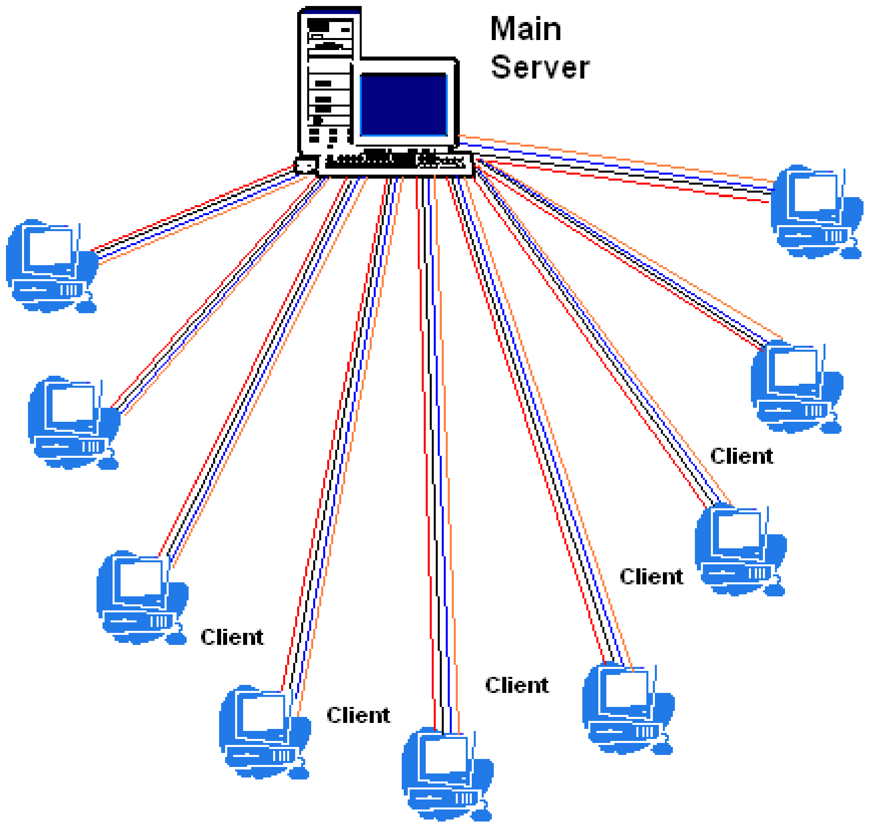


Figure 6.9: Multiple clients connected to our Server

In this case we just have more Clients to handle, following the same processes as stated before. New Clients are welcome and old ones who do not respond are removed from the System.

Let us now look at some of the Setup Parameters that we have evaluated for all other Systems

6.12 Setup Parameters

6.12.1 Connectivity

Since we are doing computation over the internet in a web browser, at this time it is crucial that the user is connected to the Server at all time since the Server Polls the user from time to time. We have to look at the sensitivity of such an application since the user can just close the browser window and we would have no way to find out if the user is still connected or not. Depending on the size of the computation, the user might be involved in the computation for a long time and submit results after long periods, in which case an offline approach is better than the current one. On the other hand, if the computations are small, then the Results need to be sent back and new Problems received frequently, in which case a constant internet connection is more useful.

6.12.2 Reliability

If a Client does not respond to Poll messages, then the Client is assumed to be dead and the Chunk is then put in a pool of unsolved Chunks. When other users are connected to the System, the Chunks are then distributed to them as well. A modification to our System can actually solve this problem, which will be presented later on.

6.12.3 Verification

At this point there is no method for verification of results, but the modification previously mentioned can aid us in this task. We can also build verification by doing more distributed computation but this time on the inverse of the series we are trying to evaluate, and finally multiplying the two results. For example, we obtain $\frac{P_1}{Q_1}$ for evaluating $e$and then we compute $\frac{Q_2}{P_2}$ for evaluating $\frac{1}{e}$ and then we verify that $R = \frac{P_1 * Q_2}{Q_1 * P_2} = 1$. It is not necessary that every hypergeometric series would indeed have a hypergeometric series representation of the inverse.

6.12.4 Communication method

All communication is done via TCP/IP using specific ports. It is fairly easy to configure the Client to work behind a firewall.

6.12.5 Expandability

This is the best part of our project. Whenever a user connects to our website and opens the applet, the latest version of all the Software that is required by the Client is given to the user. This means that if we are actually solving a different problem this time, the user does not need to download anything new. The interface for the Applet can remain the same, since the Client does not need to know that a new problem needs to be solved.

Another great thing about this is that the Client code is just a sub class of the Server code. With this comes the great benefit of super nodes and sub nodes, as mentioned before. With signed Applets, we can enable some Clients to become Super nodes, so that they can further distribute some work among other Clients. The behaviour here will be a little different, as the Server would still be the only one hosting on a website and would just forward the details of new Clients to these super nodes so that they can communicate directly. As with most other distributed applications, only with the user's permission will the Client start its Super node activities.

6.12.6 Fault Tolerance

Since the work that is incomplete by one Client is reassigned to other Clients, we have fault tolerance, but on a very small scale. With the aid of signed applets, we can actually save partial results to local files so that on a new launch of the applet, we can check our state and continue the computation rather than requesting a new Problem from the Server and starting to work on it from scratch. There will be some behavioural changes from the current setup, since a Client is assumed disconnected if it does not respond to Poll messages. This will mean that on the current Setup, the "restarted" Client will be assumed to be a new Client. Changes on the Server side and on the Client side, once implemented will aid in this new and improved working of the Clients.

6.13 Modification to the System

Suppose that we have a machine that is free most of the time and we (the Server) knows the IP address of this machine. This means that we can use this machine for our computation benefit, in such a way that there is no user intervention required. We can actually create a very specific Client for this machine, which can be launched remotely and which can do any type of work that we need it to do. Let us call this machine the “Assistant”.

We can use the Assistant to perform any type of work that we want, since we know where it is located and that is it available to be used any time. We can use it for any of the following tasks

1. Verify partial results sent from Client
2. Evaluate Chunks sent from the Server
3. Combine the Results sent from Clients to compute final Result
4. Always be available to take a part in the Computation

In essence we have a Client that is always available for helping us and we can use it in any way we want. If there are no Clients available to start the computation, we can just start by giving some work to this machine while waiting for other machines to connect. It is entirely possible that this single machine will evaluate all our Chunks for us. Since we have a control on this machine, we will assume that this machine has reliability in terms of Results, a reliable connection to us and is fault tolerant to a certain degree. We can see this via the following diagram

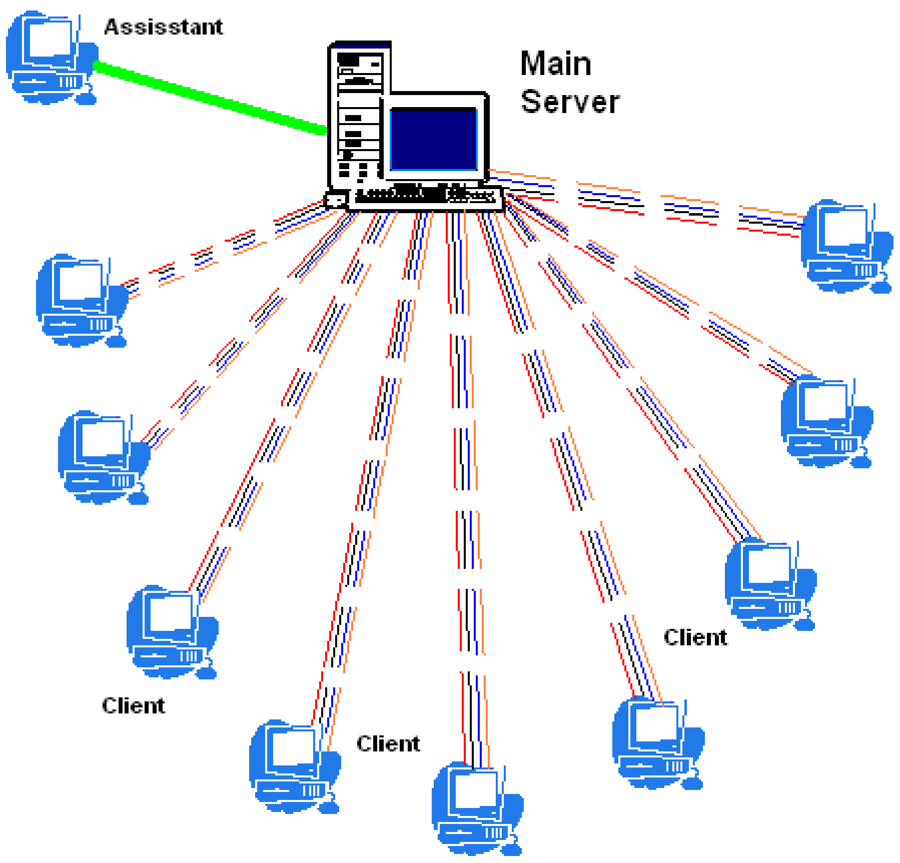


Figure 6.10: Assistant

We can see that even as other Clients come and go, we have this constant Client, the assistant, who's only job is to do any type of work we assign it to do.

Chapter 7

Tests

As discussed in Chapter 1, we test our system using the evaluation of e to high precision and see what type of results we produce. We test the System with different ways of breaking up the problem and with a different number of Clients. We also show that the System handles disconnection of Clients and integrates the connection of new Clients into the current Problem Solution Cycle.

We are, of course, interested in the System performance as well as the performance of solving the problem by distributing parts of it across different machines and obtaining a final result. What will be important to note is the Speed up of this process over the process of solving the Problem on a Single machine.

A very important thing to note here is that the most important aspect of our research is to develop a toolkit which can easily be customized to evaluate different types of series. The Clients which are Java Applets actually run much slower than the standard application via the main () method. Even doing simple I/O slows down the Applet considerably. This is because the Applet is constrained within the browser's environment, the memory is not allocated in the same way as a standard application, and above all, the Client as well as the Server are both GUI programs; they are running multiple threads for the GUI, as well as for communication with each other. We want users to also see what is going on within the System, and the developer to see the progress of the test. All of this involves more computation time that is taken by these parts and less of the CPU time is left available for the actual computation.

Since we want more and more precision in our results, we will be testing with more and more digits of accuracy, which means the number of terms in the Series, given Section 1.1 will increase and therefore the overall work required will also increase. It is important to note that since division of very large rational integers is a complex and time consuming task, in all fairness, we time the computations without doing the final division P/Q.

In the following sections, we will first state the parameters of the test, a brief discussion on why the test is important and how the System behaved in response to the parameters and the conditions. The tests were conducted with the Server and one Client running on an Intel®Pentium^{TM}4 running at 3124MHz with 512MB of Random Access Memory and another Client running on a Pentium®IV 2.6GHz laptop. Most non essential processes were killed before the tests were conducted. The Server machine also runs a web server to host the Applet. The Applet was loaded via Internet Explorer browser. Using only 2 computers, where we are not extracting all the parallelisms that we can potentially have, we hope to achieve speeds closer to the Single Machine solution, since results are slower as previously mentioned.

The common factor in all these tests is that there is a buffer of 5 seconds between sending the *Lock* message and receiving a *Locked* message. This means that before any computation starts, the 5 seconds are spent for locks and therefore contribute a whole 5 seconds to the total Processing time.

We have devised the tests so that we can be very fair to the standalone application as well as the distributed application. In all the tests, the standalone result is also obtained by splitting the problem into smaller chunks. Each Chunk's result is combined into the overall result as soon as it is available. In the Client-Server case, the Server starts joining the results as soon as they arrive. Since the Applet runs slower as mentioned before, it is not surprising when the Chunk time for Applet is almost double the Chunk time in the Application. What is important is that if we actually have many *similar* machines at our disposal, the parallelism we would obtain would guarantee us that the distributed computation would perform much faster than its standalone counterpart.

The mode of computation we use for these tests is to combine the chunks as the results arrive. Using this mode, since we have only access to 2 fast machines, we are running the Server and Client on the same machine. This means that this Client is sharing its time with the Server and therefore does not even do about one third of the total number of Chunks. The second Client solves most of the Chunks, so we lose some time this way as well. The mode can be modified to only combine the Chunks at the end, very easily.

It is also important to note that the times shown here are wall clock times and not CPU times. This means we are measuring the total time from the time that the instance for the Problem is created to the time we have combined all the chunks together (since we are not doing final division).

7.1 Test 1

Parameter	**Value**
Number of Digits	100,000
Number of Chunks	4
Disconnect any Clients	No

Table 7.1: Parameters for Test 1

Please note that number of Chunks is not the same as the number of Clients (2 in all the tests).

Parameter	**Value**
Total Communication Time	63 ms
Total Processing Time	15,000 ms
Chunk 0 Time	766 ms
Chunk 1 Time	681 ms
Chunk 2 Time	891 ms
Chunk 3 Time	4,381 ms
Combination Time	9,297 ms
Time to Evaluate on a Single Machine	12,437 ms

Table 7.2: Results for Test 1

From this test we can see that for a small number of digits, the overhead of using the Applet contributes the most time to the total time. The Chunk Combination Time for both is almost the same.

7.2 Test 2

Parameter	**Value**
Number of Digits	100,000
Number of Chunks	10
Disconnect any Clients	No

Table 7.3: Parameters for Test 2

We are increasing the number of Chunks to see how the System behaves.

Parameter	**Value**
Total Communication Time	111 ms
Total Time	18,265 ms
Avg Chunk Time	200 ms
Combination Time	12,984 ms
Time to Evaluate on a Single Machine	13,719 ms

Table 7.4: Results for Test 2

Since the size of the Chunks has increased, the average Chunk Time has decreased significantly, but the combination time has increase, which contributes to the total time. Keeping in mind the locking time, the total time here is just the lock time and the combination time, as if Chunk computation took no time at all.

7.3 Test 3

Parameter	**Value**
Number of Digits	500,000
Number of Chunks	20
Disconnect any Clients	No

Table 7.5: Parameters for Test 3

We increase the digits as well as the number of Chunks. If the number of Chunks was only 2, this test would take a very long time, as compared to running with 20 Chunks.

Parameter	**Value**
Total Communication Time	100 ms
Total Time	119,907 ms
Avg. Chunk time	1,011 ms
Combination Time	114,297 ms
Single Machine Time	126,766 ms

Table 7.6: Results for Test 3

We have improved our combination algorithm to take advantage of more parallelism, due to which our combination time (which is included in the total time) has improved and our System is now faster than the Single Machine Time.

7.4 Test 4

Parameter	**Value**
Number of Digits	1,000,000
Number of Chunks	20
Disconnect any Clients	No

Table 7.7: Parameters for Test 4

We increase the digits as well as the number of Chunks.

Parameter	**Value**
Total Communication Time	100 ms
Total Time	275,875 ms
Chunk Combine Time	268,734 ms
Single Machine Time	323,329 ms

Table 7.8: Results for Test 4

In this test, with more digits of accuracy, we see a catch-up of the Single machine by our System. We will see in the coming results that our System results are not very different from the Single machine. If we have more computers available, the gap between the two will be significant.

7.5 Test 5

Parameter	**Value**
Number of Digits	1,000,000
Number of Chunks	30
Disconnect any Clients	No

Table 7.9: Parameters for Test 5

We further increase the number of digits to see how the system behaves.

Parameter	**Value**
Total Communication Time	1000 ms
Total Time	365,641 ms
Chunk Combine Time	359,766 ms
Single Machine Time	384,400 ms

Table 7.10: Results for Test 5

We see in this test that our system performs faster than the Single machine. The difference is not very significant, but at least we are running on par with the Single Machine Time, even though we are just using 2 machines for the whole test.

7.6 Test 6

Parameter	**Value**
Number of Digits	2,000,000
Number of Chunks	30
Disconnect any Clients	No

Table 7.11: Parameters for Test 6

We increase the digits as well as the number of Chunks.

Parameter	**Value**
Total Communication Time	1563 ms
Total Time	882,875 ms
Chunk Combine Time	873,766 ms
Single Machine Time	887,968 ms

Table 7.12: Results for Test 6

We see that our results are a little faster than the single machine, due to the reasons specified previously.

7.7 Test 7

In this test, we will compute e to 3 million digits. This is because it will take a long time for this to compute, especially if we keep the number of chunks smaller, hence making each Chunk larger. This will allow us to test disconnection of Clients and connections from new Clients to the System.

Parameter	**Value**
Number of Digits	3,000,000
Number of Chunks	5
Disconnect any Clients	Yes

Table 7.13: Parameters for Test 7

We begin the testing by running the Server and 2 Clients normally. We will disconnect a Client after the data has been sent to the Client and the Client is busy. We will also connect a completely new Client to the System. It will also be tested if the Server reacts as expected when a Client connects at a time when there are no more Chunks to evaluate. Finally, we will let the System run and check if the correct output is received.

Parameter	**Value**
Total Communication Time	900 ms
Total Time	1,123,302 ms
Avg. Chunk time	280,656 ms

Table 7.14: Results for Test 7

The System passed all the required tests, producing the desired behaviour. When a Client disconnected from the System, it was marked as KILLED and subsequently removed from the System. When a new Client connected the System, it was marked as IDLE and given a Chunk to evaluate. When a Client connected to the System and there were no Chunks to be evaluated, it remained in the IDLE state. After all the Chunks are evaluated, the System produced the correct final result.

Chapter 8

Conclusion

From the previous chapter, we see a mixture of results. The timings are sometimes better of our System and sometimes it makes sense to do the computation on only one machine. There are other interesting results as well, which have to do with using Java for solving these types of problems. Let us study these results by dividing them into 2 sections.

8.1 Java as the language for implementation

Currently, Applets are the only applications that can be run inside a browser without any user input, authentication or permission. We saw that this, however, does not open the complete potential of what Java Applets can achieve, but is sufficient for our work. Another great benefit of Applets over conventional non Java applications is that every time the Applet is loaded, the latest version of the compiled code is available for the user. This means that if on the Server side, the code has been changed, updated or removed, the same changes take effect within the Client session, and we can keep changing the types of problems that we use the System to evaluate. However, with all the good benefits that Applets have to offer, we see that Applets run very slow even when doing just computations. This was because there are multiple threads running within the Applet and the computation thread has to share its time with every other thread. We also have to keep in mind that a non Java solution would mean that the Clients would have to download the binary executable code from the website and run it locally. This would yield better control of the application on the Client side, but we would forfeit any benefits of using Java Applets.

We also saw a comparison of Integer Libraries, BigInteger which is built into Java and LargeInteger, a FFT based implementation by R. Howell [8]. BigInt was the C++ implementation done by Xavier Gourdon [6]. We saw that where BigInteger lacked in speed, LargeInteger was a faster solution and vice versa. We suggested a hybrid solution where we would do some computations involving small operands using BigInteger library and perform operations on larger operands using LargeInteger library. The C++ results were provided to see how we compare against a C++ implementation.

In order to have a Java based solution, a web server needs to be run, wherever the System Server resides. This web server will allow Clients to connect to the System via the http protocol and download the Applet in their browser, to run the code. The latest version of the Java Plug-in (or Java Virtual Machine) must be installed on the Client machine; otherwise the Applet will fail to load. The Java Virtual Machine, especially on older machines, takes a long time to initialize and load Applets. But in a large group of machines, this should not pose a very significant problem.

8.2 Distributed Computation of Hypergeometric Series

The Application runs faster than the Applet in almost half the time, which can have an effect on the System. We have to a find a point of optimality in terms of number of Chunks to split the problem into, so that we can minimize the total time. As we saw in section 6.9, it is very easy to make changes to the classes in order to choose a suitable size of the sub-problem. Binary splitting [6, 7] is a popular algorithm for evaluating hypergeoemetric series, we can easily setup the system to use binary splitting instead of the current algorithms to evaluate the series.

In an unpublished paper by Cheng et al. [2], more efficient solutions for evaluating Hypergeometric Series are discussed and it is shown that by using modular arithmetic and Rational Number Reconstruction of images of S(N), the series can be evaluated faster and more efficiently than using Binary Splitting. This is obtained by choosing an appropriate modulus and doing all the computation modulus that prime. This result is important to us because we can incorporate their approach into our System by making changes in the Problem class in the way that the problem is evaluated at the Client and how it is combined together at the end.

We have shown that our solution is very scalable, since we can have any number of Clients connected and we can keep increasing the size of the Problem and still find a solution very fast. We have shown reliability in the System because we have a numeric answer at the end which we can compare against systems like Maple, Matlab or use the techniques described in the verification section.

Chapter 9

Future Work

What we have implemented and presented here is a very limited and small version of what a complete distributed System can offer. There are several avenues that still remain unexplored.

Since we have found a lot of promise in using Java Applets, we can explore the possibilities presented in Chapter 2. If we can sign our Applets through a trusted third party, the chances that a user would trust our Client as well increase significantly. This means that we can make the Client more complex. This can be accomplished by the idea of the Super Node as mentioned before, in which case the Client can also act like a Server and distributed chunks to other Clients, which will be forwarded by the main Server.

By signing Applets and making them trusted, we would have some access to the Client's local file system. The benefit would be that we can save partial results to a file within the Client's file system. This of course comes in handy when a Client is restarted by the user, which can check if any partial results are available and continues computation from that point on, rather than starting from the very beginning. Another approach would be to try and send partial results to the Server and indicating that these are partial results, so that the Server can make use of these results when combining all the chunks to get the Result. It is also possible, with some code changes to the Client, to save the State of the Client. This is similar to saving partial results, but this also incorporates any other data structures that might be in use. This is particularly useful if a Client crashes or if the user decides to shut down the Client.

The Server would need to be modified in order to incorporate changes within the Client. If partial results are going to be saved, then it would be beneficial to the Server to write all the partial results to a file and keep track of this file at all times. This once again is similar to saving the state of the Server, but is only a small portion of the State. The Server can also keep a track of any Clients by making sure that it can reconnect to the Clients in case the Server is restarted. This will decrease any dependency on the user to restart the Client to establish a connection with the Server.

We already saw the approach of using an Assistant, which would allow the Server to be free while the Assistant handles any work, such as solving any remaining Chunks in case there are no Clients connected, or doing the final combination of Chunks. We can also have the Assistant combine partial Chunks, while the Server keeps track of Clients and manages them.

Any other improvements to make the code robust would be a great avenue to explore. We have shown that using Java Applets makes things very easy for general users, if we can do anything more to improve their experience in taking part in solving mathematical problems, it is worth the effort.

Bibliography

[1] H. Cheng and E. V. Zima. *On accelerated methods to evaluate sums of products of rational numbers.* In Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, pages 54-61, 2000.

[2] H. Cheng, B. Gergel, E. Kim, E. V. Zima. *Space-Efficient Evaluation of Hypergoemetric Series.* Unpublished

[3] Bram Cohen. *BitTorrent Network, File sharing via torrents.* At http://www.bitconjurer.org/BitTorrent

[4] Distributed.net. *Node Zero.* At http://www.distributed.net

[5] K. Fritsche, J. Power, J. Waldron. *A Java distributed computation library.* Proc. 2nd International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT2001), pp. 236-243, Taipei, Taiwan, July 2001.

[6] X. Gourdon and P. Sebah. *Numbers, constants and computation.* http://numbers.computation.free.fre/Constants/constants.html

[7] B. Haible and T. Papanikolaou. *Fast multiprecision evaluation of series of rational numbers.* Technical Report TI-97/7, University of Darmstadt, 1997.

[8] R. Howell. *Java LargeInteger Library.* At http://www.cis.ksu.edu/ howell/

[9] Napster LLC. *Napster Online.* At http://www.napster.com

[10] Colin Percival. *PiHex.* At http://www.cecm.sfu.ca/projects/pihex

[11] RSA Security Inc. At http://www.rsasecurity.com

[12] SETI@Home. *Search for Extra Terrestrial Intelligence.* At http://setiathome.ssl.berkeley.edu

[13] Sharman Networks Ltd. *Kazaa.* At http://www.kazaa.com

[14] Sun Microsystems. *Java Technology.* At http://java.sun.com

[15] J. Wan. *CS 775 Lecture Notes.* University of Waterloo, Waterloo, Ontario, Canada

[16] G. Woltman. *Great Internet Mersenne Prime Search.* At http://www.mersenne.org