Technical University of Crete

"Electronic & Computer Engineer"

MA Thesis

System Development under Grid Environment

Author:
Pavlov G. Nicola

February 2008

Contents

List of Figures

List of Tables

Abstract

This thesis studies in depth technologies lying under *Grid Computing.* Grid Computing allows people to share computing power, databases and other on-line tools securely across institutions without sacrificing local autonomy. Our involvement in Grid Computing related projects during the last 2-3 years has brought us up against many challenges. One of these challenges had to do with maximization of GridFTP throughput, which is the most used data transfer protocol adopted by all versions of Grid middleware. GridFTP has its origins to the well-known FTP protocol; it additionally provides secure as well as more efficient data transfers between grid nodes thanks to a number of user-tuneable parameters. We refer specifically to the number of TCP parallel streams and the TCP socket buffer size. In the meantime, we have run many GridFTP sessions through emulated network conditions. Based on the generated results, we have produced and analyzed the best fitting model for the throughput among various mathematical and statistical models. The models studied were: Linear Regression, Least Mean Square Error and Non-Linear (exponential model). For each model, we have used the following parameters as input: bottleneck bandwidth, RTT, number of TCP parallel streams and sender TCP socket buffer size. In conclusion, we have seen that an active probing tool which could actively estimate the network conditions in terms of bottleneck bandwidth and RTT could benefit GridFTP. The question to be addressed was "which method or technique can *reliably* and *actively* estimate the network conditions, i.e. RTT and bottleneck bandwidth?". We in this thesis, have exploited also this matter.

Nowadays, a single computer, a cluster of standard computers or even a special-purpose supercomputer is not enough for the calculations scientists really want to do. Although, computers are improving incredibly fast, they still do not keep up with the increased demands of the scientists. Due to that, it's very difficult, very expensive and sometimes impossible to achieve certain scientific goals with current computer technology. Grid Computing was initially introduced to overcome the drawbacks mentioned before to become today a promisor technology in other fields like industry and commerce. We, in this thesis, examine the ability of Grid Computing in solving heavy-computational problems. We tried to grid-enable an application based on MATLAB that deal with *Children's Epilepsy.* Specifically, we studied the application running in its default state and try to parallelize it in order to run under the grid-testbed created at the Technical University of Crete. Furthermore, the application has been tested at the University of Plymouth.

Grid Computing is implemented via what is called a *middleware.* Globus Toolkit has emerged as the de facto standard middleware for Grids. It constitutes of open-source code that has been developed by the Global Research Community as means to implement the principles of Grid Computing. Unfortunately though, in order to run any middleware and exploit its various features and capabilities, someone has to undergo the whole process of installing the middleware and its various components. All this done using command prompt which first takes time and second putting every user to comprehend the operating aspect of the middleware which leads in turn, to distance the user from his/her first goals. Not to forget to mention that the creators of heavy-computational applications are mainly scientists and they do not want to add a complicated middleware on their list. They are concerned about the application itself. A mechanism or a software is really essential in order to expose the computing resources and distributed systems to general user communities without forcing them to deal with the complexities of the underlying systems. Web browser-based portal user interfaces provide access to a large variety of resources, services, applications, and tools for private, public, and commercial entities. We, at the Technical University of Crete, did create a Grid portal of our own in order to fulfil our needs. The portal includes API for querying/modifying databases based upon OGSA-DAI, an API for transferring files based upon GridFTP service and an API for submitting jobs based on GRAM.

Acknowledgements

First of all, I would like to thank my family for their everlasting support.

I am grateful to Dr. Ioannis Barbounakis for sharing his time and knowledge in our cooperative work.

I am also grateful to Professor Vasilios Samoladas for being the second supervisor.

At last, I would like to thank my supervisor Professor Michael Zervakis for sharing his experience.

Chapter 1

Introduction

Grid computing can mean different things to different individuals. The grand vision is often presented as an analogy to power grids where users (or electrical appliances) get access to electricity through wall sockets with no care or consideration for where or how the electricity is actually generated. In this view of grid computing, computing becomes pervasive and individual users (or client applications) gain access to computing resources (processors, storage, data, applications, and so on) as needed with little or no knowledge of where those resources are located or what the underlying technologies, hardware, operating system, and so on are [1].

Though this vision of grid computing can capture one's imagination and may indeed someday become a reality, there are many technical, business, political, and social issues that need to be addressed. If we consider this vision as an ultimate goal, there are many smaller steps that need to be taken to achieve it.

Therefore, grid computing can be seen as a journey along a path of integrating various technologies and solutions that move us closer to the final goal. Its key values are in the underlying distributed computing infrastructure technologies that are evolving in support of cross-organizational application and resource sharing and virtualization across technologies, platforms, and organizations. This kind of virtualization is only achievable through the use of open standards. Open standards help ensure that applications can transparently take advantage of whatever appropriate resources can be made available to them. An environment that provides the ability to share and transparently access resources across a distributed and heterogeneous environment not only requires the technology to virtualize certain resources, but also technologies and standards in the areas of scheduling, security, accounting, systems management, and so on.

Early implementations of grid computing have tended to be internal to a particular company or organization. However, cross-organizational grids are also being implemented and will be an important part of computing and business optimization in the future. As Internet connect speed increases though, the difference between having two PCs in the same office, the same building, the same city or the same country shrinks. By developing sophisticated middleware which makes sure that widely distributed resources are used effectively, Grid computing gives the user the impression of shrinking the distances further still. In addition, as the middleware gets more sophisticated, it can deal with the inevitable differences between the types of computers that are being used in a highly distributed system, which are harder to control than within one organization. One of the most popular middleware packages today is Globus [2], and it is essentially a software toolkit for making Grids. With such middleware, the aim is to couple a wide variety of machines together effectively, including supercomputers, storage systems, data sources and special classes of devices such as scientific instruments and visualization devices.

Grid computing involves an evolving set of open standards for Web services and interfaces that make services, or computing resources, available over the Internet.

Very often grid technologies are used on homogeneous clusters, and they can add value on those clusters by assisting, for example, with scheduling or provisioning of the resources in the cluster. The term grid, and its related technologies, applies across this entire spectrum.

One definition of Grid computing, by Ian Foster [3], one of the pioneers who helped coin the term, distinguishes it from other forms of computing such as distributed computing, metacomputing, clutser computing and peer-to-perr computing.

The definition is that Grid must satisfy three criteria:

1. No central administrative control of the computers involved (that eliminates clusters and farms, and also local Grid computing)

2. Use of general-purpose protocols

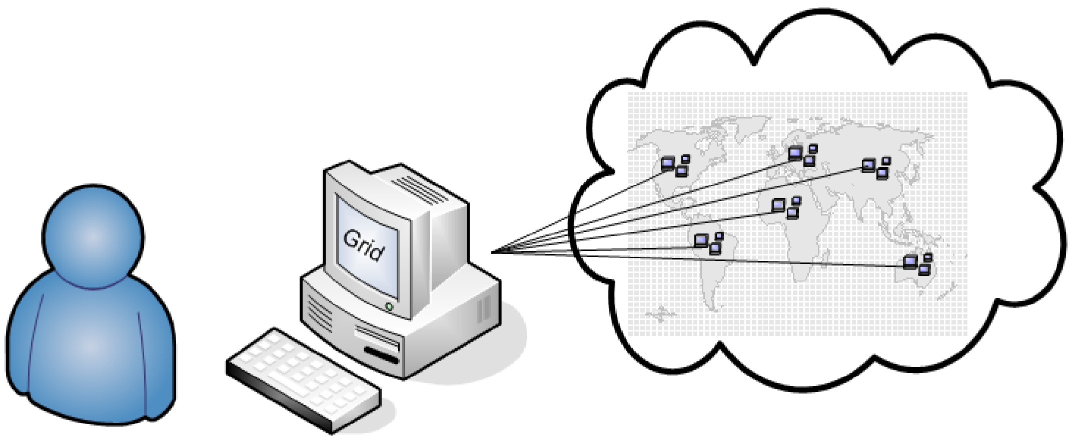


Figure 1.1: *Grid Computing Vision*

3. High quality of service (that eliminates peer-to-peer and means that Grids should not rely on cycle scavenging from individual processors, but rather on load balancing between different independent large resources, such as clusters and local Grids)

When it comes to building Grids, the basic ideas that the software engineers and developers have in mind and spend most of their time and effor working on are described below.

The most important is the **sharing of resources** on a global scale. This is the very essence of the Grid. Then, although it is hardly a novelty, **security** is a critical aspect of the Grid, since there must be a very high level of trust between resource providers and users, who will often never know who each other are. Sharing resources is, fundamentally, in conflict with the ever more conservative security policies being applied at individual computer centers and on individual PCs. So, getting Grid security right is crucial. If the resources can be shared securely, then the Grid really starts to pay off when it can **balance the load** on the resources, so that computers everywhere are used more efficiently, and queues for access to advanced computing resources can be shortened. For this to work, however, communications networks have to ensure that **distance no longer matters** - doing a calculation on the other side of the globe, instead of just next door, should not result in any significant reduction in speed. Finally, underlying much of the worldwide activity on Grids these days is the issue of **open standards**, which are needed in order to make sure that Research and development worldwide can contribute in a constructive way to the development of the Grid, and that industry will be prepared to invest in developing commercial Grid services and infrastructure.

As mentioned before, Globus Tooklit is the most well-known open-source middleware to make Grids. It contains several components such as Grid Security Infrastructure (GSI), GridFTP, Grid Resource Allocation Management (GRAM) and others.

In this thesis, all our work was based on the Globus Toolkit. Specifically we tried to exploit some of the features mentioned above beyond their simple use.

GridFTP is the Globus component that is responsible for transferring files in a Grid environment. In this thesis, we made an extensive research on the GridFTP service. We studied its throughput behaviour based on network and non-network parameters under different emulated network conditions. We then tried to find the best fitting model based on mathematical and statistical regression models. As a conclusion from the research on GridFTP throughput, we saw that a tool that can measure activley network conditions (RTT and bandwidth) would help in maximizing the GridFTP throughput dynamically. Reasearch in this field has been held for some years now. Packet pair/train technique is the most well-known methodology adopted in this field. We tested some "ready" tools that are based on this technique. These tools seemed to work only for low-scale networks. That led us to create a tool to work properly for large-scale networks. In this thesis, we present our own approach and compare it with other tools.

GRAM is the Globus component that is responsible for submitting jobs into a Grid environment. In this thesis, we tried to grid-enable an application that is created using MATALB. The application initially ran in a serial way. Our goal was to examine the application and its ability to run in a parallel way. We submitted the application using GRAM, into the grid testbed created at the Technical University of Crete. Moreover, GRAM was also used to test the application's integrity with the University of Plymouth.

The structure of this thesis is formulated as follows:

Chapter 1 is this introduction.

Chapter 2 presents the GridFTP service and its various features. GridFTP is the most known and used protocol to transfer files through grid enviornments and is adopted by all grid middleware. GridFTP throughput can be maximized using tunable features such as number of TCP parallel streams and TCP socket buffer size. In this chapter, we examine these two features and try to model the GridFTP throughput based on five different network and non-network parameters. These parameters are: File-size, RTT, bandwidth, number of TCP parallel streams and TCP socket buffer size. We also present the best fitting model for the GridFTP throughput.

Chapter 3 introduces a bandwidth estimation approach. Based on our work in chapter 2, we concluded that a tool that could actively measure the network conditions (i.e. RTT and bandwidth) could help in maximizing in an automatic way the GridFTP throughout. We adopted the most well-known technique used in the field, the packet train/pair technique. In this chapter, we introduce our approach in measuring the bandwidth based on Java Networking.

Chapter 4 presents an example of Grid applications. Grid computing was initially introduced to meet the requirements of *heavy* computations. Starting from an application written in MATLAB and ending up with an application that could run under grid environments and thus run much more effectively and quickly was our goal. In this chapter, we introduce the steps that took place and the difficulties we did face.

Chapter 5 gathers all the conclusions met so far.

Chapter 2

Maximizing GridFTP Throughput

2.1 Introduction

The majority of internet users have been utilizing physical network connections of the order of at least 100 Mbps. Recently however, there is a high demand for 1000-Mbps Ethernet connections.

The need for transferring large-sized files is continuously increasing the demand for higher throughput. The assumption that the network connection speed is the only parameter that affects the throughput has been proven inadequate. TCP/IP [4], which is the underlying network mechanism, is based on the pre-set values of parameters like the TCP socket sender/receiver buffer size and TCP block size. For the average user, all these parameters have been transparently set to a default value by the operating system or the specific applications themselves, leading to semi-dynamic behaviours. The absence of dynamic configuration of such parameters according to the ever-changing network conditions leads to a lower throughput. However, for the professional or high-skilled network user it is an important issue to be able to optimize the TCP/IP layer parameters with regard to the maximum achievable throughput. Grid Computing has been evolved like one of these professional users. It's about an initiative which enables large scale scientific and engineering computation via the connection of multiple distributed computing resources.

In general, large scale scientific and engineering computation requires large file transfers among network computers in minimal time, which are projected to higher throughout needs and make GridFTP [5] the best available choice.

GridFTP uses Transmission Control Protocol (TCP) [6] as its underlying networking mechanism. TCP is a connection-oriented, reliable and window-based protocol unlike connectionless User Datagram Protocol (UDP) [7].

In this chapter we are going to present our research on GridFTP service. In details, an overview of the GridFTP protocol, its various features and capabilities, is going to be presented. The GridFTP underlying network mechanism TCP is going to be referred. How TCP multiple parallel streams and TCP socket buffer size affect GridFTP throughput, is going to be discussed. The latter will be achieved theoretically and experimentally. Experiments have been conducted locally at the Technical University of Crete (TUC) using a network emulator and also remotely with the University of Plymouth (UoP). Results will be studied over various network and non-network parameters by applying three different mathematical and statistical models. Finally, the best fitting model is going to be presented.

2.2 GridFTP Overview

GridFTP is a common data transfer and access protocol that provides secure, efficient data transfer in Grid environments. This protocol is based on the well-known, standard FTP [9] protocol. The FTP protocol has been chosen because it is the most commonly used protocol for data transfer on the Internet. GridFTP extends FTP protocol with the following features:

- **Grid Security Infrastructure (GSI)** [10]: Robust and flexible authentication, integrity, and confidentiality features are critical when transferring or accessing files. GridFTP supports GSI authentication with user controlled setting of various levels of data integrity and/or confidentiality.

- **Third-party control of data transfer**: Third-party transfers are necessary to manage large datasets for large distributed communities. The GridFTP separates control and data channels, enabling third-party transfers, i.e. the transfer of data between two end hosts, mediated by a third host.

- **Parallel data transfer**: Improves aggregate bandwidth relative to that achieved by a single TCP stream.

- **Stripped data transfer**: Partitioning data across multiple servers can further improve aggregate bandwidth.

- **Partial file transfer**: Some applications can benefit from transferring segments of files rather than complete files: e.g. analyses that require access to subsets of massive object-oriented database files. GridFTP allows transfer of the remainder of a file starting at a specified offset.
- **Manual control of TCP buffer sizes**: This is a critical parameter for achieving maximum bandwidth with TCP/IP.
- **Reliable data transfer**: Reliable transfer is important for many applications that manage data. Fault recovery methods for handling transient network failures, server outages, etc., are needed.

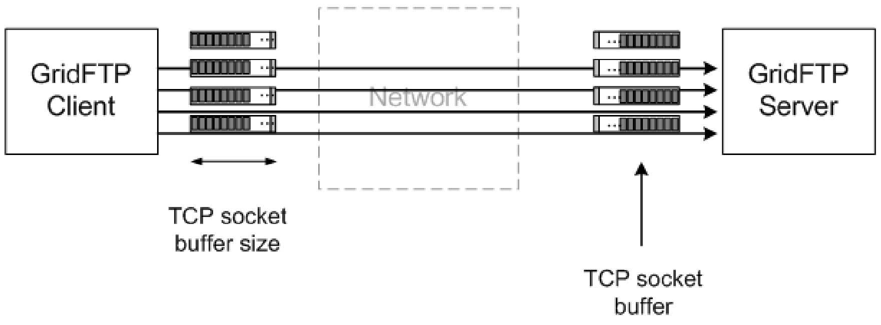


Figure 2.1: *Parallel Data Transfer*

The GridFTP protocol specifies two modes of data transfer:

- Stream Mode (Mode S)
- Extended Mode (Mode E)

Stream Mode is implemented in FTP servers where bytes flow in order, over a single TCP connection. There are no advanced features in this mode. GridFTP defaults to this mode so it is compatible with normal FTP servers.

On the other hand, *Extended Block Mode* constitutes an extension of the Stream Mode where data can be sent over the data channel in blocks. Each block consists of 8-flag bits, a 64 bit integer indicating the offset from the start of the transfer, and a 64 bit integer indicating the length of the block in bytes followed by a payload of length bytes as shown in Figure 2.3. Because the offset and length are provided, out-of-order transmission is acceptable, i.e. the 10th block could arrive before the 9th because you know explicitly

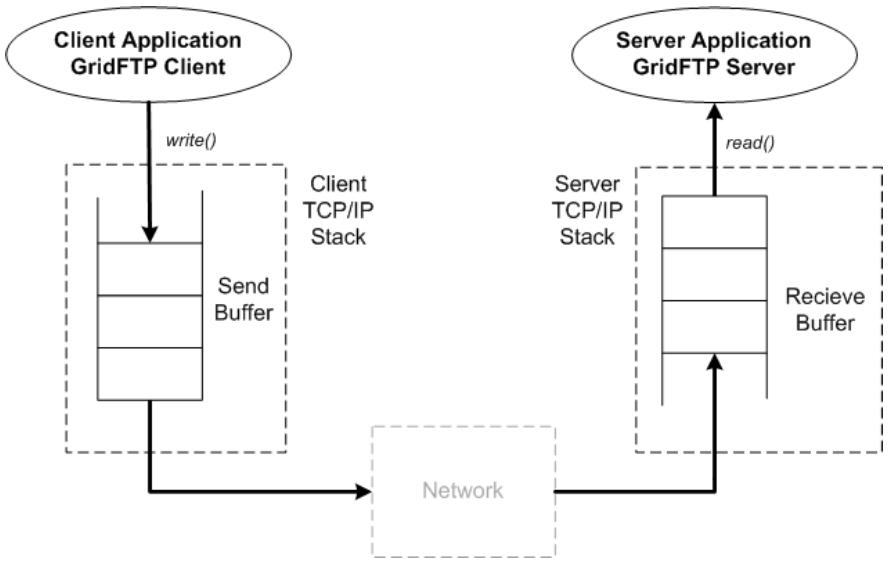


Figure 2.2: *Manual control of TCP buffer sizes*

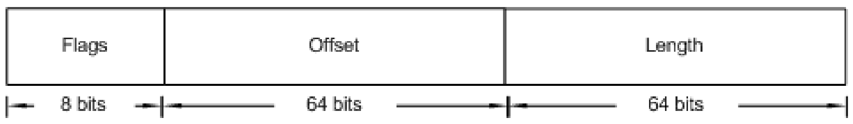


Figure 2.3: *GridFTP Extended Mode*

where it belongs. This also enables parallelism and striping.

As we mentioned before, GridFTP uses TCP as its underline networking mechanism. An overview of the TCP protocol is presented in the next section.

2.2.1 Transmission Control Protocol (TCP)

A software developer designing a network application can choose whether to use TCP or UDP as a transport protocol. As mentioned before, GridFTP is based on TCP.

TCP is a connection-oriented protocol. It processes data in streams. In other words, TCP can accept data, a byte at a time rather than as a preformatted block. It enables re-sequencing, i.e. restore the data original order. The latter enables using TCP multiple streams which is going to be fully discussed later on. It is based on flow control which would not outrun or overrun the destination machines capability to receive the data.

TCP uses what is called a "*congestion window*". A congestion window is used to determine how many packets it can send at one time. Larger congestion window means higher throughput. However, when a client sends data faster than either the network or the host on the other end, we say that we have a network congestion situation which reduces the throughput to very low levels. There are some other factors that could degrade the TCP performance. Here are some: 1) bad Network Interface Card cables, 2) poor application design and 3) too small TCP window size.

TCP uses a number of mechanisms to achieve high performance and avoid congesting the network. These mechanisms include the use of a sliding window, the slow-start algorithm, the congestion avoidance algorithm, the fast retransmit and fast recovery algorithms, and more.

Specifically, the TCP "*slow-start*" and "*congestion avoidance*" are the responsible algorithms for determining the size of the congestion window. The congestion window is directly proportional to the amount of buffer space that the kernel allocates for each socket. The kernel each operating system sets up a specific amount of buffer for each socket. Besides, the buffer size can be adjusted for both send and receive ends of the socket.

TCP was first employed in 1970. Since then, a lot of TCP extensions (versions) have been presented in order to overcome the early TCP protocol limitations. To the best of our knowledge, the more representative extensions are the TCP Reno, TCP new Reno and TCP Vegas. The most well-known TCP version is TCP Reno.

Limitations that exist in the late TCP versions are: 1) Inability to support multiple-stream, parallel transfers. TCP has been developed as a one-stream transfer protocol. 2) TCP block-size boundaries are not defined. The user application is responsible to set these boundaries. 3) TCP was originally invented for point-to-point connections. It cannot be used for broadcast or multicast transmission. 4) TCP uses a congestion window to define how many packets it can send at one time as shown in Figure 1. Larger congestion window means higher throughput.

GridFTP protocol has been designed to overcome the TCP limitations mentioned above. It achieves this goal by integrating some additional features such as manual control of TCP socket buffer size and parallel TCP connections. However, there is still a remaining problem in that there is no rule of thumb for choosing optimal values for these parameters.

2.2.2 TCP socket buffer size

Current operating systems (Microsoft & Linux) offer a good environment for TCP tuning configuration which in turn helps achieve higher performance. For example, an increase in the TCP receiver socket buffer size leads to higher throughput. To achieve maximum throughput, TCP sender/receiver socket buffer sizes must be optimal. If the TCP sender socket buffer size is too small, the TCP congestion window will never fully open up and if it is too large the sender may overrun the receiver which will cause the TCP window to shut down.

Linux SuSE (the Linux version we used as the underline OS for all the tests, simulations, etc.) default TCP send/receive buffer is 128 KB. By preserving default parameters, stable transfer operations are provided but at the cost of the transfer rates. In other words, a small percentage of the bandwidth will be consumed.

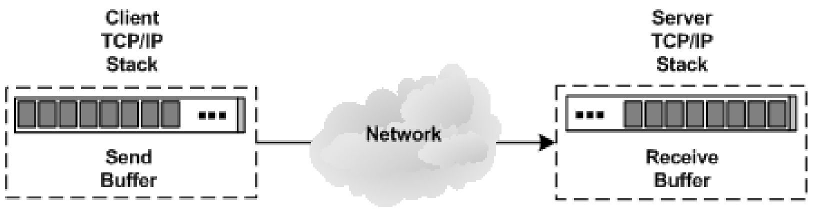


Figure 2.4: *TCP Socket Buffer Size*

The TCP sender/receiver buffer sizes are chosen depending on what is called the Bandwidth Delay Product (BDP). BDP is the amount of data that can be in transit in the network at any time. Formally, it is equal to the product of the bottleneck link bandwidth and the Round Trip Time (RTT, also known as latency or delay) of the network connection form end-to-end. Today, BDP has become more important to the TCP protocol compared with the past due to the implementation of higher speed networks.

In addition to the above, GridFTP provide parallelism, i.e. using multiple TCP streams (connections) in transmission.

2.2.3 TCP parallel streams

TCP probes the available bandwidth of a connection by continuously increasing the window size until a packet loss is detected. When the latter happens, the window size is cut in half and starts recurring again. For large WAN connections, i.e. for large BDPs, the recurring stage requires a longer time. During this stage, less available bandwidth will be used. For the performance not to be degraded, a large bottleneck link is required which, in turn, requires large TCP socket buffer sizes on all the intervening routers.

In order to improve the latter situation, TCP parallel streams can be used. This technique works as follows: data is divided into N portions and each portion is transferred with a separate TCP connection. The effect of N parallel streams is to reduce the Bandwidth Delay Product experienced by a single stream by a factor of N because they all share the single-stream bandwidth. When competing with connections over a congested link, each of the parallel streams will be less likely to be selected for having their packets dropped, and therefore the aggregate amount of potential bandwidth which must go through premature congestion avoidance or slow start is reduced. It should be noted, however, that if the bandwidth is limited by small router buffers in the path, all the streams are likely to experience packet loss in synchrony (when the buffer fills, arriving packets from each stream are all dropped) and thus gain little advantage over a single stream.

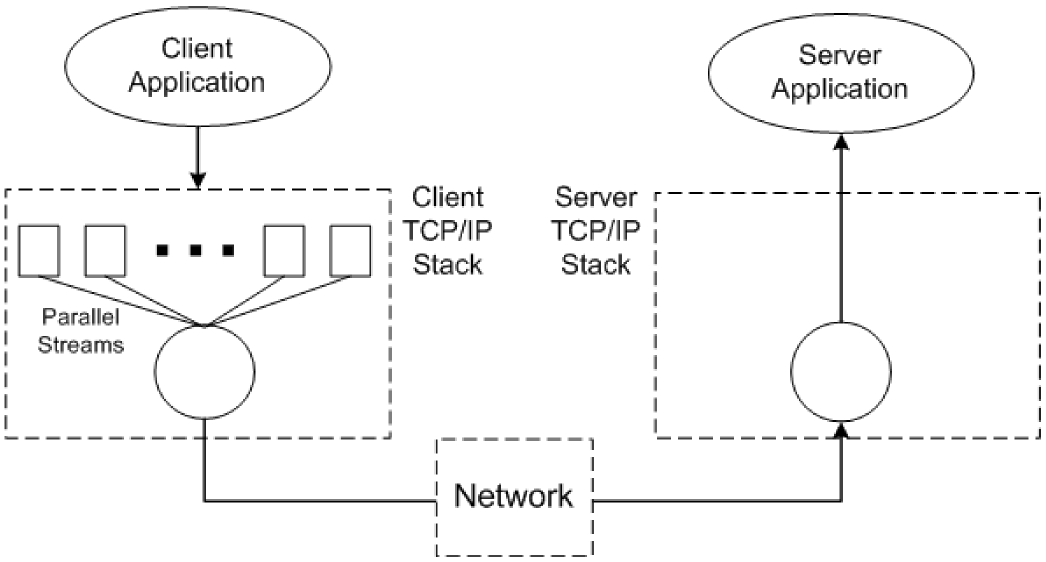


Figure 2.5: *TCP Parallel Streams*

Experience has shown that parallel streams can dramatically improve application throughput and can also be a useful technique for cases where you dont have root access to a host in order to increase its maximum TCP buffer size. However, parallel streams can drastically reduce throughput if the sending host is much faster than the receiving host.

2.3 Simulation

In the following section we are going to present the initial tests that took place locally and remotely.

globus-url-copy Command-line

The basic procedure for using GridFTP is running the `globus-url-copy` command [11].

For implicitly, just run the following:

```
user:/> globus-url-copy source_url destination_url
```

Note: Authentication and authorization processes should be met in order to run successfully the above command.

In the default mode, `globus-url-copy` uses a single TCP stream. To choose TCP parallel streams, the option `-p` is added to the `globus-url-copy` command. But the question that arises up is how do one chooses a value for the `-p` option?

For most instances, using 4 streams is a very good rule of thumb. Unfortunately, there is not a good formula for picking an exact answer.

Therefore, we tried to find an answer conducting some experiments (tests).

2.3.1 Initial experiments

The initial experiments involved transferring various files on two phases:

1. Locally (using the Grid testbed at the Technical University of Crete (TUC))
2. Remotely, between TUC and the University of Plymouth (UoP).

Those first tests included only the TCP Parallel streams feature of the GridFTP protocol.

GridFTP Server	**CPU**	**RAM**	**LAN NIC**	**OS**	**Globus Toolkit**
Grid3.tsi.gr	Pentium IV Xeon 2 GHz	512 MB	100Mbps	SuSE Linux 9.2	GT 4.0.1
Grid2.tsi.gr	Pentium II 400 MHz	128 MB	100Mbps	Slackware 10.1	GT 4.0.1

Table 2.1: *PC Specifications at TUC*

Table 2.1 shows the machines used locally at TUC during the tests. The Globus Toolkit available at that time was GT4.0.1 (In the meantime of writing this thesis a GT4.0.5 was available).

GridFTP Server	**CPU**	**RAM**	**LAN NIC**	**OS**	**Globus Toolkit**
Grid3.tsi.gr	Pentium IV Xeon 2 GHz	512 MB	100Mbps	SuSE Linux 9.2	GT 4.0.1
smb2193n2-31814	Pentium IV 3 GHz	1 GB	1Gbit	SuSE Linux 9.2	GT 4.0.0

Table 2.2: *PC Specifications at UoP*

Table 2.2 shows the machines from TUC and UoP used duting the tests.

Figure 2.6 below, shows the transfer rate vs. the parallel streams for different file-sizes (10MB, 100MB and 1GB file-size). The transfers where done locally at TUC. The transfer rate is represented in MB/s. A range from 1 to 512 TCP parallel streams was used initially. We show only some values from this range (1, 4, 8, 10, 16 and 32).

When transferring a 10MB file, the best transfer rate (9.60MB/s) is achieved when `-p` equals four. While using a 100MB file, the best transfer rate (9.95MB/s) is achieved when `-p` equals eight. Note that using four and ten parallel streams roughly achieve the same average value. Using a 1GB

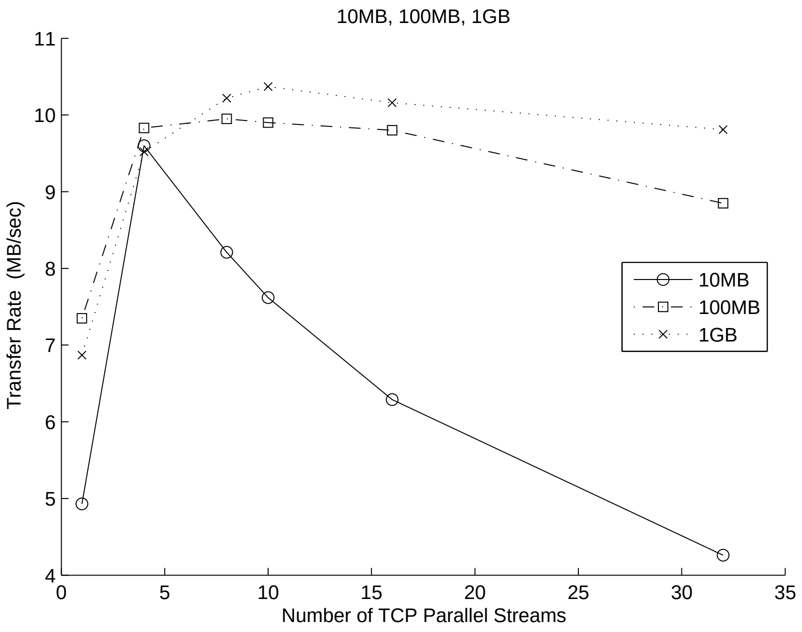


Figure 2.6: *Locally at TUC*

file, the best transfer rate had been achieved (10.37 MB/s) when `-p` has the value ten. 10. In addition to the latter, using eight parallel streams in the transfer can achieve well values (10.22 MB/s).

Observing Figure 2.6, one can originally notice that all the files (10MB, 100MB and 1GB) can be transferred at a very good rate when using four parallel streams.

Looking more precisely, one can figure out the following:

1. For small-size files (1MB - 80MB), its more preferable to use only four parallel streams;

2. For large-size files (100MB -), eight or ten parallel streams can be used for transferring files.

We have to emphasize that these values can differ from machines to machines according to the network bandwidth that underlies them.

Figure 2.7 shows the transfer rate vs. TCP parallel streams using the same file-sizes (10MB, 100MB, 1GB) as the case before. As expected, the transfer rate reduces dramatically when files are transferred remotely. From the graph, the best transfer (3.91MB/s), while using 10MB file, is achieved when sixteen parallel streams are used. That comes in contradiction with the tests done locally, where transfer rate starts to reduce after using ten parallel streams. When 100MB file was used, we can notice that the best transfer rate (6.97MB/s) is achieved also when sixteen parallel streams are used for the transfer. Finally when using 1GB-file for the transfer, we notice that the best transfer rate (7.63MB/s) is achieved when the parallel streams used are thirty two.

Note: The tests took place between 13:00 and 16:00 locally (Greece, +2GMT).

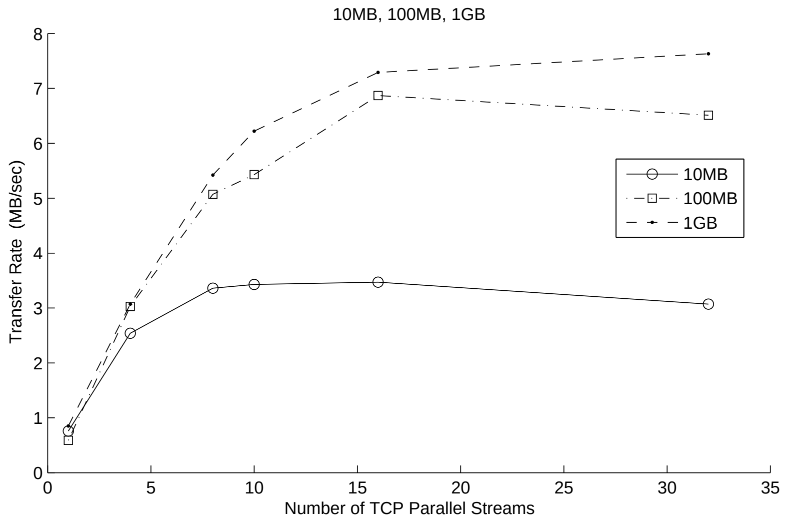


Figure 2.7: *Remotely between TUC and UoP*

Finally, according to the theory and the tests that took place, one can conclude that using parallel streams improves radically the transfer rate bandwidth. In other words, the GridFTP protocol/service takes advantage of the bandwidth thus improving vastly the transfer rate and by that we recommend using multiple parallel streams during transfers.

2.3.2 Further Experiments

In the second phase, in addition to the TCP parallel streams, TCP socket buffer size was examined. The benefit from using an adapted value for the buffer size was discussed in the previous section. We here show the tests taht took place and discuss the results.

In these tests, only remote transfers with UoP have taken place.

Figure 2.8 shows the transfer rate vs. the parallel streams of a 100MB-file for different TCP socket buffer sizes. Using Ethereal Monitoring tool and depending on previous tests with UoP the estimated RTT value between the two universities is $\approx 70ms$. The bandwidth value used at that time was 100Mbps. That gives a BDP value of 875000 bytes. As the TCP socket buffer size values, the following were used: the default TCP socket buffer size of a SuSe kernel (128K), twice BDP and finally BDP multiplied by the number of TCP parallel streams.

As can be seen from Figure 2.8, using modified TCP socket buffer size values improves further the transfer rate. Moreover, Figure 2.8 shows that using very large TCP socket buffer sizes may have negative effects on the transfer rate.

In general, using suitable values for the TCP socket buffer size and for the number of TCP parallel streams the throughput would reach to an optimal value. As mentioned above though, there is no rule of thumb on choosing these values.

2.3.3 Related Work

During the last years, there have been some efforts trying to explore and find some certain formula that could estimate and calculate efficient values for those features [12, 13, 14].

In [12], an extension of the GridFTP protocol is proposed, which determines the required TCP socket buffer size based on the BDP. Particularly, it is based on a one-time measurement of the BDP, i.e. Round Trip Time (RTT) and bottleneck (available) bandwidth. Under real conditions though, the BDP value varies over time. Moreover, other network parameters such as the number of TCP parallel streams and non-network parameters such as file-size are not taken into consideration. Finally, the proposed mechanism

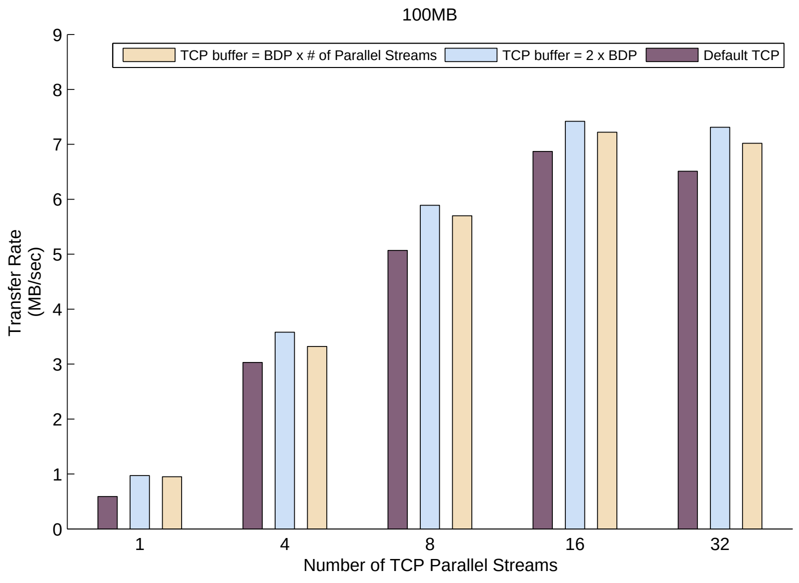


Figure 2.8: *Modified TCP buffer size*

allocates twice of the BDP per TCP connection, which is inefficient regarding memory allocation according to our conclusions.

In [13], a fluid-flow model has been adopted for the TCP control mechanism when applied to large scale IP networks. Here, multiple TCP connections are modelled as independent continuous-time systems and the bottleneck router is modelled as another single continuous-time system. An analysis of the final model is then performed. As a result, the optimal number of TCP connections and the TCP socket buffer size are computed. However, it should be mentioned that the RTT, the packet loss probability and the bottleneck bandwidth should be known in advance. Moreover, only server-to-client transfers are taken into consideration, whereas traffic on the control channel is neglected. Although, they use the TCP socket buffer size equal to BDP in their ns-2 based simulations, they suggest that TCP socket buffer size should be larger than the BDP value.

In [14], an automatic configuration mechanism, based on [13], is proposed. It is unable to apply to third-party transfers since the configuration is fully performed at the client side. This way, only client-to-server transfers should benefit, and hence this mechanism should not be integrated in an updated version of the GridFTP protocol. However, it seems that such a mechanism should be more beneficial to Grid transfer protocols if it was implemented in the middleware.

The GridFTP protocol extension proposed in [12] is based on a one-time measurement of the BDP, i.e. on both RTT (network latency) and the bottleneck (available) bandwidth. Under real conditions though, the BDP value varies over time. Moreover, other network parameters such as number of TCP parallel streams and socket buffer size play an equally important role.

In [13], a certain number of network parameters such as the RTT, the packet loss probability and the bottleneck bandwidth should be known in advance. Moreover, only server-to-client transfers are taken into consideration, whereas traffic on the control channel is neglected. Moreover, ns-2 simulator has been used as its simulation tool. Its concluded that TCP socket buffer size should be larger than the BDP. Although a reference to BDP as inadequate for practical purposes is made, their ns-2 based simulation model is based on BDP.

In [14], the proposed automatic configuration mechanism cannot apply to third-party transfers since the configuration is fully performed at the client side. This way, only client-to-server transfers should benefit, and hence the proposed mechanism should not be integrated in an updated version of the GridFTP protocol. However, it seems that such a mechanism should be more beneficial to Grid transfer protocols if it was implemented in the middleware.

2.3.4 Our Approach

In contrast to the previous works, we have approached the problem of GridFTP throughput maximization in an opposite way. We first started performing a series of experiments over real-world wide area network conditions and then recording the measurements. Afterwards, we experimented with the network throughput behaviour, via a network emulator, under the variation of the following network and non-network parameters: 1) File-size, 2) RTT, 3) Network Bandwidth, 4) Number of parallel streams and 5) TCP sender buffer size. We initially considered BDP as the only throughput-affecting parameter which characterizes the current network conditions. However, we soon discovered that BDP cannot be considered equivalent to the various bottleneck bandwidth and RTT values whose product equals BDP, i.e. the pair (100Mbps, 70ms) has the same BDP value as the pair (1Gbps, 7ms). Through a behavioural analysis, we tried to find the best-fitting model to the network throughput estimator. Through this estimator, each application and hence GridFTP will be able to pre-select (right before the data transfer) the best network parameters (number of TCP parallel streams and buffer size) in terms of maximum throughput.

Unlike [12], our work focuses on multiple TCP streams and TCP sender buffer size. Whereas, [13, 14] also focus on these two parameters, their proposed mechanism has only been tested under ns-2 and is only based upon BDP. Moreover, it cannot be applied to third-party transfers.

In this work, we find the best-fitting model for the network throughput estimator. Through this estimator, each application and hence GridFTP will be able to pre-decide (right before the data transfer) on the best parameters (number of parallel streams and buffer size) in terms of maximum throughput.

Throughput Measurement Setup

In this section, we describe two different network configuration setups to measure the throughput of the GridFTP protocol. In each case, we commit two hosts running the client and the server applications of the GridFTP protocol. In the first setup we have made use of the LANforge network emulator [15] as shown in Figure 2.9, whereas in the second setup, we have used the already established Grid infrastructure between Technical University of Crete (TUC) and University of Plymouth (UoP).

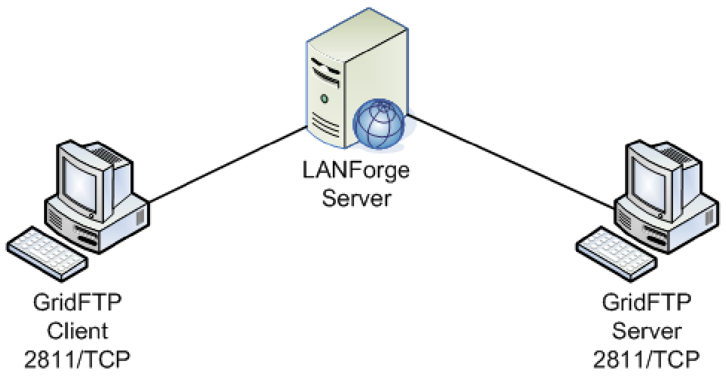


Figure 2.9: *LANForge Topology*

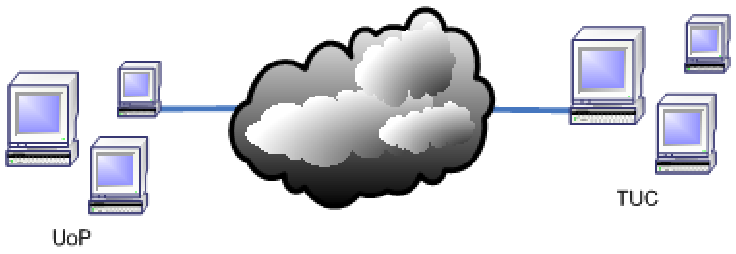


Figure 2.10: *TUC-UoP Grid Testbed*

LANForge is an advanced network emulator which provides a handful of network parameters enough to extensively test the GridFTP protocol locally without the inconvenience of having to coordinate different administrative domains.

Choosing the parameters for LANForge setup

Regarding file-size, we have used 100MB, 1GB and 10GB values. RTT values range from 20ms to 100ms. Network Bandwidth values range from 45Mbps to 155Mbps. As for the number of TCP parallel streams, we have chosen the most common values (8, 16, 32, and 64). Choosing more than 64 parallel streams had no remarkable improvement on the throughput. Finally, we have chosen TCP socket sender buffer size values in the range from the default kernel value 128KB up to 1MB. In every case, we have carefully selected the (RTT, Bandwidth) pair values so that the equivalent BDP value remains below the maximum TCP socket buffer size value being used in our experiments. Moreover, we configured LANForge so that it resembles a real network. We chose a packet-drop frequency of 10^{-3} packets and a maximum jitter value of 4ms at both ends of the WAN link.

Results produced from LANForge setup

In our first set of simulation runs, we wrote down the behaviour of 100MB file-transfers. The throughout reaches a maximum at a specific pair of values for the number of parallel streams and the TCP sender buffer size as shown in Figure 2.11. For all the other pairs, the throughput remains at lower levels. For the cases, we have chosen more than 16 parallel streams, it seems that the extra throughput gain is compensated from the time required to reassemble the packets from these extra streams.

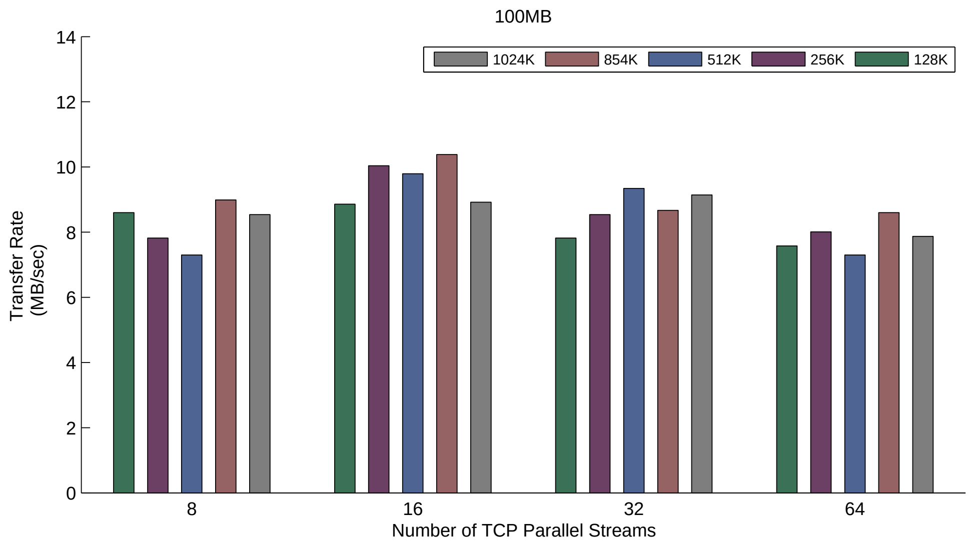


Figure 2.11: *Transfer Rate vs. Parallel streams for a 100MB-file, using different Buffer sizes*

Figure 2.12 summarizes the results from the 1GB file-transfers. In this case, it seems that the TCP sender buffer size has sufficient time to reach its maximum value, which corresponds to the maximum throughput the file transfer can take from the network. Thats why, whatever the number of parallel streams and the TCP sender buffer size used in the simulations, the throughput remains stabilized. This behaviour is further amplified by the 10GB file-transfers (see Figure 2.13).

TUC-UoP Grid testbed

During the establishment of a small Grid testbed between TUC and UoP, we had the chance to run several GridFTP tests under real network conditions. The tests were run at different times of the day and for a period longer than a month in order to have all the network variations possible. We used the Ethereal Monitoring Tool to measure RTT and bottleneck bandwidth. The measured values were 70ms and 50Mbps respectively. They produced a BDP value of 427 KB, more than three times the TCP default sender/receiver buffer size.

Figure 2.14 presents the measurements that were taken. Its worth noting the proximity between the simulated and the measured throughput values

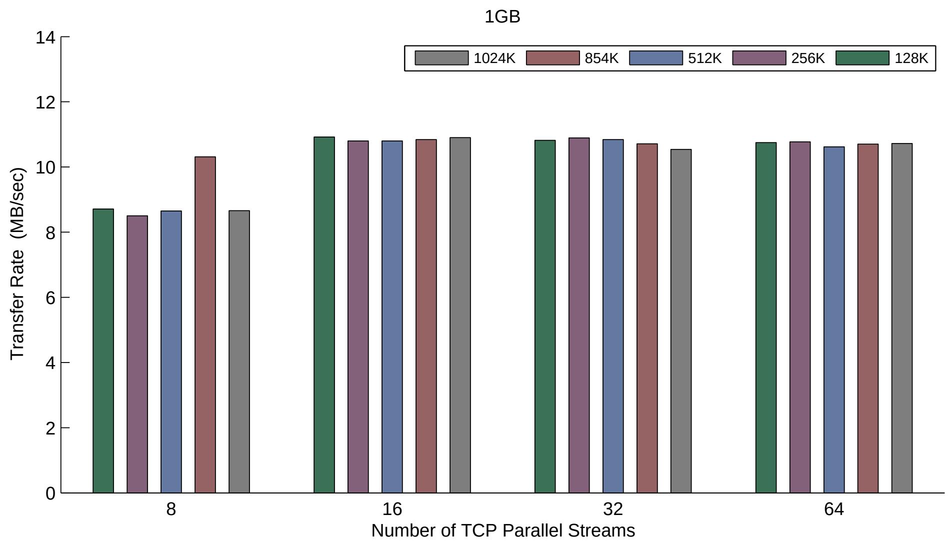


Figure 2.12: *Transfer Rate vs. Parallel streams for a 1GB-file, using different Buffer sizes*

for the case of 100MB file-transfers.

Throughput Modelling

As mentioned before, a fluid-flow model has been adopted by [13]. Applying steady state analysis, the GridFTP goodput is given as a function of the TCP socket buffer size W, the number of parallel TCP connections N, the round-trip time R, the bottleneck link bandwidth B and the packet loss probability p as shown in Equation 2.1:

$$\overline{G^*} \approx min\left(\frac{NW}{R}, \frac{N(1-p)}{2R}\left(-3+\frac{\sqrt{6+21p}}{\sqrt{p}}\right)\right) \tag{2.1}$$

The optimal number of parallel TCP connections is obtained by determining N that maximizes Equation 2.1.

In our approach however, we have already setup a network testbed as described in the previous section. Having all the necessary network measurements at our disposal, we are capable of observing the throughput behaviour in terms of each parameter alone.

We first came to the conclusion that the throughput depends more or less

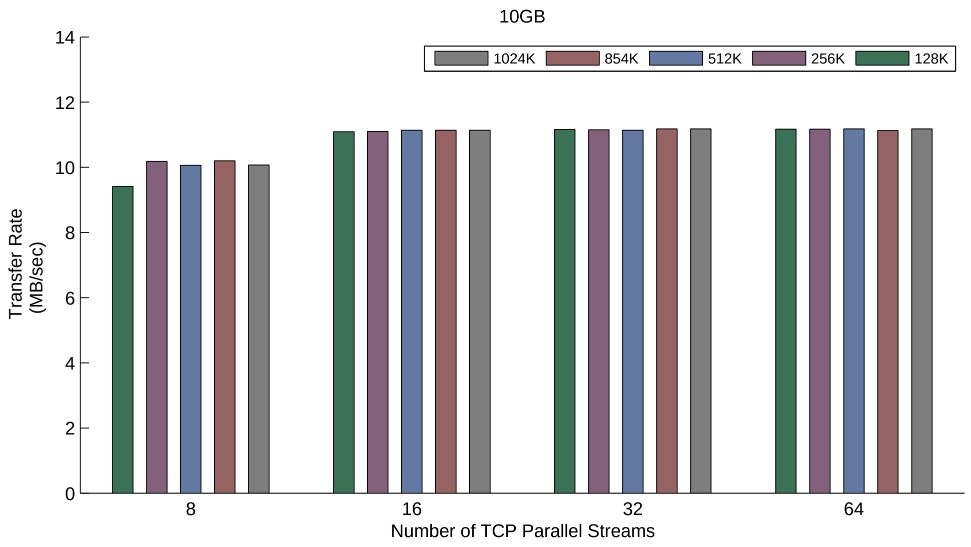


Figure 2.13: *Transfer Rate vs. Parallel streams for a 10GB-file, using different Buffer sizes*

on the five parameters used in our network setup. These parameters are file-size, RTT, Network Bandwidth, number of TCP parallel streams and TCP socket sender buffer size. Based on our simulation parameter configurations, we did not see any remarkable throughput decrease when we introduced packet drop frequency as well as jitter. This led us to avoid considering packet loss probability as a parameter throughout the modelling phase.

Reviewing the relevant literature we found that the most proper models that could apply to the observed throughput behaviour would be the Linear Regression model [16], the Non-linear model [17] Regression and the classical Least Linear Minimum Mean Square Error estimator [18]. Thus we applied all these models to the measurements and concluded which the best-fitting model was.

In particular, we have properly modelled the GridFTP file transfer throughput as a function of the network and non-network parameters mentioned above according to the following formula:

$$Y = f(X, A) + \epsilon \tag{2.2}$$

where Y is the throughput vector, X is the matrix that contains the combination of the five parameters, A is the parameter weighting vector and ϵ is the error vector.

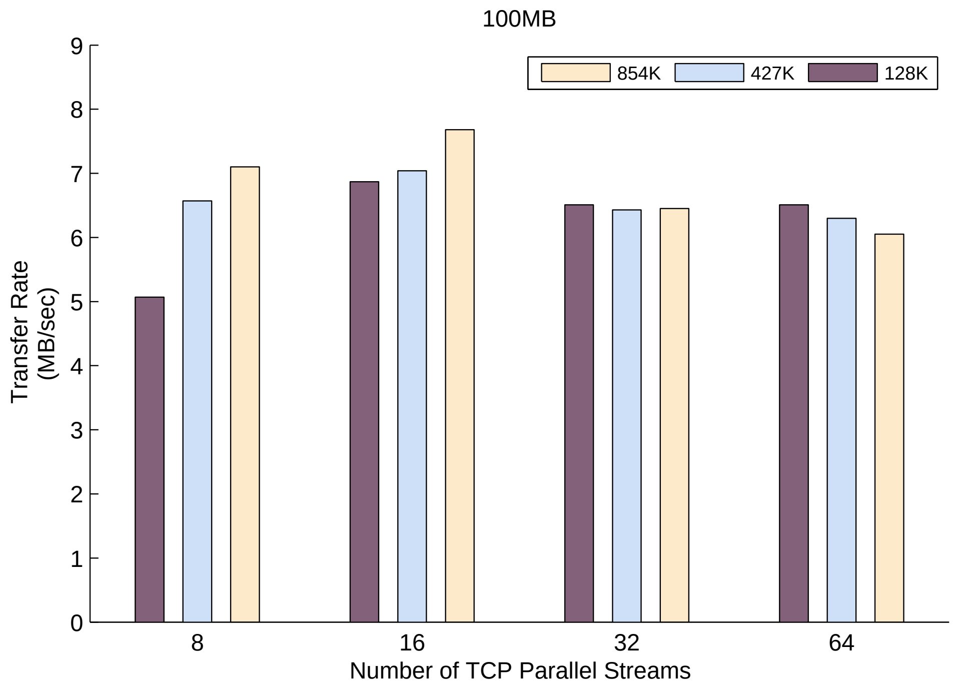


Figure 2.14: *Transfer Rate vs. Parallel streams for a 100MB-file, using different Buffer sizes between TUC and UoP*

For each applied model, the steps below were followed:

1. Computation of the A vector.
2. Estimation of the throughput Y vector based on A vector from step "1" and the parameter X matrix.
3. Statistically processing the deviations between the estimated Y vector and the observed Y vector.

The last step is used as an indicator of which model properly applies to the throughput maximization problem.

Multiple Linear Regression

Regression analysis is the statistical methodology of predicting values of one or more response (dependent) variables from a collection of predictor (independent) variable values. It can also be used for assessing the effects of the predictor variables on the responses [16].

The classical linear regression model states that Y is composed of a mean, which depends in a continuous manner on the a_i's, and a random error ϵ, which accounts for measurement error and the effects of other variables not explicity considered in the model. The values of the predictor variables recorded from the experiment or set by the investigator are treated as *fixed*. The error (and hence the response) is viewed as a random variable whose behaviour is characterized by a set of distributional assumptions.

The linear model has the following formula:

$$Y = XA + \epsilon \tag{2.3}$$

$$y_i = a_1 y_{i1} + a_2 y_{i2} + a_3 y_{i3} + a_4 y_{i4} + a_5 y_{i5} + \epsilon_i$$

$$\text{where } i = 1, 2, ..., n$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}_{(n\times 1)} \qquad A = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix}_{(5\times 1)} \qquad \epsilon = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{pmatrix}_{(n\times 1)}$$

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ 1 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ \vdots & \cdots & & & & \\ 1 & x_{n1} & x_{n2} & x_{n3} & x_{n4} & x_{n5} \end{pmatrix}_{(6\times 1)}$$

Least squares is used to solve the above problem in order to estimate the parameter vector A given the observations Y and X. The least squares procedure determines the A vector that minimizes the sum of squares given by

$$\zeta^2 = \sum_{i=1}^{n} \left(y_i - a_1 x_{i1} - a_2 x_{i2} - a_3 x_{i3} - a_4 x_{i4} - a_5 x_{i5}\right)^2$$

Using calculus the resulting equation for A is given by

$$A = \left(X^T X\right)^{-1} X^T Y \tag{2.4}$$

LMMSE Estimation

In this model Y and X are considered as Random Vectors. Assuming that these vectors are dependent, the following question, which is described in Equation 2.5, comes up: given the random vectors X and Y, what is the vector A that best estimates or approaches the random vector Y.

$$\hat{Y} = XA + \epsilon \tag{2.5}$$

The corresponding minimum-mean-square-error (m.m.s.e for short) matrix of Equation 2.5 is given by

$$m.m.s.e = E\left(y - \hat{y}\right)\left(y - \hat{y}\right)^{*} = R_x - AR_{xy} \tag{2.6}$$

The optimum choice A that minimizes Equation 2.6 (i.e. minimizes the mean-square error in the estimator of each component of the vector Y) is given by

$$A = R_{X,Y} R_X^{-1} \tag{2.7}$$

where $R_{X,Y}$ is the cross-correlation matrix between X and Y and R_X^{-1} is the inverse auto-correlation matrix of X.

Multiple Non-linear Regression

Unlike the Linear Regression (which is a simple straight relationship), the Non-linear Regression is not defined by a specific relationship in the literature. In other words, this type of regression is case dependent. Therefore, by analyzing and graphically representing (see Figures 5 and 6) the throughput measurements in terms of each parameter, we came to the conclusion that the throughput changes exponentially to the parameter variations as shown in the following formula:

$$Y = e^{XA} + \epsilon \tag{2.8}$$

After linearization, Equation 2.8 will become:

$$\log Y = XA + \log \epsilon \tag{2.9}$$

Note that, Equation 2.9 is now a linear equation which can be solved in the same way as with the Linear Regression model.

We have estimated the A vector based on our simulation results adopting the three models mentioned in the previous section with the help of Matlab [19].

From the first view, it seems that the Linear Regression model is the most simplified model of the three because it assumes that the throughput vector Y changes proportionally to the X parameter matrix. Of course, the linear model adoption proved not to be always valid. This happens especially during large file transfers, where there is enough time for the throughput to reach its highest possible value. Under this condition, we do not expect any throughput increase by further increasing the number of parallel streams.

On the other hand, the LMMSE adopts a linear relationship between the parameters and the throughputs, which are going to be estimated. This is a stochastic model and is often called by the name "Linear Least Mean Squares", whereas the previous model is deterministic and is called "Linear Least Squares". In this case, the estimated throughput results were far away from the observed values. This happens mostly due to the small number of trials that have taken place and the small degree of variance in the parameter sets that have been chosen.

Finally, the Non-linear model is based on the multidimensional X, Y data, where Y is exponentially related to X as shown in Equation 2.8. The idea here is to obtain values for the vector A associated with the best fitting curve, i.e. Least Squares. In this case, the theoretical results were found to be closer to the measurements than the other models.

In performing throughput measurement analysis, we thought we had better estimate the deviation between measured and estimated throughput over all possible configuration setups. In principle, we did not count the cases where the deviations were both positive and far away from their pre-defined values (2.2, 1.6 and 1.1 MB/s). The first results showed that the LMMSE model was far behind the other two models in estimating the throughput. Limiting our interest to the Linear and Non-linear Regression models, we computed the percentage of the simulation scenarios whose deviation was below some predefined values.

The first deviation value of 2.2 MB/s was chosen because the Non-Linear model was capable of estimating the throughput of all the scenarios (100%) with deviation below this value. Afterwards, the other deviation values were chosen to be lower to show the superiority of the Non-Linear over the Linear model. As it is shown in Figure 2.15, the accuracy of the Non-Linear

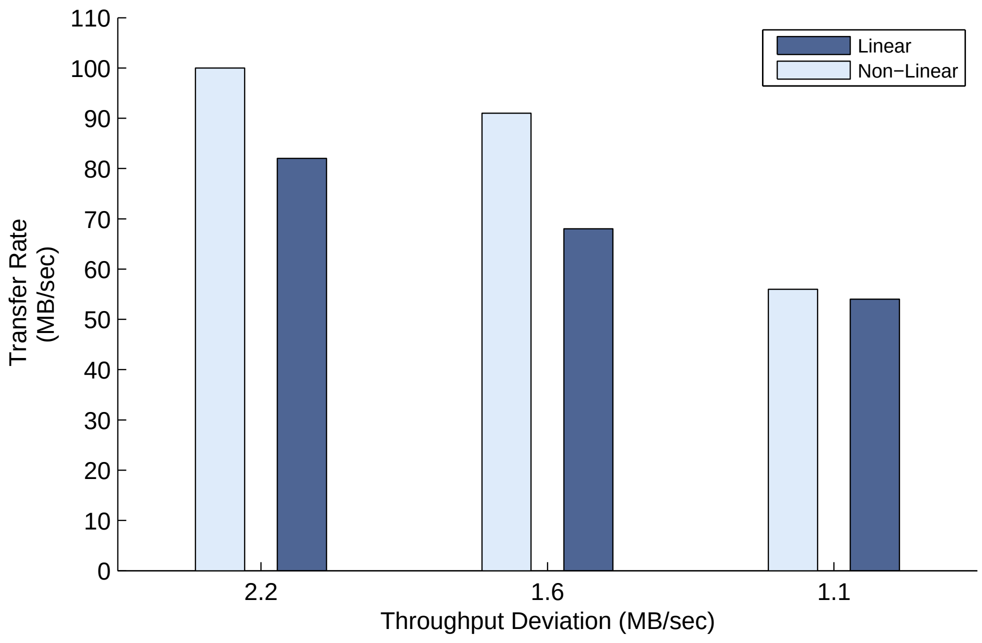


Figure 2.15: *Throughput Deviation*

throughput estimation is very satisfactory.

To sum up, in this work, we try to find the best-fitting model that best suits the GridFTP throughput among different mathematical and statistical regression models. The models examined were: the Linear, the Non-Linear (exponential) and the Stochastic. Our goal has been the correct choice of the GridFTP parameters (number of TCP parallel streams and TCP sender socket buffer size) that would maximize its throughput since no rule of thumb exist on choosing those parameters. Our efforts are based on real-condition and emulation results. Our analysis utilizes both network and non-network parameters. The model, whose estimations best fit the observed throughput measurements, is proven to be the Non-Linear (exponential) model. Comparing our approach with other related works, we can notice that [12] computes, based on one-time measurement of the BDP, the required TCP socket buffer size only. The number of TCP parallel streams is not computed which leads to sub-optimal GridFTP throughputs. Throughout this chapter we have proved that optimal GridFTP throughput could be achieved using both TCP socket buffer size and number of TCP parallel streams parameters. While [13, 14] present an apporach to measure TCP sokcet buffer size and number of TCP connections in a dynamic manner based on a fluid-flow model, no reference of the throughput behaviour under several conditions (network and non-network) is noted. In this work, we beleive that studying the GridFTP throughput would help understand and figure out the way which GridFTP should be dynamically and optimally parameterized.

2.4 Acknowledgments

We would like to express our gratefulness to both Mrs. Lingfen Sun and Mr. Pin Hu from the University of Plymouth, England for the assistance in running the GridFTP trials between our universities.

Many thanks to the creators of LANForge network emulator [15] for their licensing and their helpful hints.

Chapter 3

Bandwidth Estimation

3.1 Introduction

In the previous chapter, we talked about GridFTP service and we studied its throughput behaviour. We have produced and analyzed the best fitting model for the throughput among various mathematical and statistical models. The models studied were: Linear Regression, Least Mean Square Error and Non-Linear/exponential model. For each model, we have used the following parameters as input: bottleneck bandwidth, RTT, number of parallel streams and sender TCP socket buffer size. In conclusion, we have seen that an active probing tool which could actively estimate the network conditions in terms of bottleneck bandwidth and RTT could benefit GridFTP.

Hence our next challenge was to create a tool that could make use of the best fitting model based on the current network conditions in order to appropriately parameterize GridFTP to achieve the maximum available network throughput.

The question to be addressed was "which method or technique can reliably and actively estimate the network conditions, i.e. RTT and bottleneck bandwidth?".

Regarding RTT, first we employed a software program which sends UDP packets to the server and waits to receive them back. RTT is equal to the time difference between the reception and the transmission of each packet. We used UDP packets instead of TCP because first TCP is a feedback protocol and second TCP uses a large amount of overhead.

Regarding the bottleneck bandwidth, things were rather complicated. The extension of the RTT algorithm we first used was proved to be unsuitable in estimating the bottleneck bandwidth due to instabilities. This led us to turn to search more deeply relevant material in the area.

3.2 Packet Pair/Train Technique

In large-scale networks, such as the Internet, it is impossible to keep an updated view of the network conditions for the entire network. This is partly because the amount of information needed would be so large that no single computer would be able to process it. Another problem is that the network is divided into sub-networks that are managed by different corporations. However, to obtain the network conditions and characteristics of an end-to-end path is a feasible task by using active measurement methods. Many of these methods do not require previous knowledge of the network at all.

A general way of active bandwidth measurement is to inject probing packets into an end-to-end path and observe their behaviour to estimate the bottleneck capacity and the available bandwidth. One of the most popular mechanisms is the "*packet pair*" technique in which a source sends pair of packets of the same size to a receiver. The inter-packet dispersion (Δ_S) between these two probing packets changes according to path characteristics such as link capacities and cross-traffic (see Figure 3.1). Hence, the relationship between the inter-packet dispersion at the sender and that at the receiver is exploited to estimate the end-to-end network bandwidth.

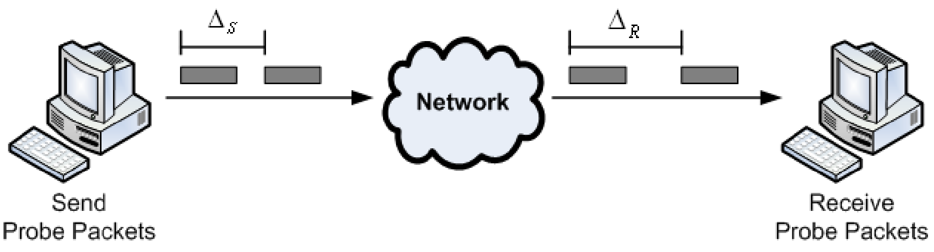


Figure 3.1: *Packet Pair Technique*

The basic packet pair algorithm relies on the fact that if two packets sent back-to-back are queued next to each other at a narrow link and therefore once they will exit the narrow link they will dispersed by, let's say, Δ as shown in Figure 3.2. Mathematically, Δ could be given by:

$$\Delta = P/B \tag{3.1}$$

where P is the packet size, and B is the bandwidth of the narrow link.

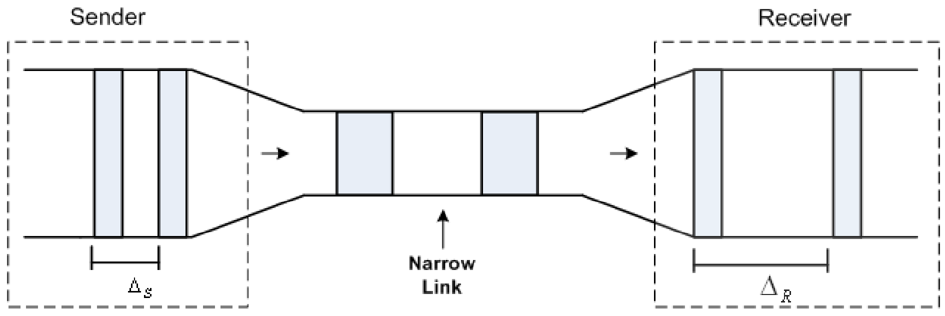


Figure 3.2: *Packet Pair Technique showing a narrow link*

If the two packets have the same size, their transmission delays are the same. This means that after the narrow link, a dispersion of Δ will be maintained between the packets even if faster links are traversed downstream of the narrow link.

From Equation 3.1, bandwidth will be given by:

$$B = P/\Delta \tag{3.2}$$

Packet pair technique involves sending two probing packets back to back as described before. When dealing with large scale networks, two packets are not enough to measure efficiently the bandwidth. Therefore, an extension of the packet pair technique is the packet train. The packet train technique sends more than two probing packets and the dispersion is measured as the time interval of the first and the last packets.

3.3 Active end-to-end Bandwidth Measurement

Suppose, a sender injects N probe packets, each of size P, into a path with an input dispersion Δ_S. Consider C as the *path capacity*. After the packets have traversed the network path the dispersion may have changed due to the network congestion, from an input dispersion Δ_S into an output dispersion Δ_R.

The sending rate, with which the sender sends the probing pakcets to a receiver, could be given by the following equation:

$$S.R. = \frac{P.(N-1)}{\Delta_S} \tag{3.3}$$

The Receiving Rate (R.R.), which is the rate of the probing packets arriving at the receiver, is defined in a similar way as the above Equation 3.3:

$$R.R. = \frac{P.(N-1)}{\Delta_R} \tag{3.4}$$

Consider now, the case where $S.R. > C$. In this case, the probe packets are obliged to queue at C, i.e. the narrow link. The latter leads to a congestion which means that Δ_R is greater than Δ_S. This is shown in Figure 3.3. The bottleneck bandwidth in this case will be equal to the $R.R.$.

Now consdier, the case where $S.R. < C$. In this case, the probe packets will pass through the path without any queue. No congestion will occur and that can be shown in Figure 3.4. In such cases no bottleneck bandwidths are found in the path which in turn means that the path with capcity C can not be measured or estimated.

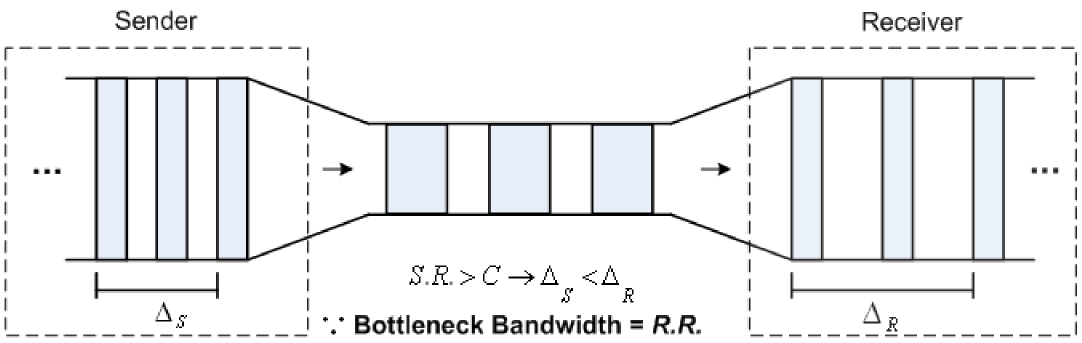

Sender
Receiver
...
...
Δ_S
$S.R. > C \rightarrow \Delta_S < \Delta_R$
∵ Bottleneck Bandwidth = R.R.
Δ_R

Figure 3.3: *A case where congestion occurs*

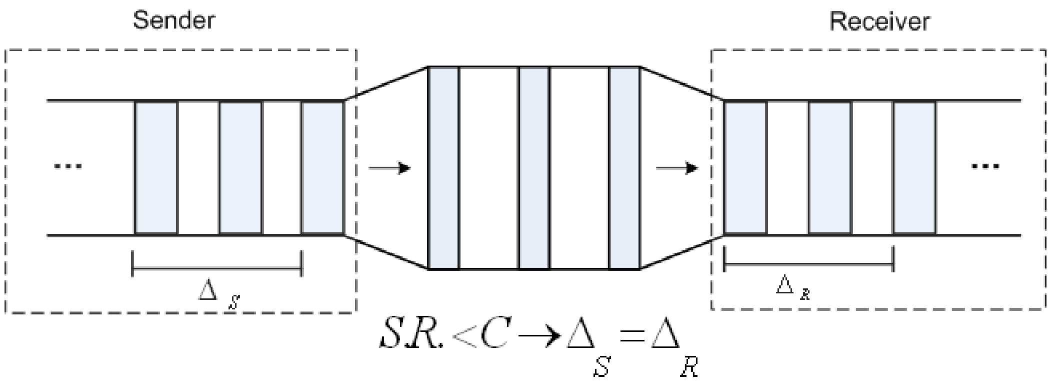


Figure 3.4: *A case where no congestion occurs*

But how do the above equations relate to bandwidth? As described in the previous section, bandwidth is given by the Equation 3.2. The relation between the bandwidth and the sending/receiving rates is stated below.

The bandwidth is *proportional* to the sending rate. In other words, when a sender injects probe packets at a rate of $10Mbps$, the bandwidth that could be estimated in this case is $10Mbps$. While when the sender, sends packets at a rate of $100Mbps$, a $100Mbps$ bandwidth could be estimated. In a path with capcity $C = 20Mbps$ for example, a $S.R. = 5Mbps$ could not estimate the bandwidth in this case. This is desrcibed above (see Figure 3.4).

Equations 3.3 and 3.4 consist of the following variables:

1. Probe packet size, P

2. Number of pakcets sent, N

3. Packet Dispersion, Δ

Consider now the case, where a sender sends N packets, of size P each, and a Δ time gap interval between each packet. Then the total dispersion between the last and the first packet would be equal to $\Delta.(N-1)$.

Equation 3.3 would become:

$$S.R. = \frac{P.(N-1)}{\Delta.(N-1)} \longrightarrow S.R. = \frac{P}{\Delta} \tag{3.5}$$

Note that the above equation and Equation 3.2 are **equal**.

In conclusion, in order to measure actively end-to-end bandwidths, it's necessary that any bandwidth measuring tool would send probe packets with a rate that is larger than the path's capacity. In other words, the basic idea here, is to *throttle* the path and then measure the rate of the arriving packets at the other end, i.e. at the receiver.

The key out here that large packets and very small Dispersions should be used in order to get high sending rates. Based on what we have shown previoulsy, high sending rates have the ability to measure high bandwidth values.

3.4 Related Work

Among the tools in the relative literature, we refer the following three: IGI/PTR [20], Pathchirp [21] and Pathload [22].

Pathload is a tool for estimating the available bandwidth of an end-to-end path from o sender to a receiver. This available bandwidth is the maximum throughput that a flow can get in the path from sender to receiver. Pathload is based on observations of the one-way delay in probe-packet trains. If the input rate of the probe packets is above the available bandwidth the one-way delay will show an increasing trend within the train. If not, the one-way delay is more or less constant. Results are obtained at the receiver side. To locate the available bandwidth Pathload performs a binary search.

The IGI and PTR algorithms send a sequence of packet trains with increasing initial gap from the source to the destination host. They monitor the difference between the average source (initial) and destination (output) gap and they terminate when it becomes zero. At that point, the packet train is operating at the turning point. The final measurement is computed using the IGI and PTR formulas. The results are obtained at the sender side.

Pathchirp injects probe packets in chirps. A chirp is a sequence of probe packets with exponentially decreasing time intervals between the probe packets. It is then possible to investigate a whole range of input rates in one chirp. The analysis is then performed by investigating the relation between delayed and non-delayed probe packets. One chirp gives an estimate of available bandwidth. Pathchirp requires three different machines in order to run (sender, receiver, and master to collect the results).

3.5 Our Approach

Our approach is also based on packet train techqniue. Specifically we make use of Equations 3.3 and 3.4. Our tool sends specific number of equal-sized packets ($N = 251$ & $P = 65000 Bytes$). Moreover, a constant gap between each packet is inserted ($\Delta = 6ms$).

According to the above values, we have:

$$S.R. = \frac{P.(N-1)}{\Delta} = \frac{65000 \times 8 \times (251-1)}{6 \times 10^{-3} \times (250)} = 86.66 Mbps \tag{3.6}$$

The above value means that our tool can measure bandwidth values up to $86 Mbps$.

Table 3.1 shows the basic characteristics of our tool, IGI/PTR and Pathload.

	Our Tool	**IGI/PTR**	**Pathload**
Packet Size, P	$65000 Bytes$	$500 Bytes$	$1472 Bytes$
Number of Packets, N	250	$60/train$	$100/fleet$
Packet Dispersion, Δ	$4-6ms$	Varies	$0.08ms$

Table 3.1: *Characteristics of our too, IGI/PTR and Pathload*

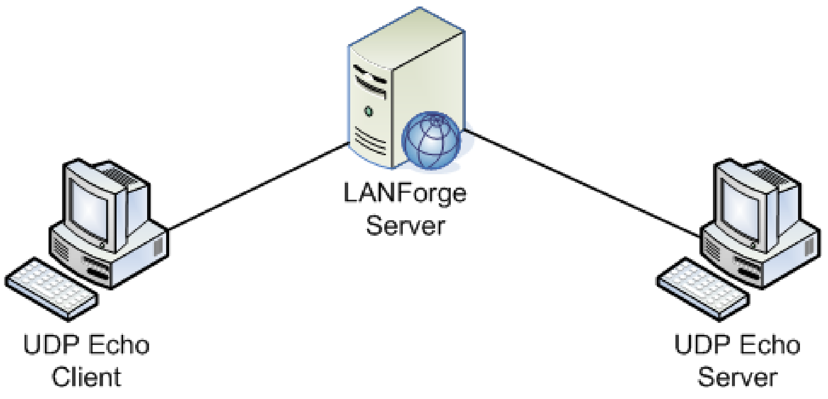


Figure 3.5: *LANForge Topology*

What really distinguishes our approach form the other tools is that we use large packet-sizes ($P = 65000Bytes$). While, all the other tools use small packet-sizes (up to $1500Bytes$). There is a reason for that. The TCP/IP stack uses what is called Maximum Transmission Unit (MTU). MTU refers to the size (in bytes) of the largest packet that a given layer of a communication protocol can pass onwards. MTU parameters usually appear in association with ethernet card communication interfaces, i.e. NIC. MTU is usually equal to 1500 bytes. That's why all the tools mentioned above use packet-sizes up to the MTU value. In our approach we neglect the lower layers of TCP/IP and focus on the application layer. According to our tool, using packet-sizes up to the MTU value would limit the sending rate and thus would limit the bandwidth value that can be estimated as can be shown below:

$$S.R. = \frac{P}{\Delta} = \frac{1500 \times 8}{6 \times 10^{-3}} = 2Mbps \tag{3.7}$$

Another difference between our tool and the other tools, is the gap or dispersion (Δ) value inserted between each packet. The other tools use small dispersions ($< 1ms$) while in our tool we are obliged to use dispersions $> 4ms$. This is described below:

Our initial tests concerned sending probing packets simultaneously from the sender to the receiver. We noticed that the receiver could not process more than 300 hundred packets sent simultaneously. Sending probing packets with a gap inserted between each packet did solve the problem. We applied to our system the case described in Section 3.3 where $S.R. < C$. In this case, we expect to have $S.R. = R.R.$. In other words, we expect $\Delta_S = \Delta_R$ (see Figure 3.4).

In order to test the above theory we considered the following scenario:

Two probing packets of size $1500Bytes$ and a bandwidth value $100Mbps$ set by LANForge emulator were used. Moreover, a dispersion range of $1ms$ to $10ms$ was used. Note that the higher sending rate is acheived when a $1ms$-gap is used. According to Equation 3.3, sending rate would be equal to $12Mbps$. Since the capacity is set to $100Mbps$ that would apply to the case mentioned above where $S.R. < C$ which lead to $\Delta_S = \Delta_R$.

Initially we used a dispersion value of $\Delta_S = 1ms$. As mentioned before we expect to have $\Delta_R = 1ms$ but the value we get is $\Delta_R = 4ms$. The same happens when using $\Delta_S = 2ms$. Other values tested are shown in Table 3.2 below. This happens due to hardware/software drawbacks and inabilities.

In this thesis, we do not refer to these inabilities.

$\mathbf{\Delta_S}$	$1ms$	$2ms$	$3ms$	$4ms$	$5ms$	$6ms$	$7ms$	$8ms$	$9ms$	$10ms$
$\mathbf{\Delta_R}$	$4ms$	$4ms$	$4ms$	$4ms$	$5ms$	$6ms$	$7ms$	$8ms$	$9ms$	$10ms$

Table 3.2: *Dispersion Comparison under No-congestion Network*

According to the above table, the tests we conducted shows that due to hardware/software inabilitites it is impossible to have $\Delta < 1ms$.

Pathload uses a maximum pakcet size of $1472Bytes$. A constant gap of 80 us. That would give us a sending rate of $147Mbps$, according to Equatoin 3.5.

IGI/PTR in it's default run uses 60 packets per train, $500Bytes$ each. This is suitable for measuring low-scale networks (up to $1Mbps$). In order to measure large-scale networks, these values should be modified. We have seen through a lot of tests, that 200 pakcets per train and $20000Bytes$ each, give satisfying results for the PTR. The IGI under such conditions would only work properly for values equaland below $20Mbps$. This is shown in Figure 3.7.

We used LANForge [15] emulator in order to test the integrity of our approach, IGI/PTR and Pahtload. Figure 3.5 shows the network topology.

Figure 3.7 compares our approach with the IGI/PTR tool. Notice that IGI works properly for $20Mbps$ bandwidths and below. While PTR can be comparable to our case. That is, because PTR uses the same formula as our tool (see Equation 3.3).

Bandwidth (Mbps)	5	10	20	30	40	50	60	70	80
Our Tool	5	10	20	30	40	50	61	72	74
PTR	4.2	8.7	19.6	28.3	38.1	46.9	54.1	62.1	72.8
Our Tool / Bandwidth	1	1	1	1	1	1	1.01	1.03	0.93
PTR / Bandwidth	0.84	0.87	0.98	0.94	0.95	0.94	0.90	0.87	0.92

Table 3.3: *Our Tool compared with PTR*

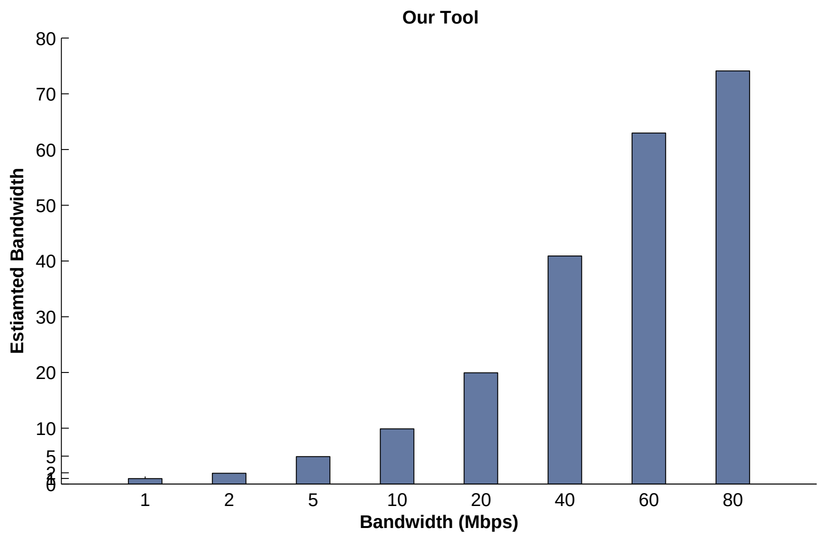


Figure 3.6: *Results from our own approach up to 80Mbps*

3.6 Implementation

Java was the first programming language designed from the ground up with networking in mind. Java was originally designed for proprietary cable television networks rather than the Internet, but it's always had the network foremost in mind. One of the first two real Java applications was a web browser. As the global Internet continues to grow, Java is uniquely suited to build the next generation of network applications. Java provides solutions to a number of problemsplatform independence and security being the most importantthat are crucial to Internet applications, yet difficult to address in other languages.

Since Java provides packages that deals with Networking our choice was Java Networking as the base language for our tool and since TCP and UDP protocols are the basis of networking we had to choose one of them. We chose UDP protocol for the same reasons described in Section 3.1. An echo UDP sender and an echo UDP receiver were implemented. The echo UDP Sender sends a specific number (N) of equally sized (packet-size P) probing packets. These probing packets are separated by an initial constant gap of $6ms$.

To sum up, our tool is based on the well-known packet pair/train tech-

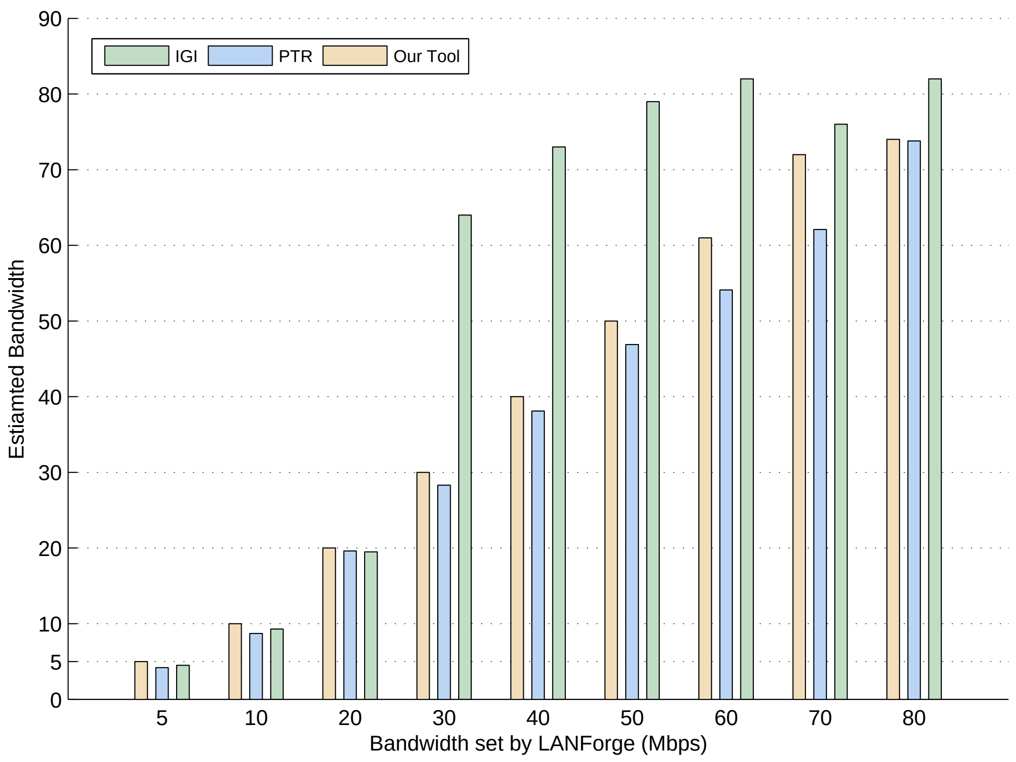


Figure 3.7: *Our tool, IGI and PTR*

nique. Our tool was implemented based on Java Networking and specifically based on UDP classes and packages. Probing packets are sent from one end to another. In order to estimate or measure the bottleneck bandwidth, the underlying network path should be burst (throttled) with probing packets. This in turn would lead these probing packets to queue at some point (narrow link). After this occurs, one can easily measure the rate at which these packets reach the other end. The receiving rate, at that point, would be equal to the bottleneck bandwidth. In our approach, we use large packet-sizes which lead to higher sending rates which in turn could estimate higher bandwidth values. Moreover, we proved that using small dispersion values, compared to other tools, would lead the receiver to a state where it can not handle or process the incoming packets and thus lead to incorrect measurements of the bandwidth. Among IGI/PTR and Pathload, only PTR seems to be comparable to our approach. Initially, when we tested PTR it produced abnormal results. We had to run a lot of tests until we started to see satisfying results. Our application is created in such a way that it would give results wihtout having to modify any parameter. Our application runs faster than the other tools since the other tools send a large number of packets (at least 1000 packets). we use only 200-300 packets.

Chapter 4

Enabling Applications for the Grid

4.1 Introduction

It is practically impossible nowadays to do science without computers. Research in many fields of science nowadays, has evolved new methods that require high computing capabilities. This means that the scientists are facing increasingly complicated problems which require much more than a single computer's capabilities. Often, a single computer, a cluster of standard computers or even a special-purpose supercomputer is not enough for the calculations scientists really want to do. Although, computers are improving incredibly fast, they still do not keep up with the increased demands of the scientists. Due to that, its very difficult, very expensive and sometimes impossible to achieve certain scientific goals with current computer technology. Physicists, biologists, chemists, etc. develop and submit applications with high computing demands to parallel, distributed or even Grid Systems with the objective of obtaining the results in the shortest possible time and of considering the largest possible problem size. Therefore, performance is a key aspect in parallel/distributed computing. We talk more about performance in the next section.

Some issues that do concern scientists could be the following:

1. The amount of data scientists need can be huge. Moreover, data is stored in different locations. Satellite images of the Earth can be a good example. It might take a lot of time to copy the data one needs to one central computer in order to analyze it. So ideally the scientist wants to do the computation where the data is.

2. The amount of calculations the scientist has to do is huge. For example,

simulating the effect of thousands of potential drug molecules on a protein related to some disease.

3. A scientific team with members around the globe wants to share large amounts of data and do complex analysis of the data rapidly online together.

To solve these problems the following idea was put forth:

1. of having huge storage space so they would never have to worry where to put the data,

2. of having nearby large computing power available for their institution whenever they need it,

3. of being able to collaborate with distant colleagues easily and efficiently, safely sharing with them resources, data, procedures and results.

If we take a look at the world today, we can see that it contains a lot of computers (several million). These could be desktop PCs, workstations, mainframes and supercomputers. Of course, these computers belong to many different people (students, doctors, secretaries) and institutions (companies, universities, hospitals). Most of these computers are probably connected to the Internet. Imagine now, that all these computers around the world are connected together. Its here where the idea of the Grid comes out.

If a scientist wants to run a colleagues molecular simulation program, (s)he would no longer need to install the program on his machine. Instead, he could just ask the Grid to run it remotely on his colleagues computer. In fact, he would not need to ask the Grid anything. It would find out the best place to run the program, and install it there.

If one needs to analyze a lot of data from different computers all over the globe, he could ask the Grid to do this. Again, the Grid could find out where the most convenient source of the data is without me specifying anything and do the analysis on the data wherever it is. Moreover, if one needs to do some analysis interactively in collaboration with several colleagues around the world, the Grid would link computers up and figure out who should be able to take part in this common activity.

Grid computing can be seen as the evolution of distributed computing and its several derivatives (metacomputing, cluster computing and P2P). All made possible by the advent of very high-speed Internet connections, and of powerful computer processors that are able to run quite complex middleware in the background without disturbing the task that the computer is trying to handle [24].

As Internet connect speed increases, the difference between having two PCs in the same office, the same building, the same city or the same country shrinks. By developing sophisticated middleware which makes sure that widely distributed resources are used effectively, Grid computing gives the user the impression of shrinking the distances further still. In addition, as the middleware gets more sophisticated, it can deal with the inevitable differences between the types of computers that are being used in a highly distributed system, which are harder to control than within one organization. One of the most popular middleware packages today is called Globus, and it is essentially a software toolkit for making Grids. With such middleware, the aim is to couple a wide variety of machines together effectively, including supercomputers, storage systems, data sources and special classes of devices such as scientific instruments and visualization devices. Finally, Grid computing aims to use resources that are not centrally controlled. The sharing is across boundaries - institutional and even national - which adds considerable complexity, while bringing also huge potential benefits.

Grid should in principle have access to parallel computers, clusters, local Grids, even Internet computing solutions, and would choose the appropriate tool for a given calculation. In this sense, the Grid is the most generalized, globalized form of distributed computing one can imagine.

But reality is far enough from theory. There are some issues that should be resolved in order to achieve all what is discussed above.

Some of these issues are listed below:

- The ability to allow for independent management of computing resources
- The ability to provide mechanisms that can intelligently and transparently select computing resources capable of running a user's job
- The understanding of the current and predicted loads on grid resources, resource availability, dynamic resource configuration, and provisioning
- Failure detection and failover mechanisms

- Ensure appropriate security mechanisms for secure resource management, access, and integrity

4.2 Performance

As mentioned before, performance is a key aspect in parallel/distributed computing. To obtain an efficient application, it is necessary to consider two main issues:

- The design and development of the application.
- Application performance analysis and tuning.

Design and development of distributed applications require a detailed knowledge of the system's features to take advantage of its capabilities. Moreover, distributed systems usually involve heterogeneous systems, and this fact further complicates the design of efficient applications. However, in many cases the developers of these high performance systems are not computer specialists and typically, they are not interested in the low-level details of their systems. Therefore, many libraries and software layers have been developed to facilitate the construction of distributed applications. The main concern is that, in many cases, the software layers that have been introduced imply degradation in the performance of the application, due to the general conception of such software [23].

Once an application has been developed and has been tested from the functional point of view, the programmers must investigate its performance. Therefore, they carry out some performance analysis to assess the behaviour of the application, to detect potential performance bottlenecks and to determine their causes. Finally, the programmers modify the application, according to their best knowledge, to overcome the problems identified.

It must be pointed out that, in practice, the causes of performance bottlenecks can be found at different levels. For example, a communication problem can result from:

- An erroneous conception of the application that provokes an unnecessary blocking time in a receive primitive.
- Communication library implementation. In many cases, the design or implementation of the software layers is generic and is not optimized

for a particular system or for particular conditions. This implies that the application may behave differently than expected.

- Operating system features. For example, an inappropriate buffer size and the message treatment at the protocol level can interfere with application message delivery times.
- Underlying hardware capabilities. The interconnection network features (latency, bandwidth, etc.) or even the contention in the network can seriously slow down the application.

Moreover, in many cases the application performance depends in the input data set. This fact implies that a set of potential bottlenecks can vary for different executions.

As a consequence, performance analysis is a difficult and costly task. The developers are forced to master the application, the involved software layers and the distributed system behaviour and this can be too complex for non-specialists.

To tackle all these problems, user-friendly tools should be available. The required tools include programming environments, debugging tools and performance analysis systems. However, in the area of performance analysis, there is still a lack of real useful tools and most of what is available require a high degree of expertise from the user.

The classical approach for performance analysis and tuning was based on visualization tools. First, programmers develop and debug their applications. Next, they run them with the help of a monitoring tool that collects the information on the behaviour of the application. Then, in a post-mortem phase, a visualization tool shows the collected information using different views (such as gantt charts, bar charts, pie charts, etc.).

However, a more difficult task is that of identifying the real causes of the bottlenecks and determining what should be modified in the application source code, or in the system itself, to overcome them. In the classical approach, these issues are not addressed. Therefore, in recent years the interest in automatic performance analysis has significantly increased. In the automatic approach, the tools guide the programmer in the performance improvement phase by searching for the causes of bottlenecks and by providing useful recommendations on how to solve them.

A major method of constructing applications to run on a computational Grid is to assemble them from components - separately deployable units of computation of well-defined functionality. Performance steering is an adaptive process involving run-time adjustment of factors affecting the performance of an application.

A component is a unit of computation which has a well-defined functionality, and which can be composed with other components to create an application. Since a component can be individually deployed on to a hardware platform, component-based applications are, in the general case, distributed applications. Component-based applications are good candidates for deployment on to computational Grids and several component frameworks exist to support the interaction of components.

A computational Grid is a network of heterogeneous machines which can inter-operate to execute applications. The essence of a computational Grid is this inter-operation, supported by so-called *grid middleware* - a suite of more-or-less integrated software which interacts to provide security, resource management and information services. Although it is possible to implement component-based applications directly on this Grid middleware layer, it is also to have a further middleware layer to support components specifically. Such *component frameworks* allow greater flexibility and generality in the interactions between components.

Central to such middleware layers is some *resource scheduler*, responsible for allocating Grid resources to applications submitted for execution. Condor is an example. It can be referred to as *external resource scheduler* because they are external software to the middleware.

An application, consisting of multiple components to be executed on a computational Grid, requires that multiple resources are allocated to it simultaneously. There are two factors associated with resource scheduling for applications consisting of components which are of relevance here. The first is the *allocation* of a set of resources to the application as a whole. The second is the *distribution* of the components of the application over these resources.

4.3 Jobs and Grid Applications

For an application to benefit from a grid environment it must have the ability to be paralleled or otherwise the application can't take advantage from the grid. Some may think even if the application cannot be paralleled it still can benefit from the grid since the Grid can be seen as a distributed cluster where even a single threaded batch job could be able to run on any set of systems taking advantage of unused cycles. That's also true. In general, a grid environment provides greater availability, reliability, and cost efficiencies than a cluster within an organization, an institute or university.

But still, if an application can be paralleled the gain from the Grid could reach ten times than that without parallelism. More precisely, if an application consists of several jobs (job is a single unit of work) that can all be executed in parallel, a grid may be very suitable for effective execution of these jobs on dedicated nodes, especially in the case when there is no or a very limited exchange of data among the jobs.

The latter could be more comprehensive by reading the following:

- From an initial job, a number of jobs are launched to execute on preselected or dynamically assigned nodes within the Grid.
- Each job may receive a discrete set of data, and fulfils its computational task independently and delivers its output.
- The output is collected by a final job or stored in a defined data store.

One can figure out that some kind of an automatic/dynamic mechanism is needed in order to achieve the synchronization between the above. Indeed, grid services, such as a broker and/or scheduler, may be used to launch each job at the best time and place within the grid.

Traditional applications execute in a static environment with fixed assets. The application I/O processes and sub-processes are all located on the same machine. In a grid environment where resources are dynamically distributed and allocated there are some considerations that should be taken into mind.

One who wants to run an application on multiple resources in a grid, (s)he must consider whether the processing of the data can happen in parallel tasks or whether it must be serialized. A lot of times, applications can be divided into parallel and serial tasks. (S)He then have to be careful how to organize those parts of the application in order to avoid any sort of failure.

Most of the time, those who are creating "*heavy-computing*" applications is not aware of the Grid. They maybe have the concept of Grid in mind, but they do not know how to create grid-enabled applications. They may not be trained to build such applications. These people are scientists, biomedical engineers, etc. It's is very important also to provide well documentation on such applications. Well documentation would help on a large scale those who are responsible for grid-enabling the applications. The grid-enabling engineers in turn have the duty, based on the documentation, to build grid-enabled applications that can benefit from a grid environment as much as possible.

Building a grid-enabled application is not an easy task. It's a hard one and it's time consuming. Adoption of the language programming essential basis, where a big problem is divided into smaller ones and then assembled all together, could help in this situation.

First, one must understand and comprehend very well what the application do. Second, divide or break the application into independent (parallel tasks) and dependent units (serial tasks). Parallel tasks could then run as individual jobs in the grid. Third, synchronization between the parallel tasks, serial tasks and parallel-serial communication should be resolved. Forth, a grid environment with a suitable middleware that can meet all the above considerations is needed.

Independent units or parallel tasks can be defined as separate data sets per job where none of the jobs need results from another job as input. Figure 4.1 illustrates a parallel application flow. For example, in the case of a simulation application that is based on a large array of parameter sets against which a specific algorithm is to be executed, a Grid can help to deliver results more quickly. A larger coverage of the data sphere is reached when the jobs can run in parallel on as many suitable nodes as possible. Such a job can be as complex as a sophisticated spreadsheet script or any multi-dimensional mathematical formula of which each requires intense computing.

In contrast to the parallel tasks, are the serial tasks. In this case there is a single thread of job execution where each of the subsequent jobs has to wait for its predecessor to end and deliver output data as input to the next job. Figure 4.2 illustrates a serial application flow. In this case, the advantages of running such tasks in a grid environment are not based on access to multiple

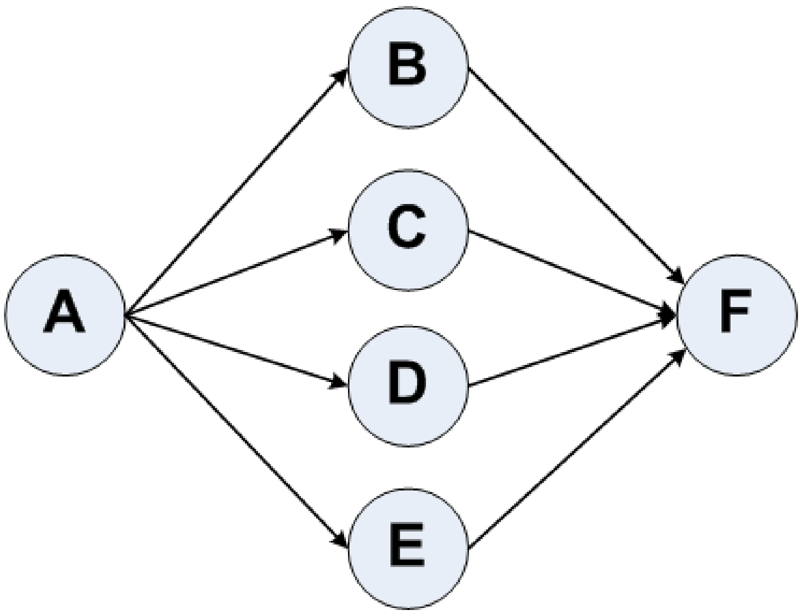


Figure 4.1: *Parallel Application Flow*

systems in parallel, but rather on the ability to use any of several appropriate and available resources. Note that each job does not necessarily have to run on the same resource, so if a particular job requires specialized resources that can be accommodated, while the other jobs may run on more standard and inexpensive resources. The ability for the jobs to run on any of a number of resources also increases the application's availability and reliability. In addition, it may make the application inherently scalable by being able to utilize larger and faster resources at any particular point in time. Nevertheless when encountering such a situation it may be worthwhile to check whether the single jobs are really dependent of each other, or whether due to its nature they can be split into parallel executable units for submission on a Grid. For best performance, these kinds of processes might be executed on a single CPU or cluster, though performance is not always the primary criteria. Cost and other factors must also be considered, and once a grid environment is constructed such a job may be more cost effective when run on a grid versus utilizing a dedicated cluster.

Loose coupling

For a grid, this means the need for a job flow management service to handle the synchronization of the individual results. Loose coupling between

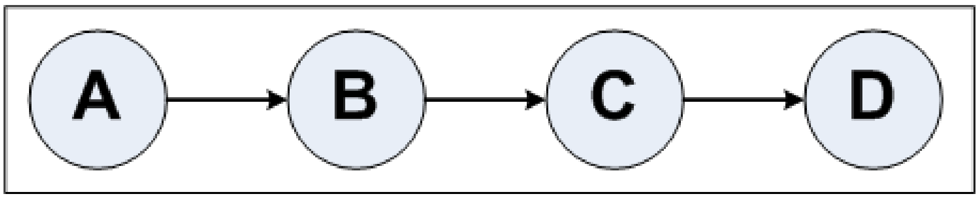


Figure 4.2: *Serial Application Flow*

the jobs avoids high inter-process communication and reduces overhead in the grid.

For such an application you will need to do more analysis to determine how best to split the application into individual jobs, maximizing parallelism. It also adds more dependencies on the grid infrastructure services such as schedulers and brokers, but once that infrastructure is in place, the application can benefit from the flexibility and utilization of the virtualized computing environment.

Jobs and sub-jobs

Another approach to ease the managing of jobs within a grid application is to introduce a hierarchical system of sub-jobs. A job could utilize the services of the grid environment to launch one or more sub-jobs. For this kind of environment an application would be partitioned and designed in such a way that the higher-level jobs could include the logic to obtain resources and launch sub-jobs in whatever way is most optimal for the task at hand. This may provide some benefits for very large applications to isolate and pass the control and management of certain tasks to the individual components.

4.4 Job Criteria

A job as part of a grid application can theoretically be of any type: Batch, standard application, parallel application, and/or interactive.

4.4.1 Batch job

Jobs in a grid environment could be a traditional batch job on a mainframe or a program invoked via a command-line interface in a Windows, Unix, or Linux environment. Normally, arguments are passed to the program, which can represent the data to process and parameter settings related to the job's execution.

Depending on its size and the network capacities, a batch job can be sent to the node along with its arguments and remotely launched for execution. The job can be a script for execution in a defined environment (for example, Shell, Java, or Perl script), or an executable program that has few or no special requirements for operating system versions, special DLLs to be linked to, JAR files to be in place or any other special environmental conditions. The client, portal, and/or broker may need to know the specific requirements for the job so that the appropriate resource can be allocated. The data for its computation are either transmitted as arguments or accessible by the job, be it in local or remote storage or in a file that can also be sent across the grid.

A batch job, especially one with few environmental requirements, in general is well suited for deployment in a grid environment.

4.4.2 Standard application

Often standard application, like spreadsheets or video rendering systems, requires an installation procedure and cannot be sent over the network to run simply as a batch job. However, a command line interface provided can be remotely used on a grid for execution of the application where it is installed.

In this case, the grid broker or grid portal needs to know the locations of the application and the availability of the node. The locations of the applications on the grid are relatively fixed, meaning in order to change it a new installation has to be performed and the application may need to be registered with the grid portal or grid server before it can be used.

New installations are mostly done manually as the applications often require certain OS conditions and application settings, or very often when installing on Windows a reboot needs to be executed. This makes a standard application in many cases quite difficult to handle on a grid, but does not exclude them. As advances in autonomic computing provide for self-provisioning, there will be fewer restrictions in this area.

Using standard software as jobs within a grid could raise licensing issues, either due to the desire to have the application installed on many different nodes in the grid, or related to single-user versus multi-user license agreements.

4.4.3 Parallel applications

Applications that already have a parallel application flow, such as those that have been designed to run in a cluster environment, may already be suited to run in a grid environment. In order to allow a grid server or grid portal to take the most advantage of these, there needs to be identifiable and accessible handles to the inner functions/jobs of such a parallel application. If this is not the case, such an application can only be handled as one unit, similar to a standard application. However, it makes sense to include such an application in a grid if the overall task requires more than the resources available in a given cluster. This means that the grid could include several clusters with copies of a parallel application.

4.4.4 Interactive jobs

Interaction with a grid application is most commonly done via the grid portal or grid server interface. This implies that other than launching the job, there should not be on-going interaction between the user and the job.

Of course, if we go back to the initial view of the gird as a virtual computing resource, it is certainly feasible to think of an application requiring user interaction to be launched on any appropriate resource within the grid as long as a secure and reliable communications channel could be created and maintained between the user and the resource. Though the GSI-Enabled SSH package is available and could be used to create a secure session, the Globus Toolkit does not provide any tools or guidance for supporting such an application.

There would be many considerations and issues involved in the development and deployment of such an application within a grid environment.

4.5 Programming Language Considerations

Whenever an application is being developed, the question of the programming language to be used arises. The grid environment may include additional considerations. Jobs that are made for high-performance computing are normally written in languages such as C or Fortran. Those jobs whose individual execution time does not play the most important role for the application, but whose contents and tasks are of more importance, may be written in other languages such as Java, or in scripting languages such as

Perl.

Within a single grid application one might even consider writing various parts in different languages depending on the requirements for the individual jobs and available resources.

Some of the key considerations include:

- *Portability to a variety of platforms*: This includes binary compatibility where languages such as Java provide an advantage, as a single binary can be executed on any platform supporting the Java Virtual Machine. Interpreted languages such as Perl also tend to be portable, allowing the application to run no matter what the target platform.Portability of source code can also be considered. For instance, one may decide to develop an application using C, and then compile it multiple times for a variety of target platforms. This will require additional work by the infrastructure to ensure that appropriate executables are distributed to any target resource.

- *Run-time libraries/modules*: Depending on the language and how the program is linked, there may be a requirement for run-time libraries or other modules to be available. Again, the successful running of an application will depend on these libraries being available on, or moved to, the target resource.

- *Interfaces to the grid infrastructure*: If the job must interface with the grid infrastructure, such as the Globus Toolkit, then the choice of language will depend on available bindings. For example, Globus Toolkit V2.2 (by the time of writing this thesis, Globus Toolkit 4.0.6 was available) includes bindings for C. However, through the CoG initiative, there are also APIs and bindings for Java, Perl and other languages. Note that an application may not have to interface with the Globus Toolkit directly, as it is more the responsibility of the infrastructure that is put in place. That is, given an appropriate infrastructure, the application may be developed such that it is independent of the grid-specific services.

One of the driving factors behind the OGSA initiative is to standardize on the way that various services and components of the grid infrastructure interface with one another. This provides programming language transparency between two communicating programs. That is, a program written in C, for example, could communicate with or through a service that is written in another language.

4.6 Job Dependencies on System Environment

As shown earlier, a grid application does not require a homogenous run-time environment, but there are certain considerations to be made in order to plan for the most beneficial deployment of it. For any job in a grid application the following environmental factors may affect its operation. When developing an application, one must consider these factors and either design it to be as independent of these factors as possible, or understand that any dependencies will need to be taken into account within the grid infrastructure.

- *Operating Systems* version homogeneous levhomogeneous parameter settings that are necessary for execution of the job, as well its reliance on certain system services and auxiliary programs such as a registry. It is worthwhile to consider whether the grid application will be capable of running its jobs on any node with different operating systems or whether it will be restricted to a single operating system.

- *Memory size* required by a job may limit the possible nodes on which it can run. The available memory size depends not only on its physical presence at a node, but also on how much the operating system is capable of granting at run-time.

- *DLLs* that are to be linked for the execution of the job. They either need to be available on the target resource or could possibly be transferred and made available on the resource before the job is executed.

- *Compiler settings* play a role as compiler flags and locations may be different. For example, subtle differences like bit ordering, and number of bytes used for real and integer numbers may cause failures when a job is compiled on a different node or operating system than the one it will eventually be executed on.

- *Runtime environment* that has to be in place and ready to receive the job for execution. For instance, the right JDK or interpreter versions may have to be planned and in place.

- *Application Server* version and standard as well as its capacity may be needed to be considered as well as access requirements and services to be used.

- *Other applications* that are needed to properly run a job have to be in place prior to deployment of the grid application. These applications can be compilers, databases, system services such as the registry under Windows, and so on.

- *Hardware devices* that are required for certain jobs to perform their tasks. For example, requirements for storage, measurement devices, and other peripherals must be considered when building the application and planning the grid architecture.

When developing the grid application, these prerequisites need to be checked in order to avoid too many restrictions for job execution. A large number of restrictions could mean more complicated enablement as well as limiting the number of possible nodes on which the job will be able to run. Therefore, it is better to restrict such requirements during development of the application such that jobs can run in as generic an environment as possible.

4.7 Job Topology

For a grid application, there are various topology-related considerations. There are certain architectural requirements covering the topology of jobs and data. When designing the grid application architecture, some of the key items to consider are:

- Where grid jobs can run
- How to determine a suitable node for executing the individual jobs
- Location and amount of data to be processed byt he jobs
- Availability and performance values of the individual nodes at time of execution

When developing grid-enabled applications you may not know anything about the topology of the grid on which they will run. However, especially in the case of an intra-grid that may be put in place to support a specific set of applications, this information may be available to you. In such a case, you may want to structure your application and grid in such a way as to optimize the environment by considering the location of the resources, the data, and the set of nodes that a particular application might run on.

4.8 Passing of Data Input/Output

Any job in the grid application needs to pass data in and out.

There are various ways to realize the passing of data input and output that are to be considered during application architecture and design:

- *Command line interface* can be a natural way for batch jobs and standard applications to receive data. In this case, the data input normally will not be complex in nature, but consists of certain arguments used as parameters to control the internal flow of the job. Such command lines can easily be integrated in scripts executed at the system level or within a given interpreter. The transfer of data to the job as a consumer happens immediately at launch time. The amount of data will normally be small. For larger amounts of data there can be arguments that specify the name of a data file or other data source.

- *Data store* of any kind, such as data files in the file system (local or on a LAN or WAN) or records in a database, a data warehouse or other storage system that is available. These data stores can be used for input as well as output of data given that the required access rights are granted to the job. The transfer of data in can be done anytime before the job executes, and likewise the output data could be read anytime after the job completes, therefore providing flexibility for data movement operations.

- *Message queues*, like those provided by WebSphere MQSeries, are well suited to be used for asynchronous tasks within a grid application, especially when guaranteed delivery of the data provided to the job and generated by the job is of high importance. A job can access the data queues in various ways, normally using specific APIs for putting or getting data as well as for polling the queue for data waiting for processing. In an environment where message queuing servers are already installed, this type of data passing may be desirable.

- *System return value*, is a corresponding case to the CLI and normally a way a batch job or any CLI invoked program will return data, or at least status information about how the job ended. This indicates to the grid server or grid portal the status of the individual job and requires appropriate management. The resulting data of the job may be passed to a data store or message queue for further processing or presentation.

- *Other APIs*, when communicating with Web services, Web servers, application servers, news tickers, measurement devices, or any other external systems, the appropriate conditions for data passing in and out have to be taken into consideration. In these cases, you may use HTTP, HTML, XML, SOAP, or other high-level protocols or APIs.

As indicated, for a grid application there may not be only one way to pass data for a job, but you may use any combinations of the described mechanisms. It is advised to program grid jobs in such a way that the data sources and sinks are generically handled for more flexible grid topologies. The optimal solution depends on the environment and the requirements to be considered at the architecture and design phase of the grid application.

4.9 Qualification Scheme for Grid Applications

In this section a usable format of a qualification scheme for grid applications is provided. We also provide a criteria list that may be looked at as a knock-out list. That is, it includes attributes of an application or its requirements that may inhibit an application from being a good candidate for a grid environment. The list may not be complete and depends on the local circumstances of resources and infrastructure. The qualification scheme acts as a basis for architecture and project planning for a grid application.

Knock-out criteria for grid applications

Earlier sections have discussed considerations for grid-enabling an application from the perspectives of infrastructure and application functionality. However, not all applications lend themselves to successful or cost-effective deployment on a grid. A number of criteria may make it very difficult, require extensive work effort, or even prohibit grid-enabling an application. Criteria below may preclude deploying an application to the grid without having to perform an extensive analysis of the application.

Some facts such as temporary data spaces, data type conformity across all nodes within the network, appropriate number of SW licences available in the network for the grid application, higher bandwidth, or the degree of complexity of the job flow can be solved, but have to be addressed up front in order to create a reasonable grid application.

An application with a serial job flow can be submitted to a grid, but the benefits of grid computing may not be realized, and the application may be adversely affected due to grid management overhead. However, by exploiting the grid and submitting the application to more powerful remote nodes it may very well provide business value.

In this list of knock-out criteria the most critical items are named that most certainly hinder or exclude an application from use on a grid:

1. High inter-process communication between jobs without high speed switch connection (for example, MPI, in general, multi-threaded applications need to be checked for their need of inter-process communication.

2. Strict job scheduling requirements depending on data provisioning by uncontrolled data producers.

3. Unresolved obstacles to establish sufficient bandwidth on the network.

4. Strongly limiting system environment dependencies for the jobs (see Section 4.6 on page 58).

5. Requirements for safe business transactions (commit and roll-back) via a grid. At the moment there are standards for secure transaction processing on grids.

6. High inter-dependencies between the jobs, which expose complex job flow management to the grid server and cause high rates of inter-process communication.

7. Unsupported network protocols used by jobs may be prohibited to perform their tasks due to firewall rules.

4.10 Data Management Considerations

No matter what the application, it generally requires input data and will produce output data. In a grid environment, the application may submit many jobs across the grid and each of these jobs in turn will need access to input data and will produce output data.

One of the first things to consider when thinking about data management in a grid environment is management of the input data and gathering of the output data. If the input data is large and the nodes that will execute the individual jobs are geographically removed from one another, then this may involve splitting the input data into small sets that can be easily moved across the network assuming the individual jobs need access to only a subset of the data.

The splitting of input data and the joining of output data from the jobs is often handled by a wrapper around the job that handles the splitting dynamically when the job is submitted and retrieves the individual data sets after each job has completed.

The second aspect of data management is during the job execution by itself. The job needs to access data that may not be available on local storage. Several solutions are available:

- Data is stored on network-accessible devices and jobs work on the data through the network.
- Data is transferred to the execution node before the job is executed such that the job can access the data locally.

While a grid-based environment may offer many advantages, any given application may not necessarily benefit from a grid. For example, some personal productivity applications are tightly coupled with a users interface and do not consume a large amount of computing resources. Running them on a grid may not provide significant benefits. However, other applications may be very suited for exploiting a grid.

If we take a parochial view of the grid as an environment that provides access to vast amounts of computing power, one of the simplest concepts for grid utilization is to be able to run an application somewhere else when your own machine is too busy or otherwise does not have the required resources. Almost any kind of application can be executed in a grid environment this way. You may not see spectacular performance gains unless the machine it runs on is much faster than the machine you usually use.

Applications that can be run in a batch mode are the easiest to execute on other resources within the grid. Applications that need interaction through graphical user interfaces are more difficult to run on a grid, but not impossible. For instance, they can use remote graphical terminal support, such as X Windows or other similar capabilities. In subsequent sections of this chapter, we discuss many considerations for applications that are CPU intensive or have various requirements associated with data access or sharing. These number-crunching types of applications have historically gained efficiencies by running in a cluster environment or more recently in a grid (that some consider a distributed cluster). However, with advances in grid middleware and the economic incentives to run more typical business applications on virtualized resources, there is a trend towards understanding how these business applications can be implemented (or modified) to take advantage of the various resources provided by a grid computing environment.

4.11 Grid User Roles

Several years ago one person would install 3rd-party software, adapt it for use in a specific field domain, and then use the software for their own research. As the adoption and deployment of Grid infrastructures matures though, it is *inevitable* that we see a partitioning of roles across groups of individuals and the emergence of specialisations within the community as well.

An example of roles division can be the following:

- End-users (domain-specific), e.g. scientists and researchers
- Application developers
- Grid administrators

This division of roles could have huge benefits in e-science projects and, in general, would bring people from different fields to the same table. Note that this paradigm is *not* absolute. According to others, the roles would differ in their structure but they all agree on the need of different roles.

End-user's Perspective

What mainly drives a typical end user are issues that arise from his/her application domain itself. For example, a biologist is interested in identifying relationships between cancer mechanisms and genes. End users (scientists, researchers, etc.) should be able to carry out experiments and applications without needing to understand detailed aspects of the underlying grid technology. In other words, end users are more interested in running applications without having the obligation to be *grid aware*. The key out here is *transparency*. Applications should be grid-enabled without changing the behaviour from the user's point of view. The latter could be wrapped entirely as a Web service for example. The main concerns of such end users include efficiency, effectiveness, ease and flexibility of use, interoperability, responsiveness and interactivity. While the main user needs could be the following: log into the grid, submit jobs, monitor their progress and recovery.

Application developers

"Application developers" perspective arises since we are talking about applications that need to be modified or rewritten in order to run and benefit from a grid. While the grid provides platform neutral protocols for fundamental services like job launching and security, it lacks sufficient abstraction at the application level to accommodate the continuing evolution of individual machines. The application developer, already burdened with keeping up the track of evolution in computer architectures, operating systems, parallel paradigms and compilers, must simultaneously consider how to assemble these rapidly evolving, heterogeneous pieces, into a useful collective computing resource atop a dynamic and rapidly evolving grid infrastructure.

Application developers role is to create, monitor, test, deploy and debug applications that would fulfil scientists needs and in the same time could run efficiently in a certain grid environment. The need for tools that are able to create and debug the grid applications is essential. Unfortunately, today there are relatively few such tools.

Grid administrators

There are roles who possess considerable knowledge of grid technology. Such roles are responsible for the following:

- Installing a suitable grid middleware on a set of machines
- Configuring, managing, monitoring and checking resources
- Enabling applications for the grid

Grid administrators are burdened by keeping up with the evolution of grid middleware and grid technologies since the underlying grid infrastructure must mature and be widely and stably deployed and supported. Moreover, different virtual organizations must possess the appropriate mechanisms for both co-operating and inter-operating with one another.

A grid-enabled version of an application should offer one or more of the following benefits to the end user:

1. be more effective
2. be more efficient (higher throughput/speedup)

4.12 Framework

There are number of paradigms and platforms, such as message passing (using MPI or PVM), for developing and executing parallel applications on distributed systems. However, using such programming paradigms/platforms for creating parallel/distributed applications involves significant development effort and is time consuming. These approaches are effective for tightly coupled systems such as MPP (Massively Parallel Processing) machines or on loosely coupled but controlled systems such as clusters. The inherent challenges in grid computing environments such as the load variability, high network latencies and high probability of failure of individual nodes make it difficult to adopt a programming approach which favours tightly coupled systems.

The MATLAB application we worked on, in this thesis, is considered to be ideal for the case mentioned above as it allows the division of the application into independent tasks just by parameterizing the input data. By achieving the latter, the application can be easily deployed on grids using grid middleware such as Globus Tooklkit. For the last 3 years, we have worked with Globus Toolkit and we have much experience in installing, deploying, configuring, transferring files, submitting jobs, etc. That was very helpful in running the grid-enabled application on the grid testbed established at the Technical University of Crete. On the other hand, we do not have any experience in running applications using MPI. Decomposition, development and debugging of applications using MPI can take considerable effort. Moreover, load balancing is often difficult when using MPI. Note that, one of the basic ideas in grid computing is the load balance (recall Chapter 1 for more information).

4.13 Our Work

The approach to build a grid-enabled application either from scratch or based on existing solutions adds a wide range of issues for problem analysis, application architecture, and design. The previous sections have provided an overview of the issues to consider for any grid application. Specifically, we look at the characteristics of applications themselves. We provide guidance for deciding whether a particular application is well suited to run on a grid. Some of these items may not apply for every project. Some aspects are familiar from other application development projects and are not elaborated on in depth. Others that are new aspects due to the nature of a grid application are provided with greater detail.

In this section we introduce an application that runs under MATLAB. It runs on one machine and in a serial flow. We try to grid-enable this application based on what we presented in the previous sections. In other words, we first study the application's ability to be parallelized, i.e. see if we can divide the application into independent (parallel) tasks/jobs and run each task/job on a different machine. We had to figure out what was the best job type (batch, standard, etc.) in order to run the application under the grid. Since we use Globus, as the underlying middleware, our application runs in *batch mode*. We considered the programming language the application should be written in.

In Section 4.5 on page 56 we introduced some key considerations which, more or less, included portability to a variety of platforms and run-time libraries. Java in this case would provide an advantage over C/C++ since Java could run under Linux and Windows. By that the portability feature would be achieved. Since MATLAB has the ability to produce stand-alone applications in C/C++ and in Java, we applied both languages on our application. Both stand-alone applications, produced by MATLAB, require run-time libraries to be available. We addressed all the required libraries needed. The next step was to take into account some environmental factors that may affect the application's operation such as the OS, memory size, DLLs, run-time environments and any other factors. We check for these prerequisites before running our application in order to avoid too many restrictions.

The next step was to cover job topology issues as described in Section 4.7 in page 59. We studied where the jobs can run, determine if a node is suitable to run an individual job, how to associate jobs with the essential data and other topology-related considerations. Section 4.8 consider various ways to pass the data input/output while Section 4.10 refers to the input/output data management. We, in our application, make use of the GridFTP service in order to meet the data input/output passing and management issues. We describe all the above in details in the following subsections.

First, we describe the MATLAB application and after that we face all the issues described above.

4.13.1 MATLAB application

The Signal Processing team, at the Technical University of Crete (TUC), worked on the problem *Children with Epilepsy.*

More precisely, the application is based on EEG data files (normal children and children with epilepsy). These EEG data files are processed as an input into MATLAB in order to be processed. Wavelets are computed over these EEG and finally a classification method is applied (linear, quadratic, etc.) on the wavelets. As output, classification error is derived.

The data files are first divided into normal and epilepsy (N for normal and E for epilepsy) and second into tasks. These tasks are actually different tests done over the EEG files (see Appendix A for more details on the data). There are eleven different tasks/tests. Each task/test consists of 40 EEG files.

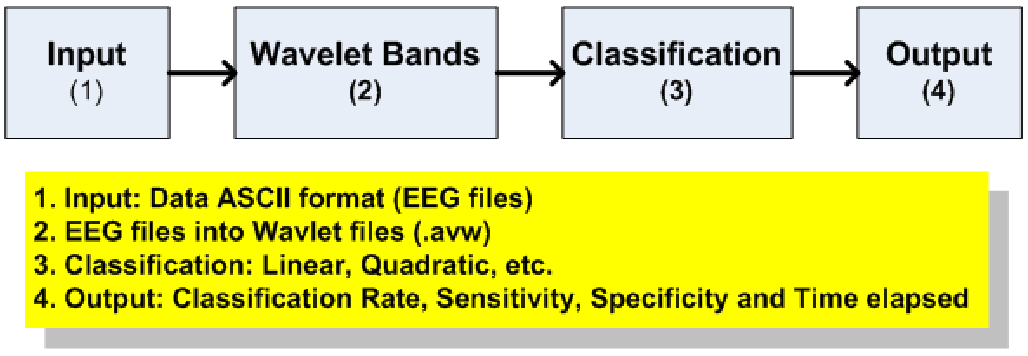

Input
(1)
Wavelet Bands
(2)
Classification
(3)
Output
(4)
1. Input: Data ASCII format (EEG files)
2. EEG files into Wavlet files (.avw)
3. Classification: Linear, Quadratic, etc.
4. Output: Classification Rate, Sensitivity, Specificity and Time elapsed

Figure 4.3: *Application Stages*

Each task/test undergoes the following steps:

1. Wavelet bands are created from the EEG files (This can be done only once)

2. Statistical tests are done over these WB

3. Find the significant channels

4. Check the filtered channels for significant regions averaging

5. Perform classification (linear, quadratic, etc.)

Input

EEG Files per task → Wavelet bands

Output

1. Classification score
2. Sensitivity
3. Specificity
4. Time elapsed in seconds

In detail, the following steps take place:

- 1st step is to create ".avw" files and their combinations. For every EEG file an ".avw" file is created.
- 2nd step is to combine all the ".avw" files to create 6 wavelet files (based on the number of bands used in the simulation).
- 3rd step is to work on the 6 wavelet files and the initial EEG files to perform classification.
- 4th step is to apply classification.

Statistics

- Time for creating ".avw" files and their combinations: $11mins/task$
- Time for classification method: $3mins/task$
- Total Time/task = $14mins$
- Total Time over all tasks = $154mins = 2.5hours$

4.13.2 Grid-enabling the application

Previously we talked about the MATALB application. We presented the basic run steps, input/output and some statistics regarding the application run.

We summarize our contribution to the following:

1. Ability to be parallelized

2. Job type
3. Programming languages
4. Environmental factors
5. Job topology
6. Data input/output passing and management

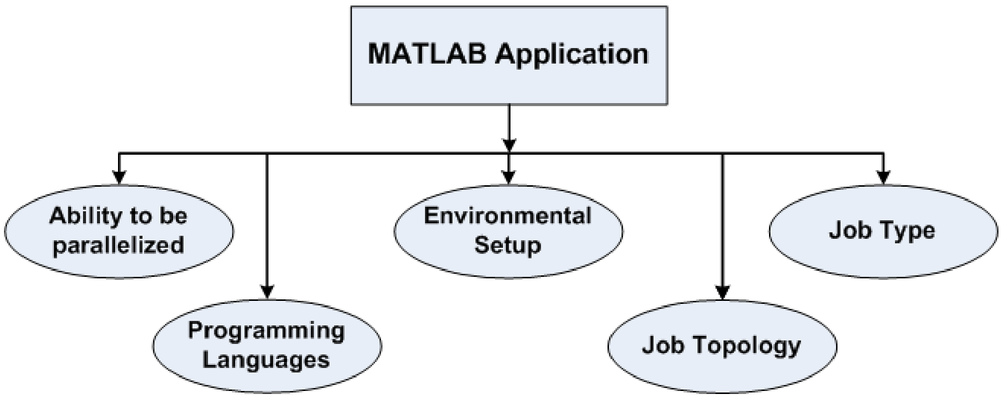


Figure 4.4: *Grid-enabling Issues*

In the following we talk, in details, about each step.

Ability to be parallelized

The MATLAB application we are considering to *parallelize*, as mentioned before, runs over 11 different tasks/tests. These tasks are *independent* of each other, i.e. none of the 11 input tasks depend on another's output task. Each task has its own input/output data. The latter means that the 11 tasks could be divided into independent units (jobs) where each job can run on a different machine. This would result in speeding-up the application by **11 times** assuming enough resources are available.

Having 11 independent jobs would need 11 programs/scripts in order to be launched in a grid environment. Each program/script will be associated with a single job. Creating a program/script for each job is time consuming and adds effort to the application's developer. Due to that, we created a **single** program/script that would apply for all the 11 jobs. The latter was achieved by modifying the MATLAB code in such a way that it could identify each task it's going to work on (Each task is identified through its input file names, e.g. $xxxx - Task01$, $xxxx - Task02$, ..., $xxxx - Task11$).

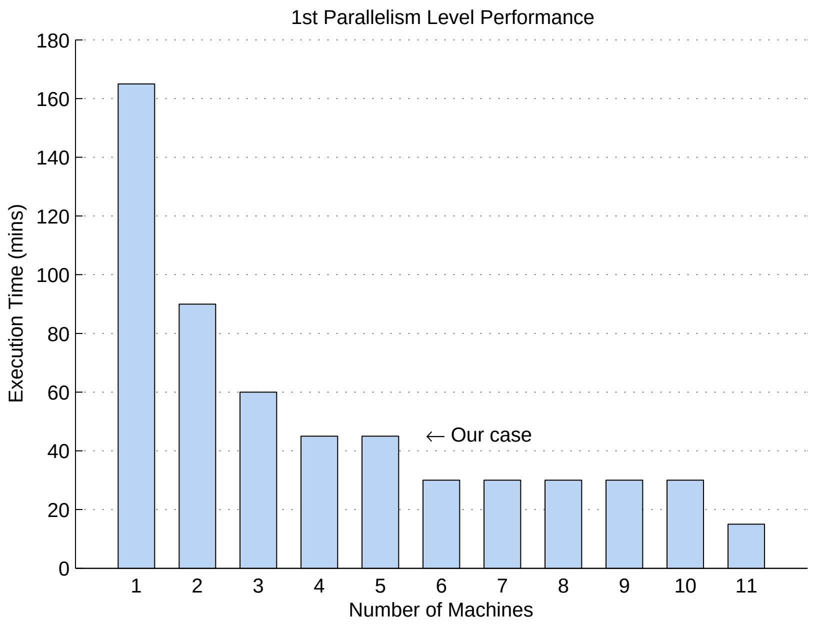


Figure 4.5: *Execution Time among Different Available Resources*

As mentioned above the application needs **2.5** hours in order to finish. Parallelizing the application, as described above, would need only **15** minutes if there are 11 machines available. Figures 4.5 & 4.6 shows the execution time and the speedup, among available resources, respectively.

What we introduced so far was based on *all* the tasks the application runs over. But what about the tasks themselves? Is there any chance or possibility that they can be parallelized?

Each task consists of 40 files. The application initially processes these files and converts each one of them into a wavelet file. This procedure is independent for each file. Moreover, this procedure takes a lot time (approximately 12 minutes, i.e. 80% of the total time). Applying the same case as before and assuming enough resources available, the speedup would increase much more. Figure 4.7 shows the speedup up to 5 machines since the grid testbed we have consists of 5 machines.

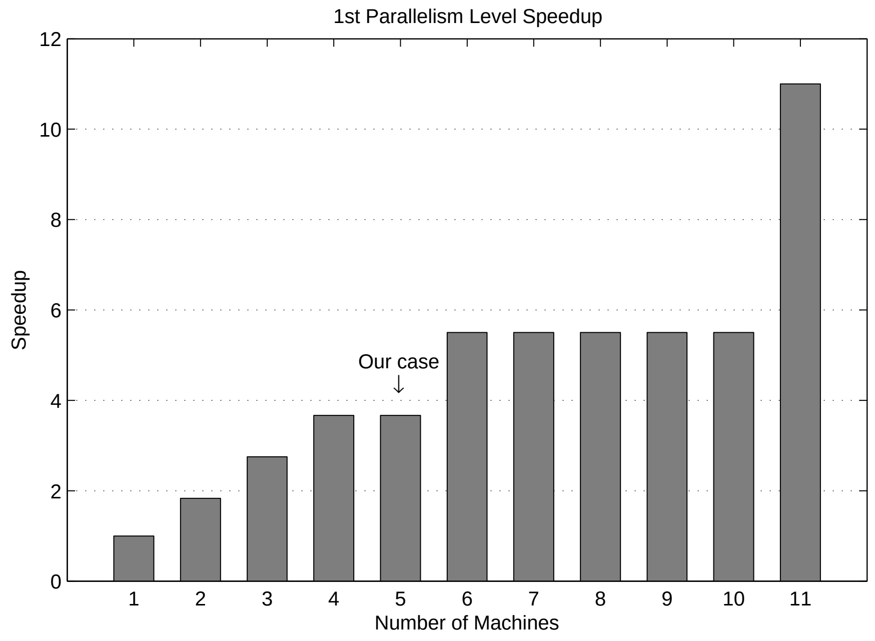


Figure 4.6: *Speedup for the First Parallelism Level*

According to what we mentioned previously, two levels of parallelism are introduced till now (See Figure 4.8).

Several issues arise at this point:

- Does a single program have the ability to perform both parallelism levels?
- What if the program runs in different grid environments?
- If a single program would achieve both parallelism levels, what overhead will be introduced?
- Are there any tools (Globus?) that would perform both parallelism levels automatically?

The issues mentioned above are **case-dependent**. We are going to discuss those issues as much as possible based on our case.

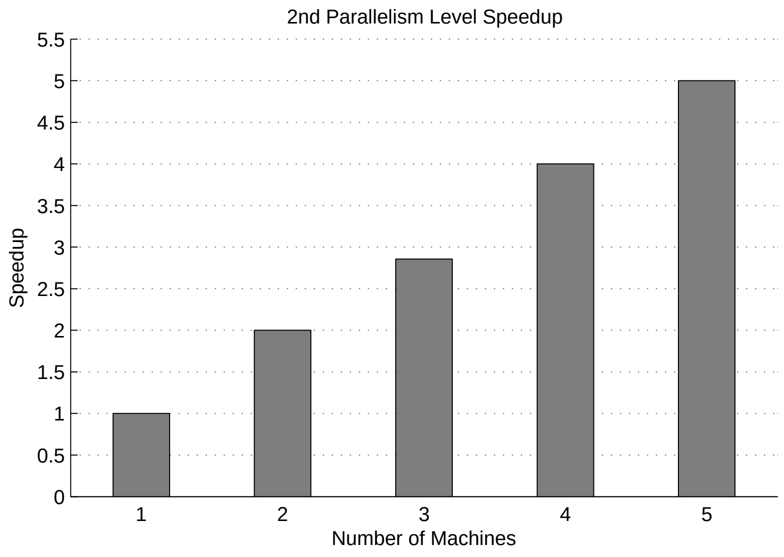


Figure 4.7: *Speedup for the Second Parallelism Level*

The initial non-grid-enabled application was designed to run over the tasks one task at a time. We performed the first parallelism level by creating a single program/script as described above. Accomplishing that, the program could run per task without the need to stall at any point. The only overhead in this case would be passing the input data to the nodes where the program will launch. Note that, if the input data is already allocated before launching the jobs, there would be no overhead. Output data in our case is small in amount and therefore it could be retrieved via the "*stdout*" (see Section 4.13.2 on page 80). Assuming enough resources are available, the speedup in this case would be **11 times**. Since we have established 5 machines in our grid testbed, the speedup would be 3.6 **times** (see Figure 4.5).

When it comes to the second parallelism level, things are rather more complicated and difficult. The second parallelism level can only be applied at stage 2 of the MATLAB application (recall Figure 4.3 on page 68). That means that the program should stall until this stage finishes before proceeding. A certain mechanism should exist that knows when to stall and when to proceed. In order to do this we had to split the program into sub-programs. We created a sub-job that is responsible for parallelizing and a sub-job responsible for proceeding the run. Moreover, since the input data differs in name for each task, the application developer has to create 40 different programs/scripts in order to achieve such way of parallelism. To avoid this we

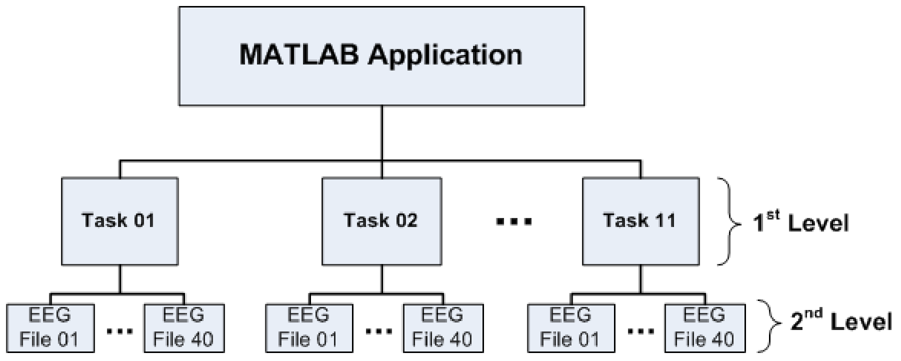


Figure 4.8: *Parallelism Levels*

renamed the input data in such a way that the program could identify between them (e.g. $xxxx - Task05 - File01$, $xxxx - Task05 - File02$, ...). Since this parallelism level has to pass input/output data much more than the previous level, we expect to have much more overhead.

Taking a closer look at the input files, we noticed that they consist of 32 columns and 4096 rows. The columns represent the number of channels (see Appendix A for more information). The MATALB application processes each channel independently which means that parallelizing could be applied also in this case. As a result, we have a third parallelism level. The program should stall until the parallel part finishes before it can proceed. Same problems appear here also where a single program could not achieve such level of parallelism. Sub-jobs should be created in turn. Figure 4.9 shows the third parallelism level speedup.

Unfortunately, there are no tools that would perform all the parallelism levels automatically. It's the application's developer duty to create programs/scripts that would manage the parallelism levels. It's the application's developer duty to reduce as much as possible the sub-jobs interdependencies. This is not always feasible or easy. A program that is designed to run under large grid environments would have severe performance problems when dealing with smaller grid environments. The application developer has to keep in mind that although (s)he would have the ability to create such a program, the total overhead produced would overcome the total speedup. In that case no benefit will be gained.

Previously, we mentioned a parallelizing process where independent jobs

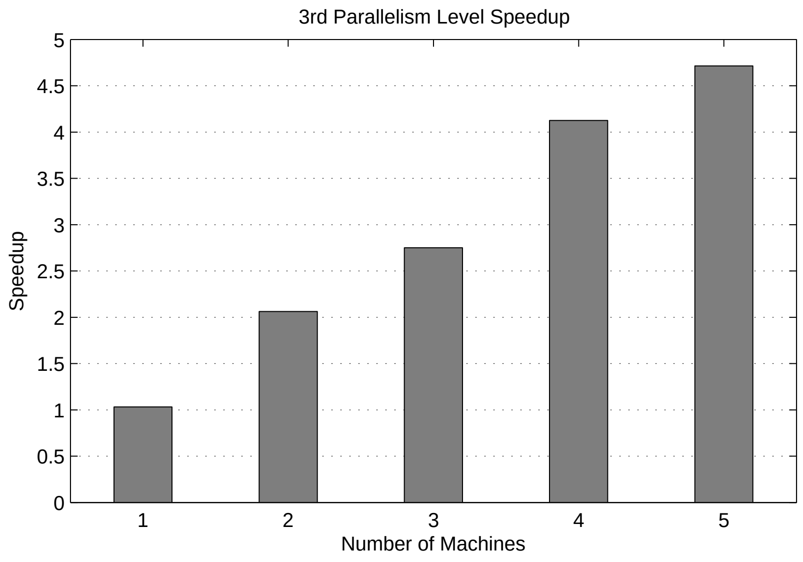


Figure 4.9: *Speedup for the Third Parallelism Level*

can run on different machines. Another parallelizing process is the Message Passing Interface (MPI). MPI is a parallelizing process where jobs can run on multi-CPU machines. Note that MPI is implemented in C and Fortran programming languages. MPI is not an easy task. It's time consuming and involves much development effort. It requires application's underlying algorithms to be re-written in order to be able to run on multi-CPU machines.

Recalling back the second parallelism level, we mentioned that it takes 80% of the total time. Since MPI could be applied here, we could predict the speedup using Amdahl's law. The maximum theoretical speedup considering 40 processors are available is equal to:

$$Overall\,Speedup = \frac{1}{(1-p)+\frac{p}{s}} = \frac{1}{(1-0.8)+\frac{0.8}{40}} = 4.5 \qquad (4.1)$$

where p is the fraction of a program that is enhanced & s is the speedup of the enhanced portion

Concerning each input file independently now, the portion that can be enhanced, in this case, reaches to 99%. Applying MPI in this case would speedup the application execution even more. According to Amdahl's law, the maximum total speedup in this case would be:

$$Overall\,Speedup = \frac{1}{(1-p)+\frac{p}{s}} = \frac{1}{(1-0.99)+\frac{0.99}{32}} = 24.4 \qquad (4.2)$$

MATLAB latest version, specifically R2007b, supports multi-threaded computing. But, at the meantime this version supports only "Basic Linear Algebra Subroutines" (BLAS) library, such as matrix multiply. As mentioned before, additional MATALB code should be re-written to be able to run on multi-CPU machines. Due to the reasons mentioned above, we do not implement MPI in our work.

Job type

The grid resource manager is concerned with resource assignments as jobs are submitted. It acts as an abstract interface to the heterogeneous resources of the grid. The resource management component provides the facilities to allocate a job to a particular resource, provides a means to track the status of the job while it is running and its completion information, and provides the capability to cancel a job or otherwise manage it. In Globus Toolkit, the remote job submission is handled by the Grid Resource Allocation Manager (GRAM). Within a grid environment, applications that can be run in batch mode are the easiest. GRAM uses the `globusrun-ws` command in order to run batch jobs.

globusrun-ws Command-line

For implicitly, just run the following:

```
user:/> globusrun-ws -submit -f job.xml
```

The `globusrun-ws` command is typically passes an RSL string that specifies parameters and other properties required to successfully launch and run the job. RSL stands for Resource Specification Language. RSL is a language used by clients to specify the job to be run. All job submission requests are described in an RSL string that includes information such as the executable file; its parameters; information about redirection of stdin, stdout, and stderr; and so on. Basically it provides a standard way of specifying all of the information required to execute a job, independent of the target environment. It is then the responsibility of the job manager on the target system to parse the information and launch the job in the appropriate way. The syntax of RSL is very straightforward. Each statement is enclosed within XML tags.

Since we use Globus Toolkit, the job type we use in our application is batch mode. The job we use to submit is written in RSL as mentioned before.

Programming languages

MATLAB programs in general run within the software MATLAB and in form of "*.m*" files. We stated before that the job type we are going to use for our application is the batch mode. In order to run MATLAB programs out of the MATLAB software box, we need to convert these programs into applications and libraries that they can be distributed to end users who do not have MATLAB installed.

MATLAB provide the "MATLAB Compiler" that supports all the features of MATLAB, including objects, private functions, and methods. Someone can compile M-files, MEX-files, or other MATLAB code. Using MATLAB Compiler you can generate the following:

- Standalone C and C++ applications on UNIX, Windows, and Macintosh platforms
- Standalone JAVA applications on UNIX, Windows, and Macintosh platforms

For our application we created two different standalone applications. One application compiled using C language and another application based on Java language. Java standalone applications have the advantage of running on any platform. While C standalone applications complied under Linux platform would only run under Linux platforms. The same happens when using MATLAB Compiler under Windows.

When running Java applications, the user has to keep in mind issues such as setting the CLASSPATH at the target machine when running the application. Another issue that arises here is setting the suitable shared libraries for Java language. We will talk about this in the next part.

MATLAB Compiler can be used via a GUI. Since MATLAB Compiler wraps MATLAB files in packages, it's not a difficult task to achieve but one has to be *careful* that all the required MATLAB runtime files are added to the package.

Environment setup

There are several factors that could lead a grid application to face restrictions. A large number of restrictions could mean more complicated enablement as well as limiting the number of possible nodes on which the job will be able to run. That's why it's preferred to check for such requirements in order to avoid too many restrictions when executing the jobs.

Based on our application, the environmental factors could be summarized to the following:

- OS
- DLLs and runtime environments
- Application version

Memory size and storage requirement in our case is not an issue and therefore they are not listed here as factors that could restrict our application state of execution. Morever, hardware is not an issue in our case.

The grid testbed deployed in the Technical University of Crete consists of 5 machines running under SuSE Linux. The middleware was used to deploy the grid infrastructure is the Globus Toolkit. Globus Toolkit has wide use under Linux platforms. Under Windows platforms, Globus Toolkit is available with minimal features (Java-based Globus Toolkit).

Path specifications and wild card attributes vary based on the underlying OS. For example, slash ("/") is used to specify a file's path under Linux platforms where backslash ("\") is used under Windows platforms. Our application accesses input files from storage system which means that the programs/scripts, we are creating, should be aware of this feature. This leads to the need for creating different programs/scripts to run under different platforms. The latter can be solved using the following MATLAB commands: `isunix`, `ispc`, `icmac`. The command `isunix` checks if the unerlying OS is UNIX, `ispc` checks if the underlying OS is Windows and `ismac` checks if the underlying OS is Macintosh. Using the previous MATLAB commands to check the underlying OS, just before the application starts running, will lead to a single program/script thus reducing the path specification restriction.

When packaging and distributing applications and libraries that are generated by MATLAB Compiler, one must include the MATLAB Component Runtime (MCR) as well as a set of supporting files generated by MATLAB

Compiler. You must also set the system paths on the target machine so that the MCR and supporting files can be found. An application or library generated by MATLAB Compiler has two parts: a platform-specific binary file and an archive file containing MATLAB functions and data. For an application, the binary file consists of a main function, and for a library the binary file exports multiple functions that can be called by users of the library.

Setting the MCR libraries into the system paths on a target machine was *not* an easy task. The libraries that should be set are a large set of libraries and missing out one library would lead to the failover of the application. But once the application is tested on one target machine, setting the system paths on other target machines would be an easy task.

Another restriction would be the MATLAB and the MATLAB Compiler versions. Since MCR comes with MATLAB, someone should be careful when compiling MATLAB files using a new MATLAB version and trying to run the standalone application on a target machine where an older MCR version is installed.

We should mention here that the MATLAB application was first implemented under Windows platform. For the same reasons mentioned before, we had to modify the application in order it can run under Linux platforms. It was a difficult and a hard task since we had to check all the MATLAB files for platform dependencies. Allocating the appropriate wild card attributes for Linux, for example, was not easy at all.

Job topology

The grid-enabled application we created was implemented and tested at the Technical University of Crete. The grid testbed established consisted of 5 machines, running SuSE Linux platform and Globus Toolkit 4.0.3 middleware. Moreover, the application was tested remotely at the University of Plymouth (UoP). The test took place to check the application's integrity. We ran our application on one machine only due to lack of available machines at UoP.

Since we worked with Condor broker we had to ensure that an appropriate target resource is selected. This requires that the application accurately specifies the required environment (operating system, processor, speed, memory, and so on). The more the application developer can do to eliminate specific dependencies, the better the chance that an available resource can be found

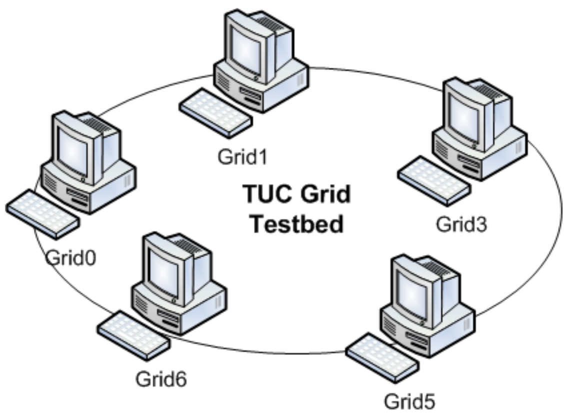


Figure 4.10: *TUC Grid Testbed*

and that the job will complete. But we, in our grid testbed, ensured that all the machines were capable of running the application. We made sure that GRAM or any other job scheduler/broker (condor was also used) would always find these resources in order to run the application.

Globus Toolkit and its basic components (GridFTP and GRAM) were just enough in order to run the application. Of course MCR was installed on all machines since it's a prerequisite when dealing with MATLAB standalone applications.

GRAM provides mechanisms to query the status of the job as well as perform operations such as cancelling the job. The application may need to utilize these capabilities to provide feedback to the user or to clean up or free up resources when required. For instance, if one job within an application fails, other jobs that may be dependent on it may need to be cancelled before needlessly consuming resources that could be used by other jobs.

Data input/output passing and management

Passing and managing the essential input/output data was achieved using GridFTP service. GridFTP facility provides secure and reliable data transfer between grid hosts. Its protocol extends the File Transfer Protocol (FTP) to provide additional features including: GSI, TCP socket buffer size negotiation, Parallel data transfer and others (recall Chapter 2, for more info on GridFTP).

GridFTP is used to pass all the essential input form storage systems to target machines in order to execute jobs. GridFTP is also used to transfer the output data back to the user. But For large datasets, it is not practical and may be impossible to move the data to the system where the job will actually run. Using data replication or otherwise copying a subset of the entire dataset to the target system may provide a solution. Since our application does not use large datasets, this would be not an issue. We simply use GridFTP as we mentioned before.

If a job returns a simple status or a small amount of output, the application may be able to simply retrieve the data from stdout and stderr. However, the capturing of that output will need to be correctly specified in the RSL string that is passed to the `globusrun-ws` command. If more complex results must be retrieved, the GridFTP service may need to be used by the application to transfer data files.

Security, reliability, and performance issues should be kept in mind when moving data across the Internet or another WAN. A logic has to be build to handle situations when the data access may be slow or prevented.

4.13.3 Grid Portal

Tranparency is an important issue for the end users. Transparency could be acheived by using user interfaces based on Web browsers. They are called "*Grid Portals*".

Grid portals can deliver complex grid solutions to users wherever they have access to a web browser running on the Internet. Hence, grid portals have been proven to be effective mechanisms for exposing computing resources and distributed systems to general user communities without forcing them to deal with the complexities of the underlying systems. The interfaces to the services and resources available should be intuitive and easy to use.

Where most of the Grid portals expose functionality like launching jobs for remote execution or retrieving remotely-stored data, grid portals can also include application specific interfaces customized for a particular domain. We should point out here that security gains prominence in Grid portals largely because of the nature of the Grid resources they expose.

The primary requirements for a grid portal system from a user's point of view involve access to Grid Services. These include:

- Security services: Only authenticated users could log onto the grid portal
- Remote file management: Users should be able to access remote files
- Remote job management: Users should have the ability to submit jobs to the grid for execution

We, in this thesis, created a grid portal that meets the above requirements. Specifically, we created a login module based on HTTPS and SSL. Users can log into the portal using their X.509-based certificates. A GridFTP API was created to access remote files. A GRAM API was also created to submit jobs for execution. Using our grid portal, the initial creators of the MATLAB application could now run the grid-enabled version of the application more *transparently*.

4.14 Acknowledgments

We would like to express our gratefulness to both Mrs. Lingfen Sun and Mr. Pin Hu from the University of Plymouth, England for the assistance in running our grid-enabled application between our universities.

Chapter 5

Conclusions

The Grid is emerging as a new mean for solving problems in science, engineering, industry and commerce. A computing node or even more, an entire single site can no longer meet all the resource needs of today's demanding applications. Moreover, using distributed resources can bring many benefits to application users.

During this thesis, we have extensively studied GridFTP service. Specifically we examined its various features (TCP socket buffer size and TCP parallel streams) that lead to its throughput maximization. We analyzed the throughput behaviour under five different parameters (File-size, RTT, Bandwidth, TCP socket buffer size and TCP parallel streams) and tried to find the best-fitting model (Linear Regression, LMMSE and Exponential Non-linear Regression) that suits the throughput. We observed that the Exponential Non-linear Regression better suits the GridFTP throughput. In conclusion, we saw that an active tool for estimating RTT and bandwidth could benefit GridFTP service. Therefore, we have studied bandwidth estimation techniques, i.e. packet pair/train technique. We have tested already existing tools but most of these tools seemed to function only in low-scale networks and need a lot of testing to get them work good. We have proposed an approach based on Java Networking for large-scale networks. We managed to estimate bandwidths efficiently up to $80Mbps$.

Also, in this thesis, we worked on a MATLAB application. We tried to examine its ability to be parallelized in order to run under a grid environment. Since Grid Computing was initially introduced to meet heavy-computational applications, our goal was to run this application in a parallel way and thus run quickly in a shorter time possible. This application was tested locally at the Technical University of Crete and remotely at the University of Plymouth. The application in default-run needs 3 hours to terminate. After grid-enabling the application, it has the ability to terminate in 15 minutes assuming enough resources available if applied on a the first parallelism level. We managed to extract three parallelism levels. There are many factors though, to consider in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. Furthermore, there are no practical tools for transforming arbitrary applications to exploit the parallel capabilities of a grid. However, automatic transformation of applications is a science in its infancy. This can be a difficult job and often requires top mathematics and programming talents, if it is even possible in a given situation. New computation intensive applications written today are being designed for parallel execution and these will be easily grid-enabled, if they do not already follow emerging grid protocols and standards.

Grid computing is accessed through what is called a middleware. A middleware consists of various components (tools, programs, etc.) Installing and configuring a certain middleware is not an easy task. In contrast, it requires expert knowledge in order to put a middleware and its components to work and function properly. Still today, this is all done via command-prompt. Besides the installation/configuration part, a user has to memorize and run a lot of command-line instructions in order to invoke a service, such as transferring a file using GridFTP or submitting a job using GRAM. All this would "*distance*" the user form his/her initial goals. For that reason, many tired and still trying to create a more friendly user interfaces. They are called Grid Portals. We, during this thesis, have created a mini grid portal. We have had the ability to comprehend and understand the requirements and the specifications of a grid portal. We created different APIs. An API for querying/modifying a database, an API for transferring files and an API for submitting jobs.

Appendix A

Data Description

A.1 Children 2004

Below are some features

- *Filters*: 0.1 200 Hz
- *ADC*: 400 Hz, 12 Bits
- *Gain*: (f.s.d.) 500 V

We have normal and epilepsy children 1 : 1 for comparison. Every EEG file is described by its name. For example file $C06N08$ means that this file contains recordings from child N^o06, which is Normal (Children with epilepsy have E), 08 is the test number. The tests are described below.

Test Description

The tests are described below:

1. Child observes a star [01 or A].
2. Child watches geometrical structures. 500 msec x 30 [02 or B].
3. Child watches one digit numbers (0, 1, 2) msec x 30 [03 or C].
4. Child adds one digit numbers , $n = 18$, right-wrong answers by turns : dx or sin [04 or D].
5. Child compares two digit numbers $n = 16$ (Uses right or left hand if greater number is right or left [05 or E].

6. Child subtracts numbers $n = 16$: Right or wrong by turns: dx or sin. two digits numbers subtractions or two digits minus one digit. [06 or F].
7. Letters 500 msec x 30 [07 or G].
8. Phonologic : vowel or consonant before K in a word, $n = 14$, by turns: right hand or left hand [08 or H].
9. Semantic: Animal greater than dog, vegetarian, $n = 14$, by turns: right hand or left hand [09 or I].
10. Kanisza, 500 msec x 30 [10 or K].
11. Fractals [11 or L].
12. Rest, eyes closed [12 or M]. **Not included**.

Channels

Table A.1 shows the 32 channels where the EEG files are based:

1	Γ1	VEOGL	17	A1	FP1
2	Γ2	FP2	18	A2	F3
3	Γ3	F4	19	A3	C3
4	Γ4	C4	20	A4	P3
5	Γ5	P4	21	A5	O1
6	Γ6	O2	22	A6	F7
7	Γ7	F8	23	A7	T3
8	Γ8	T4	24	A8	P7
9	Δ1	P8	25	B1	GND
10	Δ2	Cz	26	B2	Fz
11	Δ3	Pz	27	B3	FC3
12	Δ4	FC4	28	B4	FT7
13	Δ5	FT8	29	B5	CP3
14	Δ6	CP4	30	B6	TP7
15	Δ7	TP8	31	B7	FCz
16	Δ8	CPz	32	B8	Oz

Table A.1: *Channels*

Note Channel 1 (eye movements) and 25 (Ground) should be removed before analysis.

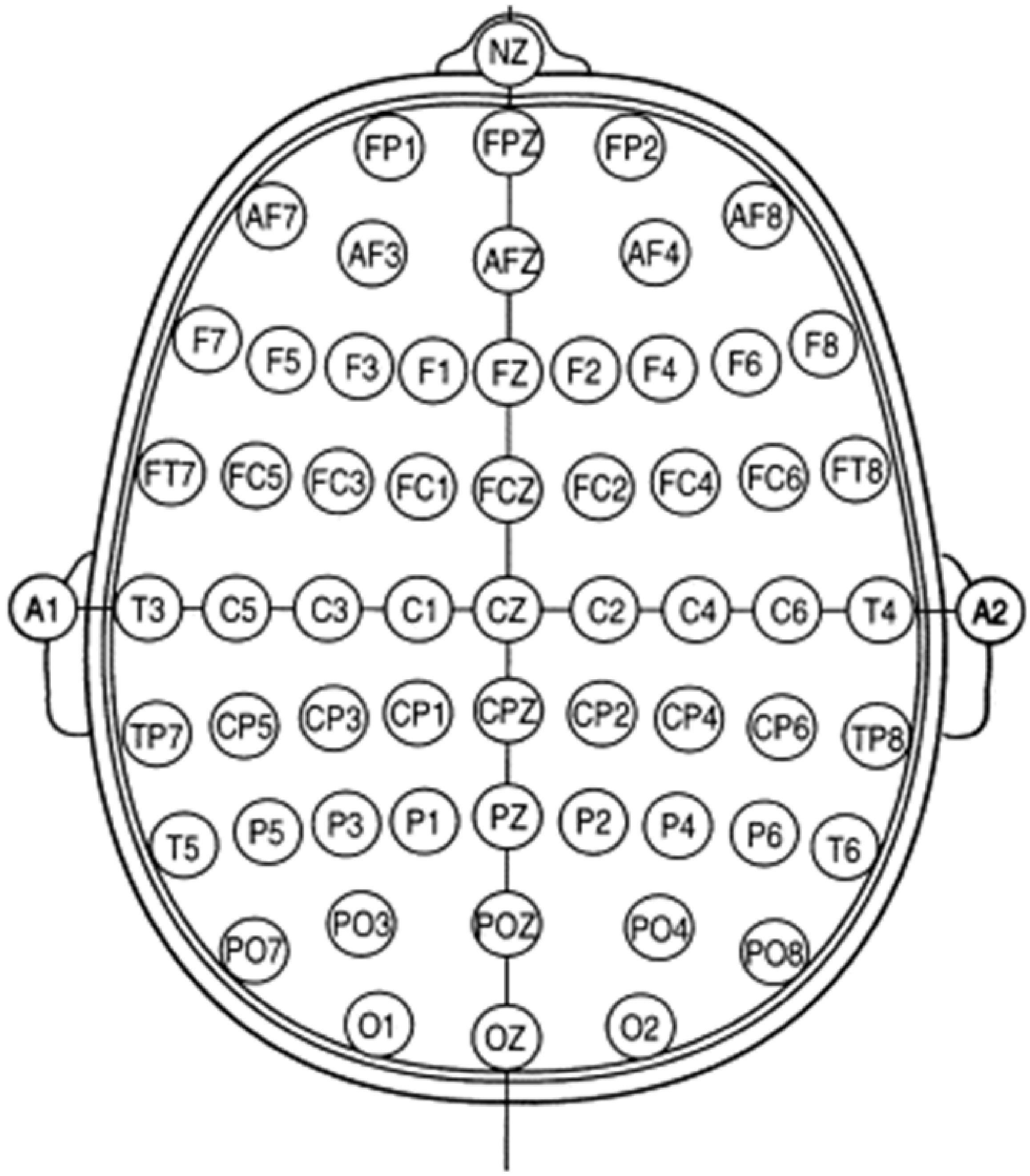


Figure A.1: *Channel Locations*

Appendix B

Multiprocessing in MATLAB

B.1 Overview

MATLAB supports two types of multiprocessing: *implicit* and *explicit*.

B.1.1 Implicit Multiprocessing

Characteristics of implicit multiprocessing:

- Runs multiple threads on a single machine, most often using one thread per processing unit.
- Requires a multiple CPU (multiprocessor or multicore) system.
- Speeds up element wise computations such as those done by the sin and log functions, and computations that use the Basic Linear Algebra Subroutines (BLAS) library, such as matrix multiply.
- Does not require any changes to your MATLAB code.
- Works behind the scenes to take advantage of the processing units available to you. It does this by multithreading the computationally-intensive math library functions that you use in the course of your MATLAB session.

Enable implicit multiprocessing with the MATLAB Preferences Panel to enable or disable, or to set the number of threads to be used. You can change the maximum number of threads programmatically using the `maxNumCompThreads` function.

B.1.2 Explicit Multiprocessing

Characteristics of explicit multiprocessing:

- Runs separate processes on one or many machines.
- Requires installation of Distributed Computing Toolbox (DCT).
- Speeds up execution of large MATLAB jobs. Enables you to run jobs simultaneously on a cluster of computers, or as several processes on a single machine.
- Requires that you modify your MATLAB code.
- DCT supports programming constructs for distributed arrays and parallel for (`parfor`) loops. It also supports both interactive and batch execution. Enable explicit multiprocessing by installing Distributed Computing Toolbox.

B.2 Implicit Multiprocessing

Multithreaded computation runs in a single instance of MATLAB and generates simultaneous instruction streams on a multiple CPU (multiprocessor or multicore) system. The multiple processors share the memory of a single computer. The work to be processed is implicitly partitioned for execution on multiple threads. In particular, multithreaded computation in MATLAB speeds up element wise computations such as those done by the sin and log functions, and computations that use the Basic Linear Algebra Subroutines (BLAS) library, such as matrix multiply.

If you are using a multiple-CPU system, you can run a demo to see the performance impactsee Multithreaded Computation in the Help browser Demos pane, under MATLAB Mathematics. For information regarding specific functions, search for “What MATLAB Functions Support Multithreaded Computation” on The MathWorks online Support page.

B.2.1 Platform Differences and Multithreaded Computation

The BLAS library used for multithreaded computation differs according to which platform you are using: **Note** On Macintosh PowerPC platforms, multithreaded computation is always enabled for the Accelerate BLAS. To

Platfrom	**BLAS Used**
Windows with Intel processors	Intel MKL BLAS
Windows with AMD processors	AMD ACML BLAS
Linux with Intel processors	Intel MKL BLAS
Linux with AMD processors	AMD ACML BLAS
Macintosh Intel-based	Intel MKL BLAS
MacIntosh PowerPC	Mac Accelerate BLAS
Solaris	Sun Performance Library BLAS

Table B.1: *Platform BLAS Libraries*

enable multithreaded computation for element wise operations, use MATLAB preferences.

B.2.2 Enabling Multithreaded Computation

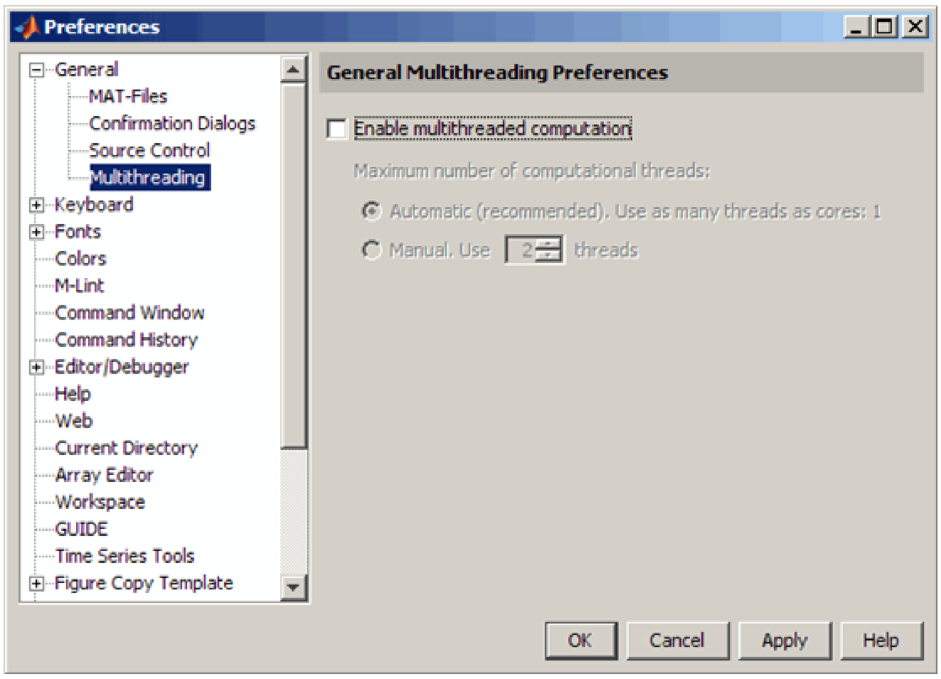
The preference automatically detects the number of CPUs on your system and recommends the number of threads based on that.

Multithreaded computation in MATLAB is disabled by default. To enable it and set the maximum number of threads to use, follow these steps:

1. Select **File** > **Preferences** > **General** > **Multithreading.** The **General Multithreading Preferences** panel opens.
2. On the **General Multithreading Preferences** panel, select **Enable multithreaded computation**.
3. Specify the Maximum number of computational threads. Accepting the Automatic option is recommendedMATLAB automatically sets the value to the actual number of computational cores on your system. Note that if your system uses hyperthreading (where one processor is logically configured as two), MATLAB sets the value to 1.

If you choose **Manual**, enter the maximum number of threads you want to set; use a positive integer not greater than 16. (Selecting a number other than the recommended value might increase performance for some computations, but might decrease performance for others.)

Note You may find that, at certain times, a library function uses a number of threads smaller than what you have specified. This can happen if the



General Multithreading Preferences

[ ] Enable multithreaded computation

Maximum number of computational threads:

[● Automatic (recommended). Use as many threads as cores: 1]

○ Manual. Use 2 threads

Figure B.1: *Multithread Unset*

function finds the specified number of threads to be inappropriate.

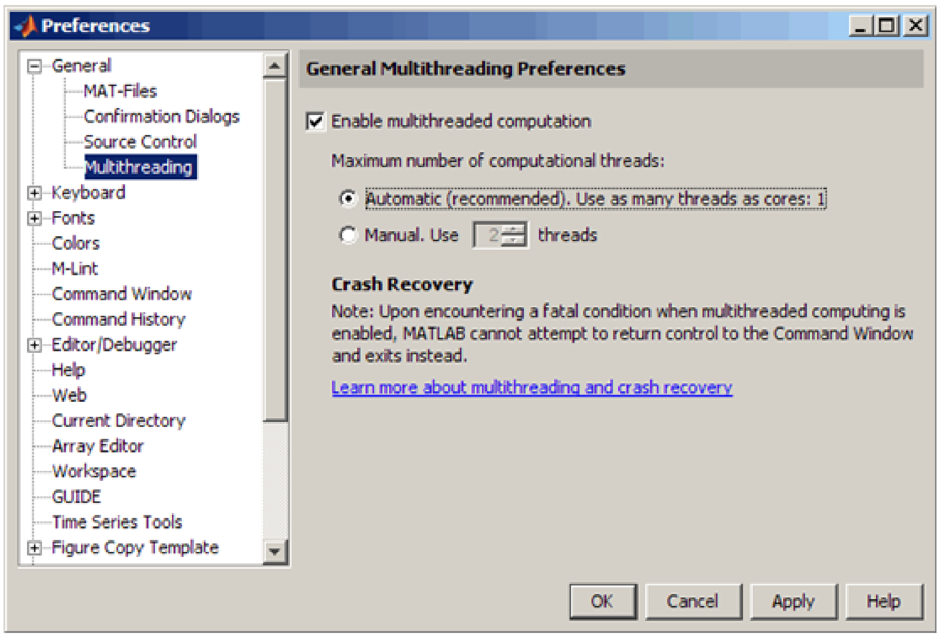
In the event of an abnormal termination with multithreaded computation enabled, MATLAB behaves differently than when multithreaded computation is not enabled.

Making this setting in the **Preferences** panel not only affects your current MATLAB session, but future sessions as well. To disable multithreaded computation, clear the **Enable multithreaded computation** selection and click **OK**.

Note For Macintosh PowerPC platforms, BLAS multithreaded computation cannot be disabled.

B.2.3 Setting the Number of Threads Programmatically

To set or retrieve the maximum number of computational threads from within an M-file program, use the maxNumCompThreads function. You



General Multithreading Preferences

[✓] Enable multithreaded computation

Maximum number of computational threads:

[● Automatic (recommended). Use as many threads as cores: 1]

○ Manual. Use 2 threads

Crash Recovery

Note: Upon encountering a fatal condition when multithreaded computing is enabled, MATLAB cannot attempt to return control to the Command Window and exits instead.

Learn more about multithreading and crash recovery

Figure B.2: *Multithread Set*

can either set the maximum number of computational threads to a specific number, or indicate that you want the setting to be done automatically by MATLAB.

To set the maximum number of computational threads to a specific number `N`, use

```
maxNumCompThreads(N)
```

To have MATLAB set the maximum number of threads, use:

```
maxNumCompThreads('automatic')
```

`maxNumCompThreads` also returns the current maximum number of threads if you call it with an output value:

```
old_N = MaxNumCompThreads(new_N)
```

MATLAB keeps the settings you make using maxNumCompThreads synchronous with your Preferences settings. If you change the maximum number of computational threads by means of the maxNumCompThreads function, MATLAB updates the Preferences panel to agree with the new setting.

Note Setting the maximum number of computational threads using maxNumCompThreads does not propagate to your next MATLAB session. To make this setting carry over to future sessions, use the Preferences panel instead.

B.2.4 Crash Recovery and Multithreading

If MATLAB experiences a segmentation violation or other serious problem when multithreaded computation is enabled, it cannot try to return control to the Command Window. You do not have an opportunity to view a segmentation violation message in the Command Window as you might when multithreaded computation is not enabled. Instead, your platforms vendor, for example, Microsoft or Apple, provides an error dialog box. MATLAB then terminates. Upon the next MATLAB startup after a fatal problem, the "Error Log Reporter" prompts you to e-mail the log to The MathWorks.

B.2.5 Measuring Performance Improvement for a Single Operation

This example uses two threads (defined in the variable `numThreads`) for one sample operation, matrix multiply. You can experiment by increasing the number of threads if your system has more than two CPUs. There are some overhead costs associated with running code the first time, so perform timing comparisons with a second and subsequent runs to remove effects of that overhead.

First, define some parameters and generate random data in variables `A` and `B`.

```
numThreads=2;                % Number of threads to test
dataSize=500;                % Data size to test
A=rand(dataSize,dataSize);   % Random square matrix
B=rand(dataSize,dataSize);   % Random square matrix
```

Next, set the number of computational threads to one and time the operation of interest.

```
oldstate = maxNumCompThreads(1);
C=A*B; % Do not perform timing comparison with the first run
tic;
C=A*B;
time1=toc;
fprintf('Time for 1 thread = %3.3f sec\n', time1);
```

```
Time for 1 thread = 0.074 sec
```

Now, set the number of computational threads to numThreads and time the operation. You can experiment by increasing the number of threads if your system has more than two CPUs.

```
maxNumCompThreads(numThreads);
tic;
C=A*B;
timeN=toc;
fprintf('Time for %d threads = %3.3f sec\n', numThreads, timeN);
```

```
Time for 2 threads = 0.040 sec
```

Calculate performance improvement.

```
speedup=time1/timeN;
fprintf('Speed-up is %3.3f\n',speedup);
```

```
Speed-up is 1.855
```

For more information, read about **Multithreaded Computation** in MATLAB documentation.

Appendix C

MATLAB Compiler

C.1 Overview of MATLAB Compiler

Use MATLAB® Compiler to convert MATLAB® programs to applications and libraries that you can distribute to end users who do not have MATLAB installed. You can compile M-files, MEX-files, or other MATLAB code. MATLAB Compiler supports all the features of MATLAB, including objects, private functions, and methods. Using MATLAB Compiler you can generate the following:

- Standalone C and C++ applications on UNIX, Windows, and Macintosh platforms
- C and C++ shared libraries (dynamically linked libraries, or DLLs, on Microsoft Windows)

Use the `mcc` command to invoke MATLAB Compiler. Alternatively, you can use the graphical user interface for MATLAB Compiler by issuing the following command at the MATLAB prompt:

```
deploytool
```

C.2 How does MATLAB Compiler Work

MATLAB Compiler Generated Application or Library

When you package and distribute applications and libraries that are generated by MATLAB Compiler, you must include the MATLAB Component

Runtime (MCR) as well as a set of supporting files generated by MATLAB Compiler. You must also set the system paths on the target machine so that the MCR and supporting files can be found. An application or library generated by MATLAB Compiler has two parts: a platform-specific binary file and an archive file containing MATLAB functions and data. For an application, the binary file consists of a main function, and for a library the binary file exports multiple functions that can be called by users of the library.

Wrapper Files

To create the platform-specific binaries that you specify, MATLAB Compiler generates one or more wrapper files. A wrapper file provides an interface to the compiled M-code. Wrapper files differ depending on the execution environment.

The wrapper file does the following:

- Performs initialization and termination as needed by a particular interface.
- Defines data arrays containing path information, encryption keys, and other information needed by the MCR.
- Provides the necessary code to forward calls from the interface functions to the MATLAB functions in the MCR.
- For an application, contains the `main` function.
- For a library, contains the entry points for each public M-file function. Users of libraries generated by MATLAB Compiler must call the library initialization and termination routines in their client code.

Component Technology File (CTF)

MATLAB Compiler also generates a Component Technology File (CTF), which is independent of the final target type - standalone application or library - but is specific to each operating system platform. This file, which is named with a `.ctf` suffix, contains the MATLAB functions and data that define the application or library.

C.3 Before You Begin

Before you can use MATLAB Compiler, you must have it installed and configured properly on your system. At a minimum, you must run the following command once after installing a new version of MATLAB Compiler:

```
mbuild -setup
```

If you need information about writing the M-files that you plan to compile, see MATLAB Programming, which is part of the MATLAB product documentation.

C.4 Using the GUI to Create and Package a Deployable Component

Open the Deployment Tool by issuing the following command at the MATLAB prompt:

```
deploytool
```

Use the Deployment Tool as follows to create and package either a standalone application or a shared library:

1. Create a new project.
2. Add files that you want to compile.
3. Set properties for building and packaging.
4. Save the project.
5. Build the component.
6. Edit and rebuild as necessary.
7. Package the component for distribution to programmers or end users.

For more information read MATALB Compiler 4, User's Guide.

References

[1] K. Fukui B. Jacob, M. Brown and N. Trivedi. *Introduction to Grid Computing*. IBM, December 2005. ibm.com/redbooks.

[2] The Globus Alliance. http://www.globus.org/.

[3] Director of Computation Institute Ian Foster. http://www-fp.mcs.anl.gov/~foster/.

[4] RFC 1180 TCP/IP tutorial. http://www.faqs.org/rfcs/rfc1180.html.

[5] Data Management GridFTP service. http://www.globus.org/toolkit/docs/4.0/data/gridftp/.

[6] RFC 793 Transmission Control Protocol. http://www.faqs.org/rfcs/rfc793.html.

[7] RFC 768 User Datagram Protocol. http://www.faqs.org/rfcs/rfc768.html.

[8] Greece Technical University of Crete Chania. http://www.tuc.gr.

[9] RFC 959 File Transfer Protocol. http://www.faqs.org/rfcs/rfc959.html.

[10] Globus Security GSI. http://www.globus.org/toolkit/docs/4.0/security/.

[11] Multi protocol data movement. http://www.globus.org/toolkit/docs/4.0/data/gridftp/rn01re01.html.

[12] W. Feng S. Thulasidasan and M. K. Gardner. Optimizing gridftp through dynamic right-sizing. *In Proceedings of IEEE International Symposium on High Performance Distributed Computing*, 2003.

[13] H. Ohsaki T. Ito and M. Imase. On parameter tuning of data transfer protocol gridftp in wide-area grid computing. *In Proceedings of Second International Workshop on Networks for Grid Applications*, pages 415 – 421, 2005.

[14] H. Ohsaki T. Ito and M. Imase. Automatic parameter configuration mechanism for data protocol gridftp. *In Proceedings IEEE Symposium on Applications and the Internet (SAINT)*, 2006.

[15] LANForge. `http://www.candelatech.com`.

[16] R. Johnson & D. Wichern. *Applied Multivariate Statistical Analysis*, chapter 7, pages 377 – 390. Prentice-Hall Inc., fourth edition, 1998.

[17] Multiple Non linear Regression. `http://www.graphpad.com/curvefit/introduction.htm`.

[18] A. Sayed T. Kailath and B. Hassibi. *Linear Estimation*, chapter 3, pages 78 – 88. Prentice-Hall Inc., 2000.

[19] The MathWorks MATLAB and Simulink for Technical Computing. `http://www.mathworks.com/`.

[20] P. Steenkiste N. Hu. Estimating available bandwidth using packet pair probing. Technical report, CMU-CS-02-166, September 2002.

[21] R. Baraniuk J. Navratil V. Ribeiro, R. Riedi and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurement Workshop 2003*. 2003.

[22] C. Dovrolis M. Jain. Pathload: A measurement tool for end-to-end available bandwidth. In *ACM SIGCOMM*, August 2002.

[23] A. Hoisie A. Malony & B. Miller V. Getov, M. Gerndt. *Performance Analysis and Grid Computing*. Kluwer Academic Publishers, 2004.

[24] N. Bieberstein C. Gilzean J. Girard R. Strachowski S. Yu B. Jacob, L. Ferreira. *Enabling Applications for Grid Computing with Globus*. IBM, June 2003. `ibm.com/redbooks`.

[25] June 2005 The GridChem Project. `https://www.gridchem.org/`.

[26] June 2005 Lattice QCD Portal. http://lqcd.jlab.org/.

[27] June 05 US National Virtual Observatory. http://www.us-vo.org.

[28] June 05 Fusion Grid Collaboratory. http://www.fusiongrid.org.

[29] June 05 Particle Physics Data Grid. http://www.ppdg.ne.

[30] June 05 Cactus Project. http://www.cactuscode.org/.

[31] June 05 Biomedical Informatics Research Network (BIRN). http://www.nbirn.net.

[32] June 05 The nanoHUB Project. http://www.nanohub.org.

[33] June 05 Geosciences Network (GEON). http://www.geongrid.org.

[34] June 05 NASA JPL QuakeSim portal. http://complexity.ucs.indiana.edu:8282.

[35] June 05 Earth System Grid. https://www.earthsystemgrid.org/.

[36] S. Graves D. Reed K. Droegemeier R. Wilhelmson B. D. Plale, D. Gannon and M. Ramamurthy. Towards dynamically adaptive weather analysis and forecasting in lead, May 2005.

[37] Grid Computing. http://www.gridcomputing.com/.

[38] Grid Café. http://gridcafe.web.cern.ch/gridcafe/.

[39] C. Kesselmen I. Foster and S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. Intl. Journal of Supercomputing Applications, 2001.

[40] A. Geist K. Chanchio and M. Chen. *A Study of Site Security and Grid Computing Policies*.

[41] I. Foster. *What Is the Grid? A Three Point Checklist*. July 2002.

[42] J. Schopf and B. Nitzberg. *Grids: The Top Ten Questions*.

[43] J. Chen B. Hess A. Kowalski W. Watson, I. Bird and Y. Chen. *A Web Services Data Analysis Grid*.

[44] E. Deelman. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, Vol. 1, 2003.

[45] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputing Applications*, Vol. 11, 1997.