PARALLEL REMOTE INTERACTIVE MANAGEMENT MODEL

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Faris Nabeeh Zuriekat

September 2007

PARALLEL REMOTE INTERACTIVE MANAGEMENT MODEL

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Faris Nabeeh Zuriekat

September 2007

Approved by:

Ernesto Gomez, Advisor, Computer Science	Date
Dr. Arturo Concepcion	
Dr. Keith Evan Schubert	

ABSTRACT

The thesis discusses PRIMM which stands for parallel remote interactive management model. PRIMM is a framework for object oriented applications that relies on grid computing. It works as an interface between the remote applications and the parallel computing system.

The thesis shows the capabilities that could be achieved from PRIMM architecture, such as communication using UDP will be more reasonable in Runtime Execution Management. The thesis contains an analytical study for applying matrix multiplication and parallel search using PRIMM interface and MPI-Cluster. PRIMM showed satisfactory results in improving the performance of a single machine by distributing the work remotely to a parallel server.

PRIMM contributed in the development of grid methodologies by applying shortcutting techniques, in which PRIMM server can be managed to shortcut the execution of other processes.

PRIMM is designed for solving small to medium problems relatively to grid computing, in which it could be very expensive to solve these problems on personal or regular business computers.

Finally, PRIMM is an object oriented framework, programmed in java and supported by open source API. Developed to be customizable to all possible remote applications. It is language independent. It gains control to the parallel cluster through the operating system, it bridges this control to a remote client through a network.

ACKNOWLEDGEMENTS

- Everything I am, everything I have achieved and everything I will achieve to be for the glory of my Lord.
- This work would not have been possible without the support and encouragement of my advisor Professor Ernesto Gomez, under whose supervision I chose this topic and began the thesis.
- I would like to extend my deepest appreciation and gratitude to Professor Keith Evan Schubert for his help and support, especially his help in overcoming the hardest obstacles crossed my path. He motivated me to proceed and achieve my goals, I wouldn't be able to proceed without his support. A very big "thank you" to him .
- I would also want to thank Professor Arturo Concepcion for his abundant support and encouragement during my master program. He has a great contribution to my scientific knowledge. I have been also influenced from him to become a hard worker. I also want to thank him so much for everything he did for me.
- The road to my graduate degree has been long and winding, so I would also like to take the chance to express my deepest sense of gratitude to my sister Helina and my brother in law Dr.Michael. My success would not been possible without their tremendous help and support.
- I am grateful to my brother Eng. Firas Zuriekat for his great help in searching and reading about grid frameworks and supporting the idea of PRIMM.

- I would like to say a big 'Thank You' to the wonderful organization I work for iMedRIS Data Corporation. Thanks to my manager Eng. William Schroeder for being patient and giving me the chance to work on my thesis. Special thanks to Eng. Giang, Eng.Rick, Catherine, Amber, Eng. Kan and Nathan for supporting me in every step I took.

- I also thank my colleague in the master program Fadi Shihadah for helping me in Latex. I would also want to thank him for being a great friend and a great helper when I was really facing the hardest obstacles during my thesis.

- My faculty at the California State University, San Bernardino.

- Thanks to Monica Latimer for her helping me getting my thesis signed and approved.

- Thanks to our advisor Dr. Josephine Mendoza for all the efforts she made during the master program.

- Finally, I am forever indebted to my parents Dr.Nabeeh and Amal Zuriekat for their understanding, endless patience and encouragement when it was most required.

DEDICATION

To Talia, Laith, Tia, Tamer, Hannah and Landon.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Each of the past three centuries has been dominated by a single technology. The 18th century was the era of the great mechanical systems accompanying the industrial revolution. The 19th century was the age of the steam engine. During the 20th century, the key technology was information gathering, processing and distributed computing.

In 1965, Gordon Moore made the following simple observation, "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000."

According to Moore's observation, by the year 1975 , devices with as many as 65,000 components would become feasible on a single silicon chip occupying an area of only about one fourth of a square inch. Nowadays in 2007, we have seen the tremendous advances in microprocessor technology that Moore expected. For example, clock rates have increased from 40 MHz in 1988 to 3.2 GHz in 2007. Not only one single microprocessor can be integrated into the motherboard circuit, but also we have seen the dual microprocessor architecture being available in our personal computers.

The move to multicore architecture is forcing every system to be distributed. Distributed computing or Grid computing is trying to decrease the limitations of the single microprocessor technology and also supports systems with more computational resources.

In the past decade, grid technology has gained an important role in sustaining the ability to solve huge computational problems. Grid technology was limited in the beginning to scientific applications, but nowadays it appears in servers and user applications.

Because grid technology relies on network speed it was limited to very small class of problems. However, network speed has been dramatically increasing during the last ten years. This allows grid technology to play a more important role in the software development process and encourages developers to adapt to the new grid computing approaches.

Grid frameworks have evolved from scientific research that attempts to solving huge problems. During the implementation of these frameworks , developers and researchers took into consideration all the environmental factors such as networks latencies, processors speeds and available resources. But since these environmental factors are variable and always get new enhancements, further development must be taken into account. In the past couple of years, enterprise development has taken place to enhance the performance of existing servers and applications. The industry of software engineering has noticed that the great enhancement in network speed and microprocessor technology will have a positive impact on the grid systems.

1.1 Grid Frameworks

A Grid Framework is a basic conceptual and software structure used to solve relatively complex problems through the application of Grid Computing. The grid framework includes a group of libraries that will simplify the development, distribution and synchronization of grid computations.

Grid computing is an emerging model of distributed computing that employs a dynamic pool of dispersed commodity computer resources to tackle large, time-consuming tasks. The chief attraction of grid computing systems is that they promise to provide large amounts of computing power more economically than costly high-end computers, and their potential capacity is scalable beyond that of the most powerful supercomputers.

The success of a number of experimental grid computing projects spurred a growing interest within the academic community and, more recently, in the realm of commercial enterpriser grid computing [13]. This has led to the development of open standards, which dominate scientific grid applications, and a variety of proprietary architectures aimed at the commercial grid market.

Many grid systems were successfully developed for various needs and purposes [6]. Some examples are: Globus Alliance tool kit, Enabling Grid for E-SciencE, Fushion Grid,

NorduGrid, Open Science Grid, OurGrid, Sun Grid, Xgrid, UC Grid, JPPF, Scalable Cluster Environment, BOINC, GRIA, Vishwa and IceGrid. And other national grid systems such as China Grid Project, D-Grid (Germany), GARUDA(India) , Malaysia National Grid Computing, Naregi Project, Singapore National Grid Project and Thai National Grid Project.

1.1.1 Outline of Grid Systems

This section will give an overview of some main grid frameworks and applications.

The Globus Alliance organization and many others all over the world worked together on developing an open source software named Globus Toolkit [2]. Globus toolkit contributes to solving problems in large grid projects. For example Globus toolkit has been used in Earth System Grid ESG, which is concerned with solving the earth climate problem.

Globus toolkit adopted many grid computing standards, it also supports and influences grid computing communities such as IETF, W3C, OASIS and GGF. It is also worth mentioning that Globus toolkit supports heterogeneity by designing special interfaces and services which the developers can use.

Globus toolkit has defined the class of problems that could be solved and implemented efficiently using Globus toolkit. The components of Globus toolkit are the Software Developmental Kit SDK , developer API, basic grid services and tools.

Another well-known gird is GridSolve. GridSolve is an RPC-based library for executing solver code on Grid resources. It uses a client/agent/server architecture, with a very simple client model [12]. It is integrated with a wide variety of development environments (MATLAB, Octave, C, FORTRAN, etc.) and supports many types of compute resources and authentication systems. The server runs in user space, and the agent handles Grid complexity.

GridSolve evolved as an enhanced model of NetSolve [12]. GridSolve basically works as a middle interface between the grid resources and the clients. It performs allocation for the resources through multiple network services. Any resources including both hardware and software could be assigned for a request. Gridsolve [12] is interested in solving complex scientific problems and in developing a grid computing environment to apply these solutions.

Another famous grid to the industry is Xgrid. Xgrid is developed by Apple Corporation. Xgrid turns a group of Macs into a supercomputer, so they can work on problems greater than each individually could solve [4]. Xgrid may operate in screensaver mode, so whenever users arent working, the Mac can crunch away at some data set. Or if developer has a group of Macs dedicated to the task, Xgrid makes it easy to set up a cluster that works around the clock, every day of the year. Developers or users of Xgrid can not perform concurrent execution management, but they can retrieve their computational results after a certain period of time. Xgrid depends on TCP/IP implementations. It could work independently via local network or communicate with their global cluster via the Internet.

Sun Microsystems has considered Sun Compute Utility Grid as the world's first and only true compute utility [8]. Sun Gird supports solutions to different sizes of problems. In another words, Sun Gird target all different computational problems in different corporations and organizations sizes. Their clients communicate to Sun Grid through the internet, no local networks are feasible. They charge 1 USD per CPU-hour. They are unlike most grid frameworks, they intend to cover both scientific and enterprize problems. It relies on implementations built on top of the TCP/IP protocol. All computations run on top of Solaris 10 operating system. They do not support remote concurrent execution management, or the interactive access, all the calculations must run to completion. Sun Grid has Sun Grid Job Submission Application Programming Interface (API) to allow developers develop tasks and jobs. Sun Grid Compute Utility allows application developers - including end users and Independent Software Vendors (ISVs) - to publish applications in a viewable list called the Job Catalog. The Job Catalog is a resource for users to search and sample applications that are available on the Sun Grid.

The Java Parallel Processing Framework JPPF is java-based grid computing framework that has been famous lately [1]. JPPF is tool to run applications in parallel, in which it could increase the application's performance and reduce the execution time. JPPF allow developers to write the code once, deploy one and execute everywhere. JPPF is supported by well-documented easy to use API [1]. It is scalable up to an arbitrary number of processing nodes. It has built-in failover and recovery for all components of the framework (clients, servers and nodes). Runs on top of Java Virtual Machine JVM. It's network communications are based on TCP/IP implementations. JPPF has a java-based job submission language and it has been adapting Java Management Extension JMX.

1.1.2 PRIMM System

Consider the problem of developing a system that may require at some point of execution more resources. There are many possible solutions that could solve the problem, but not all of these solutions are cost efficient and feasible. For example, consider the cost for a company to build a parallel cluster, or for an individual developer to buy more hardware resources, or consider the cost of maintaining it, or consider that the system is made for research purposes and doesn't have enough funds. The idea here is that there are some developers that have relatively big systems that require more resources that are not available; also they may not have the choice to buy or invest in more resources because of funding and financial limitations.

It is well known that JDBC and ODBC are interfaces between a system and its database. RPC and RMI are other well known methods to access remote resources. LDAP also is a well known protocol to access active directories. So there are many well known interfaces and ways that a developer would use to gain access to resources. Given the abundance of available remote resources, the question becomes: what would a developer use to gain access to parallel resources remotely? How would the developer manage, control and customize the access to remote resources?

Among object oriented interface that manage parallel resources is PRIMM. It contributes in solving developer's problems by distributing computations over parallel resources that could be shared between multiple clients. PRIMM gives the developers the ability to access parallel resources remotely. For example a developer in Mexico with poor hardware resources who is running a server on his personal computer could use PRIMM to access parallel resources in California State University.

The idea of sharing parallel resources remotely with a client's applications is not new to the industry. Sun Grid for example allows client's applications gain access to parallel resources for a fee based on CPU hours used. However, PRIMM is different than these systems for many reasons, for more details about these differences continue to the following section.

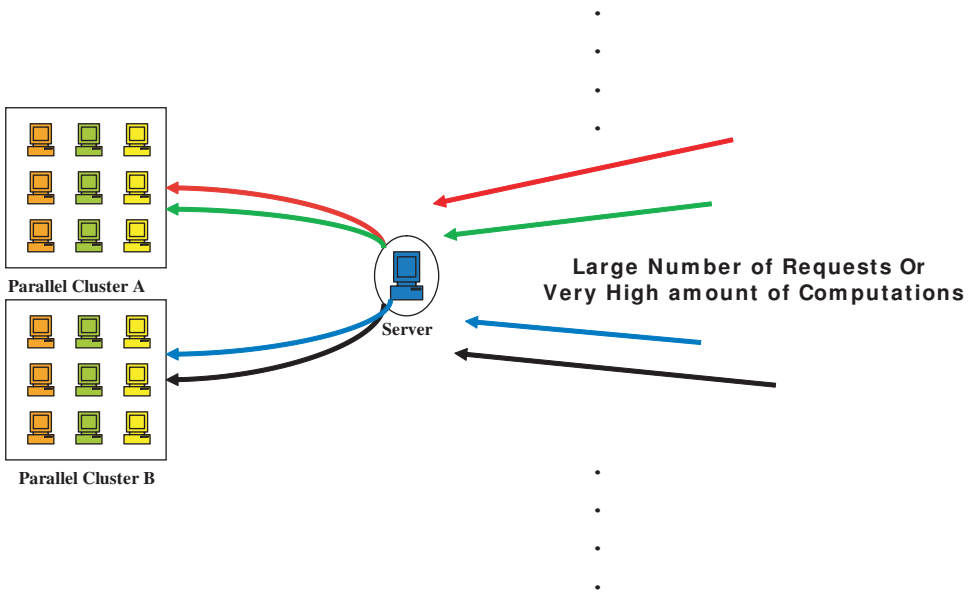

Fig. 1.1: Example of PRIMM Application

PRIMM is a relatively small system that has been developed under J2SE. It contains two main components, a server object and a client object. The two objects are used to build an interface between the parallel computational resources and the remote clients. The client is supported with an API that will help developers to build applications that may use parallel computational resources. Using PRIMM may result in reducing the overall execution time of an application or even it may increase the applications' capabilities of handling huge computational loads.

PRIMM is targeting specified type of problems. In general, PRIMM tries to solve small computational problems relatively to grid computations, in which these problems could be very costly on regular machines.

PRIMM has many functionalities. It allows developers to ship their code, compile it and run it remotely. It also allows the client application to keep track of the parallel executions, for example the client application will still be able to have an interactive relation with the parallel clusters. The interactive relation will help the client applications to know the execution status and also allows them to control it remotely.

PRIMM adopted the User Datagram Protocol UDP. UDP is a famous connectionless transport layer that has been used widely. However, in grid and distributed computing, developers mainly used TCP for communications mainly because it does not feature packet loss. TCP is a famous and reliable connection oriented protocol that could run on unreliable networks such as the World Wide Web (WWW). TCP also exhibits satisfying performance in various contexts including parallel and distributed computing. PRIMM proposes the usefulness of UDP over TCP in certain situations. UDP can perform better when information is time dependent and needs to be used for managerial purposes that are bounded by small time frames. In addition the thesis will go into an analysis of the UDP behavior in concurrent process management and compare it to the TCP behavior.

The thesis still considers the TCP protocol as a very important implementation in grid and parallel computations, applying TCP in PRIMM is considered a future work.

Both PRIMM's client and PRIMM's server are multithreaded objects. They try to perform many functionalities in non-blocking manner, for example the receiving thread is separated from the validation thread, and the sending thread is separated from the monitoring thread, and so on. These threads contributes in managing and controlling the computations in both sides.

PRIMM has a distinct feature over all other grid frameworks. It has the ability to perform shortcutting. Developers and users can manage PRIMM's threads to perfrom shortcutting as soon as the thread could find a desirable results. The shortcutting idea is described later in full details.

1.1.3 PRIMM in Comparison to Others

In comparison with Globus Toolkit. PRIMM is a centralized grid system, in that is does not distribute computations over multiple testbeds, and does not assigned tasks to unknown resources. While Globus is world-wild computing server,

and distributes it's calculations on a large scale of networks and testbeds [2]. PRIMM and Globus are completely different in consideration of the problem sizes that both of them are targeting. Globus targets the huge computational problems, that will be cheaper to solve them through the grid technology.

Java Parallel Processing Framework JPPF [1] and PRIMM shares an easy to use API. Both of them are interested in application-based problems. Java 2 Second Edition J2SE was the basic implementation language for both, but JPPF has also included services based on Java 2 Enterprise Edition J2EE. PRIMM and JPPF are different on both the problem sizes and architecture, JPPF tries to solve relatively more complex problems than PRIMM. JPPF developers manage resources through Java Management Extension JMX. PRIMM supports a runtime execution management in order to monitor and control the execution in more efficient way. JPPF is a TCP/IP based system, all communications are performed through TCP/IP or protocols on top of it. JPPF is language dependent, jobs and task are dependent on specific implementation.

Xgrid and PRIMM are two different systems. Xgrid communicates a bunch of MAC machines together to form a distributed cluster. Xgrid runs calculations that are not required to be executed instantly, it takes advantage of the idle clock cycles in the personal MAC machines. PRIMM is completely different since it tries to return the results as soon as the results are calculated. It has an active access to the execution in which users and developers can know the status of the execution and at what stage the execution is going to. In Xgrid the main interest is to solve a huge computational problems that does not require intensive management and control during execution. Xgrid has no interest in application development, unlike PRIMM that concentrates to bring resources and computational power to applications. Xgrid is a TCP/IP based system, it relies on LDAP- and Kerberos-based authentication and credentials protocols, it does not rely on any real time transport protocols.

GridSolve and PRIMM shares two main features, both of them are designed to be an interface to the grid resources. PRTIMM is similar to GridSolve client-server architecture, but GridSolve relies also on an agent that doesn't exist in PRIMM. Both of them have language and implementation independency for example GridSolve implementations could be in (MATLAB, Octave, C, FORTRAN, etc.). PRIMM is designed to solve personal and small businesses problems within a reasonable number of resources, while GridSolve is designed to solve complex scientific problems remotely. GridSolve has relied on TCP/IP transport protocol, and kept the option of UDP protocol applicable. In PRIMM, the main protocol is UDP and the future development will include the TCP/IP option.

Sun Compute Utility Grid and PRIMM are both interested to serve commercial applications. Sun Compute Utility Grid and PRIMM are both accessible systems world wide, in which PRIMM can communicate to any parallel cluster with a known IP address, and Sun Grid can be accessed from any location world wide. PRIMM can run also on a local network unlike Sun Grid that can not have clients on its local network. Sun Grid runs on Solaris 10 Operating System, while PRIMM's server is independent on the operating system. Sun Grid solves complex and simple computational problems that organizations and companies may have, it also could be adapted to solve scientific problems. Sun Grid tasks and jobs are dependent on Sun Grid implementations. Both PRIMM and Sun Grid has an API to perform the required development.

Based on the transport layer protocols [7], PRIMM can support features through the adaptation of UDP that other systems will not be able to provide efficiently. For example concurrent runtime management with very high surviving rates.

PRIMM is the first grid system that is aware of applying shortcutting methodologies. Developers can customize the monitoring threads to shortcut each other based on certain circumstances. PRIMM has the ability to open a streaming link between the operating system and the client. The clients will be able at any time to monitor their executions and make decisions during the execution of the implementations.

1.2 Parallel Languages

In the field of computer science, extensive research has already taken place to build suitable implementations for parallel computation. Many software libraries exist in different models and for different purposes.

A well-known model for parallel computations is the SPMD model, which stands for Single Program Multiple Data [3]. PRIMM depends on the parallel implementations of this model. In this model, the single program will run on all the processing elements but in different ranges of input. Only one single program is developed to run on all distributed processes in SPMD.

Message Passing Interface MPI, is a library specification for message passing, proposed as a standard by a broadly based committee of vendors, implementers, and users [14]. It was designed for high performance on both massive parallel machines and on workstation clusters. The MPI programs are function calls of the message-passing interface mixed with programming language code, most likely C or Fortran. These function calls manage the communications between the processes and performs a set of different operations. The MPI implementation is used to benchmark PRIMM's performance.

MPI showed a successful impact on the parallel and distributed computations. Research was done to build parallel computation systems on top of MPI. The PLanguages, referring to both Parallel C and Parallel Fortran have been built as language extension on top of the message-passing interface. Parallel C for example showed an elegant programming style compared to the MPI programming style.

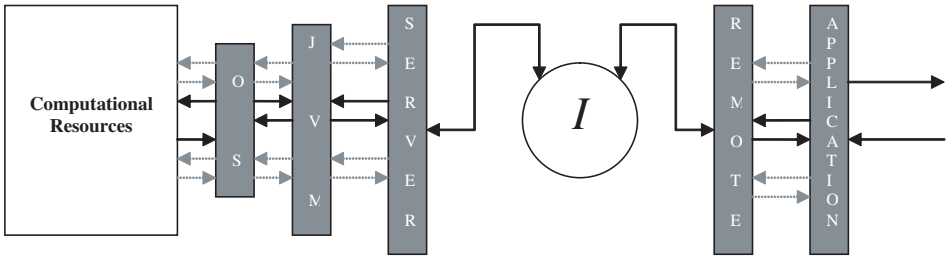

Fig. 1.2: PRIMM's Architecture General View, I:Internet or Networking Media, OS: Operating System, JVM: Java Virtual Machine, Remote: PRIMM's Client Object, Server: PRIMM's Server

1.3 PRIMM Architecture

The PRIMM idea is not completely new in reference to the current available systems in the industry. However, it still has a very distinctive specification and feature set that contributes in new grid methodologies. Recently, ideas similar to PRIMM have attracted many grid developers' interests on different web forums. PRIMM tries to bring up grid functionalities to regular developers in very simple approach.The figure 1.2 shows the general architectural view.

PRIMM is built to be an object oriented system; notice the separation between the client and the application in figure 1.2. The remote instance on the left side is a multithreaded object that contains all the necessary methods and functions to interact with the server. The server in figure 1.2 is a multithreaded class that works as an interface between the Java Virtual Machine and the remote object. The Java Virtual Machine bridges between the operating system and the server. Server and remote client are built to have a listening socket and multithreaded classes to manage the massages that are being sent back and forth between them.

PRIMM is not built from scratch, but instead it uses already existing features of other implementations. For example, PRIMM controls running process indirectly through the interaction with the operating system, performs process distribution through the message passing interface, and so on.

For the current implementation PRIMM performs its remote communication depending on the UDP protocol, making the adaptation of TCP as future work. Reasons for this choice are described in the thesis.

PRIMM has its own basic communication protocol that was built on top of the UDP protocol. Future development that should be applied to this protocol includes security, acknowledgments strategies, overlapping and others.

PRIMM is built to be a framework for grid computations. These computations could be applied using PRIMM's client object that itself could be part of the software architecture. The remote client has it's own API that exposes many classes and interfaces that could be extended or just be called at any point of time during execution of the application. PRIMM has no preference on which kind of application could be developed; for example the remote client can be used in web server development or a reporting system application development, among others.

The remote client helps the running application during its heavy load computations. The remote server operates by trying to handle the requests from the main application and transfer them to the parallel system. It does this in order to decrease the execution time on the remote machine and as a result increases the performance of the application. For example a web server that is running on a regular personal computer may have certain resource limitations. If a huge number of requests occur within a certain period, the web server may not be able to handle all the requests, or handle them in an efficient manner. The web server may ask the remote client for help in executing and processing the requests. The remote client will divert the requests to the parallel system that has much more resources than the regular machine. The server load comes from the complexity of the computations rather than from the number of requests.

PRIMM is targeting a certain class of problems that has certain size of complexity. PRIMM is not intended to solve huge problems. On the other hand, PRIMM is intended to solve problems that are considered huge for personal machines or regular businesses machines. PRIMM aims to give developers in the industry the ability to

utilize remote resources for efficiently dealing with heavy and/or complicated request handling situations.

The computational resources are the processing elements that will run the parallel algorithms. The left and right arrows show that there is input and output streaming between entities. The repetition of arrows means that multiple parallel executions may be running at the same time.

PRIMM optimizes resource usage, solves specific type of non-deterministic problems (i.e. with shortcutting), attempts to provide real-time interaction (or close to real-time interaction) between parallel processes and remote clients and perform monitoring and control. Furthermore, it could be customized in an object-oriented fashion.

In the server side, remote clients can apply their own algorithms, using any language implementation, create files and headers, compile them and run them. PRIMM is a parallel model that is language independent. Developers using PRIMM will be able to run programs in MPI, MPI 2, PC, Linda and others. PRIMM can interact with other grid systems, and resolves many issues that have been an obstacle in the development of grid systems.

PRIMM also has a shortcutting option, in which it can save more resources and decrease execution time by a significant factor for some class of problems.

PRIMM is currently used in the following situation: One parallel system and multiple remote clients. This client-server architecture is similar to the Remote Method Invocation RMI or the Remote Procedure Call RPC. (Refer to communication protocol chapter). PRIMM tries to solve the problem by using a reasonable number of resources, unlike other grid technologies that spread calculation other hundreds or sometimes thousands of processing elements. The sever address is assumed to be static meaning that the remote clients know the server IP address and port number.

Globus, Sun's Grid Engine, Java Parallel Processing Framework JPPF, Teragrid and others are great grid frameworks. There are many reasons why developers would want to use them. However, PRIMM is a framework that provides an API for developing a different family of Grid technologies, which rely heavily on the UDP protocol.

A. Language Independency

PRIMM simplifies applications development that relies on grid computations by providing the applications with an interface to different parallel implementations. Being language independent is not only useful for applying variant types of implementations but also increases the scope of grid usage. With PRIMM, more developers have a way to get involved in this type of development without being limited to a single type of parallel implementation.

PRIMM is not the only system that concerns about being language independent; GridSolve [12] for example can be applied into three different parallel implementations. But by providing a more comprehensive open-source API, developers will have more flexibility in controlling and managing these different parallel implementations.

B. Runtime Execution Management

PRIMM introduces the Runtime Execution Management feature to grid computations. This allows applications to perform control during the execution of parallel implementations within a small time frame. PRIMM relies on the analysis of transport layer protocols. It also takes into consideration that most of grid systems have been developed on top of TCP, or other protocols that are built on top of TCP, like SOAP for example.

The goal of the Runtime Execution Management in PRIMM is to show that Runtime Execution Management contributes to higher performance and an increased capability to solve problems that require many frequent interactions with the server.

C. Shortcutting

PRIMM is the first grid system that was developed with the consideration of the shortcutting technique in mind. PRIMM performs shortcutting on local and remote machines. The most important issue here is that shortcutting increases the performance and contributes to better resource management.

D. Customization

Through an object-oriented approach, PRIMM seeks to make remote development easier and more powerful.

2. COMMUNICATION PROTOCOLS

This chapter has been included to describe the important differences in protocols architecture and how these protocols can be a key factor in the development of grid technologies [23]. The Real-Time Transport Protocol RPT is described in abstract to give a close idea about the requirements of PRIMM's protocol. PRIMM functionalities is highly dependent on understanding the differences between the Transmission Control Protocol TCP and the User Datagram Protocol UDP.

The internet has two main protocols in the transport layer,a connectionless protocol and a connection-oriented one [22]. The connectionless protocol is the UDP. The connection-oriented protocol is TCP. Because UDP is basically just IP with short header added, it would be easier to start with it.

2.1 User Datagram Protocol - UDP

UDP is one of the core protocols of the Internet protocol suite, UDP is a connectionless oriented protocol that uses an IP address for the destination host and a port number to identify the destination application. UDP is very close to IP, There's almost a one-to-one correspondence between UDP and IP. For most small packet sizes, one UDP packet corresponds to one IP packet [16].

UDP is unreliable; there is no guarantee that a UDP packet will actually be received, and also does not ensure that packets will be received in order. UDP is packet-oriented. It sends a series of discrete messages. The packets that carry relatively small chunks of data are called datagram packets.

UDP transmits segments consisting of an 8-byte header followed by the payload. The two ports in the header serve in identifying the end points within the source and destination machines. When a UDP packet arrives, its payload is handed to the process attached to the destination port. This attachment occurs when BIND primitive or something similar is used; the binding process in UDP is similar to the TCP binding process. In fact, the main value of having UDP over just using the IP is the addition of the source and the destination ports. Without the port fields, the transport layer would not know what to do with the packet. With them it delivers segments correctly.

Port number is divided into three categories. First the well-known privileged port numbers that are assigned by Internet Assigned Numbers Authority IANA, this category has a range from 0 to 1023, these port numbers are reserved for well-known universal application, as a way making standard ports for especial applications. Second, register users port number, this category is based on user definition not related to any universal applications; it ranges from 1024 to 49151, other name for this category in the user port numbers. Third category is private/dynamic port numbers, ranges form 49152 to 65535, these ports are neither reserved nor maintained by IANA, they can be used for any purpose without registration so they are appropriate for private protocol used only by a particular organization [22].

Developers can design there application to bind to any port number in the range of user port numbers category. The application address on the local host if defined by the port number and the address of the machine that has that application is defined by an IP address. For example, imagine that the IP address is the address of your apartment complex, and then the port number will specify which apartment number in that complex you live.

The source port is primarily needed when a reply must be sent back to the source. By copying the source port field from the incoming segment into the destination port field of the outgoing segment, the process sending the reply can specify which process on the sending machine is to get it.

The UDP length field includes the 8-byte header and the data. The UDP checksum is optional and stored as 0 if not computed (a true computed 0 is stored as all 1s). Turning it off is foolish unless the quality of the data does not matter (e.g. digitized speech).

It is probably worth mentioning explicitly some of the things that UDP does not do. It does not do flow control, error control, or retransmission upon receipt of a bad segment. All of that is up to the user process, or the application level and the way the developer design his application header. What it does do is provide an interface to the IP protocol with the added feature of de-multiplexing multiple processes using the ports.

One area where UDP is especially useful is in client-server situations. Often the client sends a short request to the server and expects a short reply back. If either the request or reply is lost, the client can just time out and try again, not only is the code simple, but fewer messages are required (one is each direction) than with a protocol requiring an initial setup.

For example, an application that uses UDP this way is DNS (the Domain Name System). In brief, a program that needs to look up the IP address of some hostname, for example www.csci.csusb.edu, can send a UDP packet containing the host name to a DNS server. The server replies with UDP packet containing the IP address. No setup is needed in advance and no release is needed afterward [16].

UDP is considered connectionless since there is no virtual circuit set up, just a series of datagram packets is being sent. But on the other hand, it is considered point-to-point, since it is possible that one sender can have only one receiver.

Be aware that the Ethernet has a limit on the size of the UDP packet can be transferred, this size limitation is basically related to the hardware limitation. The Maximum Transmission Unit MTU is preferable not to exceed 1500 bytes per Ethernet packet. The problem gets more complex, because an IP packet being sent across the internet may cross several types of hardware. That means an IP packet may be fragmented as it travels from source to destination. So if the packets carry

more than 1500 bytes it is more likely to get fragmented, meaning that the packet will be divided into two or more smaller packets.

Other constraint is the Operation System OS, many OS's have memory limits on their input buffers that limit default incoming UDP packets to 8192 bytes. Any host must take at least 512 bytes.

Three main causes of getting the UDP packets dropped. First, it possible that the router drop the packet because of many issues the router might have. Second, the machines input buffer (at 8192 bytes) fills up; packets that arrive while the buffer is still full are dropped. Third, some transmission error, IP packet get dropped, UDP packet dropped as a result.

UDP works well in peer-to-peer designs. There is not always one central server; instead, each host communicates with whatever host it needs to. And it is also possible to build one centralized server.

UDP blocking technique, is to separate the socket processes in single thread rather than blocking the whole communication.

2.2 Remote Procedure Call - RPC

Sending a message to a remote object or host and receiving back a respond is a lot like making a function call in programming languages. In both cases you start with one or more parameters and you get back a result you need. This observation can lead people to attempt to arrange request-reply interaction on the networks to be cast in the form of procedure calls. Such an arrangement makes network applications much easier to program and more familiar to deal with. Imagine a function named getAddressByHostname(hostname) that works by sending a UDP packet to DNS server and waiting for the reply that contains the IP address, timing out and trying again if nothing return back quickly enough. In this way, all the details of networking can be hidden from the programmer.

Birrell and Nelson came up with the idea of remote procedures calls in 1984 [22]. In a nutshell, what Birrell and Nolson suggested was allowing programs to call procedures located on other remote machines. When a process on machine A calls a procedure on another machine B, the calling process in A is suspended and the execution of the called procedure takes place on machine B. Information can be transported from the caller to the called process in an invisible way from the developer [22].

RPC and UDP are good fit to each other. UDP is commonly used to build the RPC. However, when the parameters or results may exceed the size of a UDP packet, results and values are divided into multiple UDP packets and sent, but it would be more reliable if TCP connection is used in this case. Or when the requested operation is not idempotent - (i.e. cannot be repeated safely, such as incrementing a counter).

The typical point of view in RPC is that the caller performs an execution of a procedure call on another single remote machine. RPC idea could be expanded to perform a whole set of computations using a remote parallel system. Through the thesis, multiple considerations are made to preserve the properties of RPC, meaning that procedure calls from the caller point of view are the same, but from the server it is a parallel implementation.

2.3 Real-Time Transport Protocols - RTP

Client-server RPC is one area in which the UDP is widely used. This is because if RPC is to be expanded to parallel implementations, it would be necessary to go through the UDP real time implementations. As streaming media such as Internet radio, internet telephony, music-on-demand, videoconferencing, video-on-demand, and other multimedia applications become more commonplace, people will discover that these types of applications would require higher Quality of Service [22]. PRIMM shares with RTP applications the real time interaction requirement between it's clients and servers. Developing real time processing may contribute in many new ideas in the grid technologies and it's applications.

The position of RPT in the protocol stack is somewhat strange. It was decided to put RTP in the use space and make its implementation run on top of UDP [22]. It operates as follows. The multimedia application consists of multiple audio, video, text, and possible other streams. These feed into RTP library, which is in the user space along with the application. This library then multiplexes the streams and encodes them in RTP packets, which it then stuffed into a socket. At the other end of the socket, UDP packets are generated and embedded in IP packets. If the computer is on the Ethernet, the IP packets are then put in frames for transmission. The protocol stack for this situation is shown bellow. The second figure shows the similarity with the PRIMM protocol requirement.PRIMM's protocol requires to be located in the same level as the RTP, see figure 2.1.

The basic function of RTP is to multiplex several real-time data streams onto a single stream of UDP packet. The UDP stream can be sent to a single destination (unicasting) or to multiple destinations (multicasting). Because RTP just uses normal UDP packets, its packets are treated normally through the routing process, unless special quality of service is defines in the IP protocol header at the network layer, there are no special guarantee about delivery and jitter.

In the RTP stream each packet sent is given a number one higher than its predecessor. This numbering allows the destination to determine if any packets are missing. If a packet is missing, the best action for the destination to take is to approximate the missing value by interpolation. Retransmission is not a suitable option since the retransmission is going to consume time that will make the re-sent packets invaluable at that time.

The RTP payload may contain multiple samples and they may be encoded in many formats the way the application wants. To allow for interpreting, RTP defines several profiles (e.g. a single audio stream). For each profile, multiple encoding formats are allowed, for example a single audio stream may be encoded as 8-bit PCM samples at 8 kHz, delta encoding, predictive encoding, GSM encoding, MP3, and so on. RTP provides a header field in which the source can specify the encoding but is

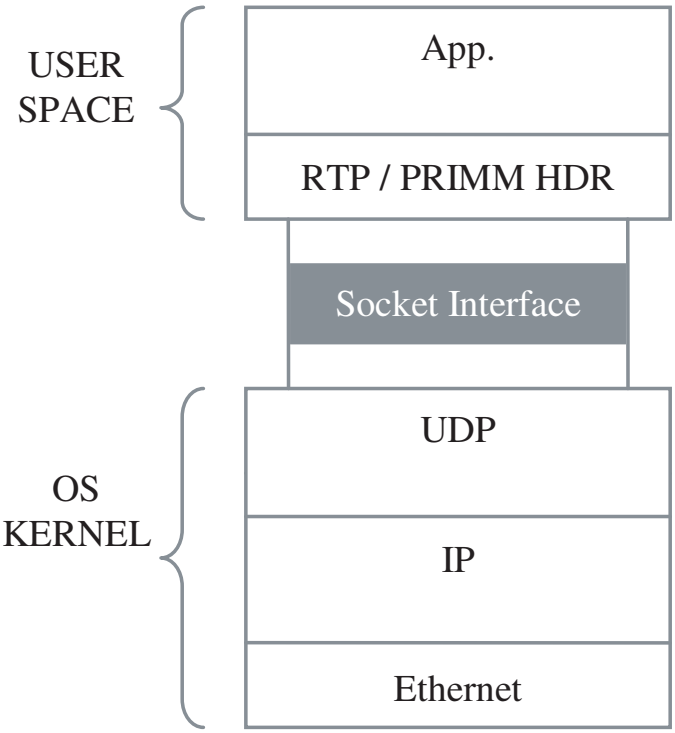

Fig. 2.1: Protocol Interface Level

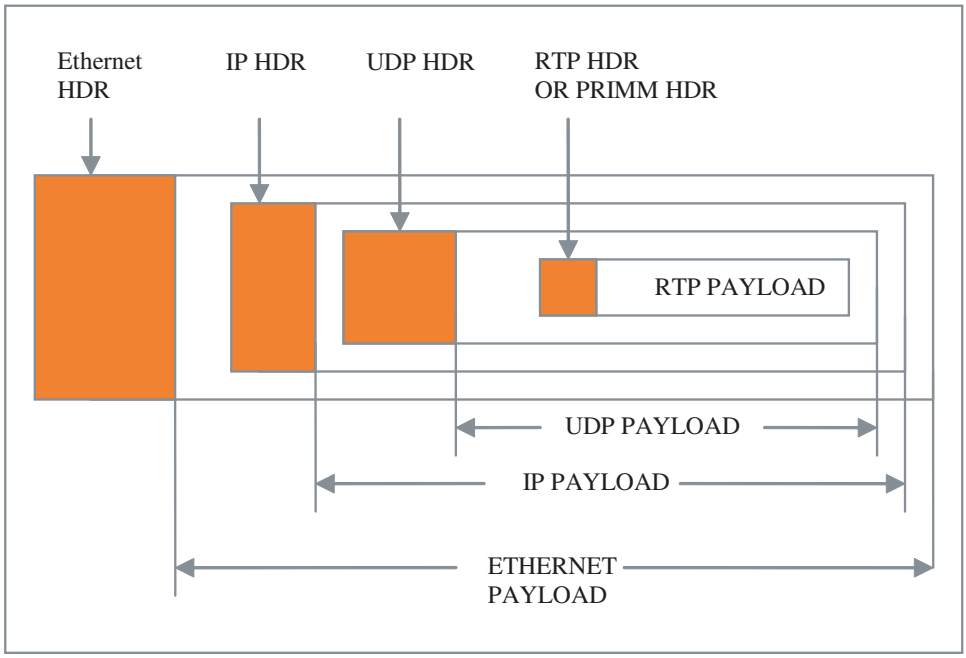

Fig. 2.2: Network Headers

otherwise not involved in how encoding is done.

PRIMM's protocol suggests the use of variant payloads' formats. The encoded formats may help PRIMM's protocol to adapt to different network congestion levels. This may come a key element if PRIMM's application depends on transferring a big chunks of data, for example image processing.

A vital and shared feature between multimedia real-time application and PRIMM client application is the time stamping. The idea here is to allow the source to associate a timestamp with the first sample in each packet [15]. The timestamps are significant; the absolute values have no meaning. This mechanism allows the destination to do a small amount of buffering and play each sample the right number of milliseconds after the start of the stream. This is done independent of when the packet containing the sample arrives. Not only does time stamping reduce the effect of jitter, but it also allows multiple streams to be synchronized with each other. For example, a digital television program might have a video stream with two audio streams.

The details of RTP and header formats could be easily found on the web or in the thesis references [22] and [15].

2.4 Transmission Control Protocol - TCP

Transmission control protocol was specifically designed to provide a reliable end-to-end byte stream over an unreliable network [15]. The internet network differs from a single network because different parts may have widely different topologies, bandwidths, delays, packet sizes, and other parameters. TCP was designed to dynamically adapt to properties of the network and to be robust in the face of many kinds of failures.

Obviously each machine that supports TCP has a transport entity: Either a library procedure, a user process, or part of the kernel. In all cases, it manages TCP streams and interfaces with the IP layer [16]. A TCP entity accepts user data streams from local process, breaks them up into pieces not exceeding 64 KB (in practice, often 1460 data bytes in order to fit a single Ethernet frame with the IP and TCP headers), and sends each piece as a separate IP datagram. When a datagram packet containing TCP data arrives at a machine, they are given to the TCP entity, which reconstructs the original byte streams.

TCP service is obtained by both the sender and receiver creating end points, called sockets, each of the sockets have a socket number and address, basically IP address and port number,.

The TCP socket may be used for multiple connections at the same time [16]. In other words, two or more connections may start or terminate at the same socket,

All TCP connections are full duplex and point-to-point. Full duplex means that traffic can go in both directions at the same time. Point-to-point means that each connection has exactly two end points. TCP does not support multicasting or broadcasting.

A TCP connection is a byte stream, not a message stream. Message boundaries are not preserved end to end. For example, if the sending process does four 512-bytes writes to TCP stream, these data may be delivered to the receiving process as four 512-byte chunks, two 1024-byte chunks or even on 2048-byte chunk.

So being stream oriented means that the data is an anonymous sequence of bytes. There is nothing to make data boundaries apparent. The receiver has no means of knowing how the data was actually transmitted. The sender can send many small data chunks and the receiver receive only one big chunk, or the sender can send a big chunk, the receiver receiving it in a number of smaller chunks. The only thing that is guaranteed is that all data sent will be received without any error and in the correct order. If any error occur, it will automatically be corrected (retransmitted as needed) or the error will be notified if it can't be corrected.

For instance, as the data is a stream of bytes, your application must be prepared to receive data as sent from the sender, fragmented in several chunks or merged in bigger chunks. For example, if the sender sent "Hello " and then "World!", it is possible to get only one trigger for data availability event and receiving "Hello World!" in one chunk, or to get two events, one for "Hello " and the other for "World!". You can even receive more smaller chunks like "Hel", "lo wo" and "rld!". What happens depends on traffic load, router algorithms, random errors and many other parameters you can't control.

The application may pass data to TCP. TCP may send it immediately or buffer it in order to collect more amount of information to fill the buffer and to be sent at once, suppose a remote application is using command line and it has been finished typing and the carriage return typed, it is essential that the line be shipped off to the remote machine immediately and not buffered until the next line comes in. This is important in order to force the socket to send data immediately. TCP has also urgent data mechanism, in which the data should be sent as soon as it is urgent and could not handle any waiting time in the buffer.

The urgent data mechanism in TCP sockets helps in developing a concurrent control between client and application and performs real-time interaction between each other. But more intensive discussion will be mentioned in the concurrent control section.

A key feature of TCP connection is that it has its own 32-bit sequence number. When the Internet began, the lines between routers were mostly 56 kbps leased lines, so a host blasting away at full speed took over 1 week to cycle through the sequences. At modern network speeds, the sequence numbers can be consumed at an alarming rate. Separate 32-bit sequence numbers are used for acknowledgments and for the window mechanism

Sending and receiving TCP entities exchange data in the form of segments. A TCP segments consists of a fixed 20-byte header, plus an optional part, followed by zero or more data bytes. The TCP software decides how big the segments should be. It can accumulate data from several writes into one segment or can split data from one write over multiple segments. The limitation of segment size, is that each segment including the TCP header, must fit in 65,515 bytes IP payload, and must fit the maximum transfer unit or MTU, which is 1500 bytes usually.

TCP is considered in the development of parallel systems, since its unique features make it more reliable to perform communications other multiple machines. TCP has not been specifically developed for parallel computations purpose, but because TCP is connection- and stream- oriented, it has become easier for developers to adapt to it.

3. RUNTIME EXECUTION MANAGEMENT

3.1 Runtime Execution Management in Grid Computing

Imagine that you turned on your chatting program -for example yahoo or msn messenger -on your personal computer to do PC to PC chatting with a friend in a different country. You start your voice conversation saying "hello", wait for your friend to reply while expecting his reply within a small time frame. As soon as you hear him interact with your "hello" word, you start the rest of your conversation, for example saying "how are you". The sentence "how are you" should arrive at a certain time range, meaning that you want your friend to hear the sentence "how are you" within the next two or three seconds, otherwise the whole conversation may be meaningless if you and he couldn't hear each other within a reasonable time frame.

For the above example it is important that streamed packets arrive within a certain time frame. Unfortunately the Internet has connectionless property [15], meaning that for some reasons packets may not arrive at the other end in the correct manner. For example it may arrive in a different order, or some of the packets may get lost. The reliability in the above example does not play the main role, because even if the packets arrived and arrived in order, it may exceed the time frame limit, so the whole data being sent and received would not have a meaningful value in such applications that are time-dependent, for example IM and Video Conferencing applications.

Another example is video and radio streaming. Users may lose some packets during the transformation of data, but at least users still get a reasonable number of packets that allow them to proceed watching or hearing. Or sometimes, the real time application may perform a packet recovery by some statistical analysis or by

using predefined functions, for example interpolation or extrapolation. So it is more important to receive the data and buffer it within certain time frames rather than receiving the whole data after a pre-determined time limit.

All existing grid computing systems are independent on the time factor during the execution. The transformation of data within a time frame is useful only for a specific and limited class of problems. Runtime execution management based on UDP transport protocol exists only in PRIMM, while other grids could perform remote management based on TCP transport protocol.

Suppose in the above two examples that TCP/IP protocol is used. Then all packets in all cases would be guaranteed to arrive in order. However, TCP/IP doesn't have any guarantee that the virtual deadline explained earlier would not be exceeded. For instance, if there is network congestion, then TCP/IP might be a bad choice [11]. Not to mention, TCP/IP may also add to the network latency and slow down the communication channel between the source and destination. It has been proved that TCP/IP is not a suitable choice for these types of applications.

The computer industry developed many real time protocols and many modified versions of TCP (example fast TCP) to replace the dominant TCP/IP protocol [11]. Many of these protocols are built on top of IP and UDP. The UDP has much more capabilities in handling real-time application problems than TCP/IP [15].

The transferred data in parallel and distributed computations are sometimes time dependent. If the data arrived or is sent too late it might lose its high level value (i.e., the value in the information it is supposed collectively portray to the human receiver). Efficient runtime execution management and optimal resource management requires real-time (or close to real-time) interaction between the client and the server, meaning that packets carrying commands and data must be available based on specific status and situation.

The PRIMM should handle the missing specifications in the UDP protocol in order to keep all parallel and distributed communications running in more reasonable manner. Using the UDP protocol in the packet streaming process will make the

runtime execution management possible and more achievable. On the other hand, the TCP/IP has poor performance in real time interaction applications, which makes the TCP a bad choice to perform a runtime execution management, or even an unrealistic choice, when network congestion is variable.

Assume a parallel computation system is designed to perform a statistical analysis on the New York stock market based on the dynamic stock prices at the current time. Basically, the parallel computations try to predict the stock market status in the next coming minutes, and helps customers to make decisions: Either buy or sell. The current stock market prices may be the input that feeds the parallel computing system. The system must make a decision and return values to the customers as soon as possible. It should read current stock market prices at an acceptable frequency and perform a suitable procedure. So in this situation it's desirable to have the parallel computations run as close to real time as possible in order to provide a more accurate next minute predictions.

The typical system manages the parallel computations based on real-time stock market factors because business decisions and calculations may be subject to be canceled or modified or changed at any time. The system must have concurrent control to allow immediate reflections to the parallel computations, if there is no runtime execution management then the parallel computations will not really reflect the intended calculations for the current stock prices. So latency in performing an interaction between the server and the client is not acceptable, since it may produce data that has no significant value. Data must be transferred within a defined time frame and calculations must be performed within a reasonable duration.

The importance of runtime execution management also introduced in controllers, controllers may be controlled by parallel computing system if the network latency is assumed to be very low.

The bellow figures show a single packet analysis assuming that control commands could be carried in one packet. Later, a more discussion on streaming packets or the command controls that requires a stream of packets, and PRIMM protocol requirements will be addressed based on the analysis of TCP and UDP, refer to [17].

The bellow figure shows that the UDP packet is sent from source to destination under normal network congestion level, the packet is supposed to arrive within a specific time frame as shown in the figure, the computation mode changes based on the control values of the packet, the computation mode A and B are just an examples for parallel computations that run on certain input and then switch to another. The computation mode A is supposed to finish execution at different point than the one that is shown in the figure, but because of the Runtime Execution Management the execution under mode A is terminated and switched to B. In the bellow figure the parallel computing system responded to the application within the time frame limit.

The arrows have a slop to show that the messages have spent an amount of time while transferring from one side to another. The figures don't reflect all the possibilities of the TCP or UDP cases. Many other cases may exist are not discussed down bellow. The basic idea is to show how the transport protocol can play a role in grid systems and the features that are available in these systems.

The time frame constraint could be a requirement in receiving the packets in both the parallel computing system and the real-time application, since the parallel computing system is performing calculations that reflect the real-time application. But for the bellow analysis, it is assumed that the time frames exist in both with synchronization differences, and that the time frame for the parallel computing system depends on the time frame of the application.

The bellow shows that the UDP packet is sent from source to destination under high network congestion level, the packet is supposed to arrive within a specific time frame as shown in the figure, but it exceeded that limit. If the packet is lost or arrived late, the same consequences will happen, in the UDP protocol, no need to resend the

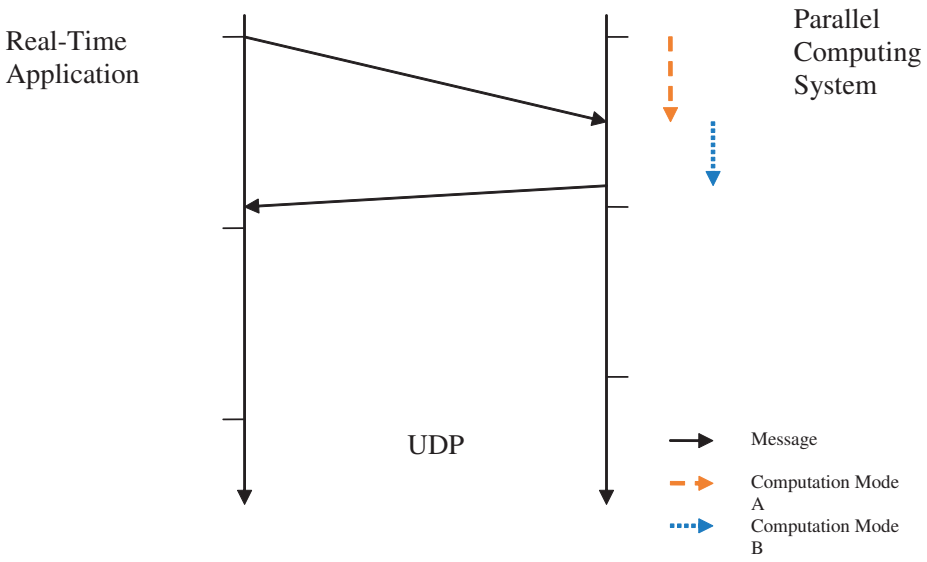

Fig. 3.1: Datagram Packet Analysis, Successful Control

packet, or most likely the resent packet will not arrive within that time constrain.

Though out the thesis the TCP messages are intended to be data segments, not packets, making reader differentiate between TCP data segments that are carried through network layer packets IP, and the UDP that are packets carried by the network layer packets IP, refer to TCP and UDP sections. So a message is meant to be a packet in UDP and data segment carried in one internet packet.

The bellow shows that the TCP segment message is sent from source to destination under normal network congestion level, the segment message is supposed to arrive within a specific time frame as shown in the figure. For each complete segment is sent an acknowledgment must be received. The TCP succeeded to perform concurrent control action in this bellow case.

The bellow shows that the TCP segment message is sent from source to destination under normal network congestion level, the segment message is supposed to arrive within a specific time frame as shown in the figure. For each complete segment is sent an acknowledgment must be received. The second segment from parallel computing system to client application got lost, the parallel computing system resend

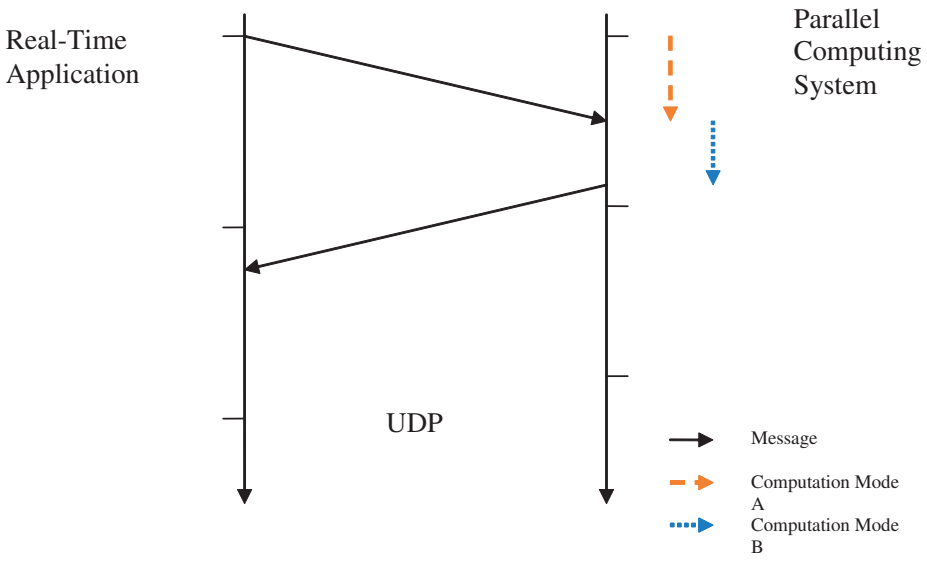

Fig. 3.2: Datagram Packet Analysis, Control Failure

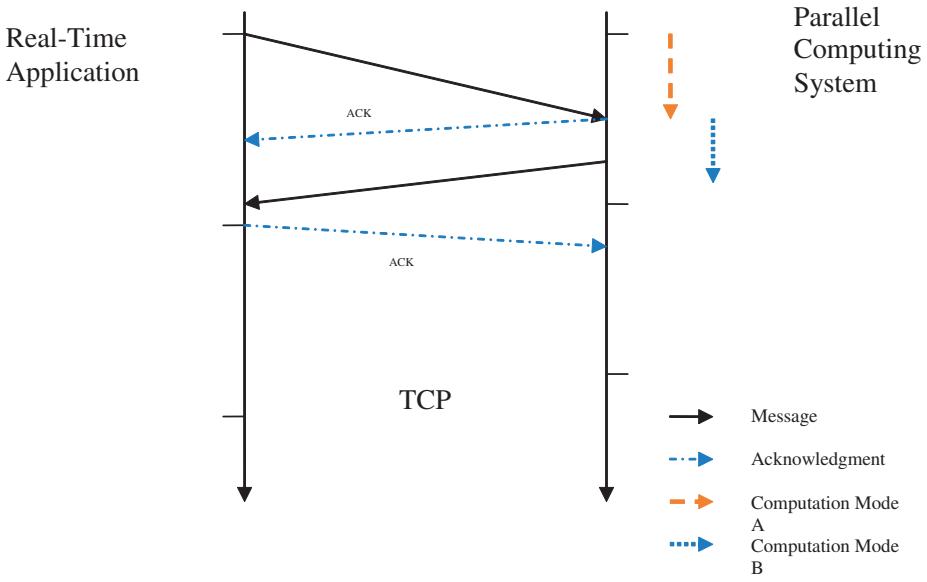

Fig. 3.3: TCP Segment Analysis,Successful Control

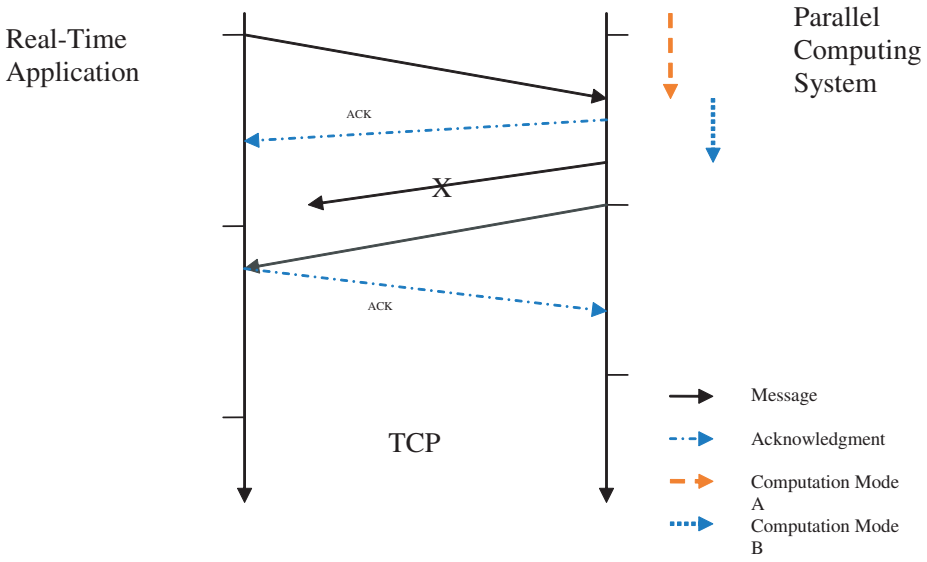

Fig. 3.4: TCP Segment Analysis, Control Failure

the segment, but the segment couldn't arrive within the time frame constrain. The last segment participates in network congestion, since both the TCP segment and its acknowledgment are useless.

Let's take in consideration the multiple packets streaming between client application and parallel computing system using both TCP and UDP.

Notice in the bellow figures, that the TCP requires more network usage, since the actual amount of packets or messages being sent and received by any side requires an acknowledgement message. So for example if four control messages being sent to the server, then another four acknowledgements should be received from the receiver, in the case of responding to each message sent to the parallel computing server, then another four responding messages will be sent to the application or to the remote client, and four acknowledgements to be received from the receiver, and so on, making the total number of messages and acknowledgements being sent an received within that time frame equal to sixteen. While if you refer to the UPD protocol only eight messages is being sent and received in that time frame.

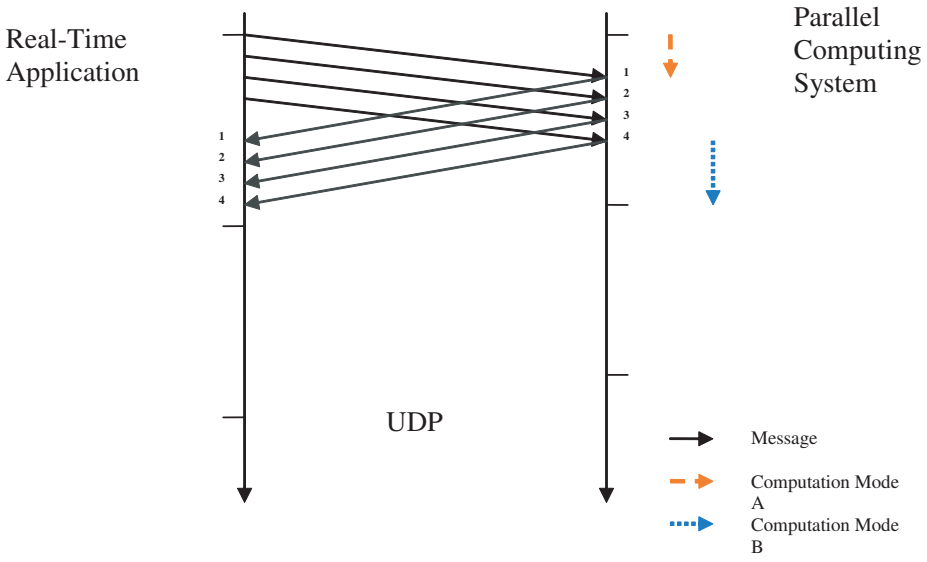

Fig. 3.5: UDP Streaming Control

The UDP messages may not arrive in order, or may not arrive at all, or may arrive exactly within the time frame and in order. So network congestion plays a vital role in what happening between the application and the parallel commutating system.

The computation mode was switched to mode B at the point where all four requesting messages were served by the previous computation mode A. Notice that when message four is sent to the real-time application the parallel computing system starts mode B, remember that this will not cancel the fact that the parallel computing system can run in many different modes and different types of calculations at the same time, but for simplicity, the figures show two modes, as an example that controlling commands can be sent and received to change the status of the computations right away.

The bellow figure shows that the UDP packets going form source to destination or to the parallel computing server succeeded in responding to the client application within the time frame limit. It is assumed that network congestion level is normal.

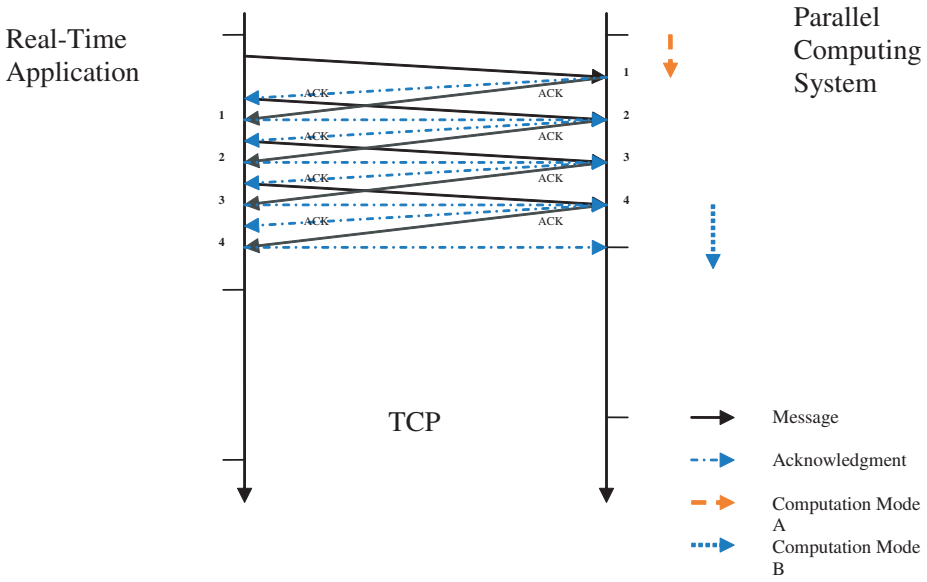

Fig. 3.6: TCP/IP Streaming Control

The real-time application that needs a concurrent interaction with the parallel computing system, requires that the application and the parallel computing system to be in sending and receiving mode at all times.

The UDP protocol, as shown in the bellow figure, used to perform concurrent control. The application starts with sending requests that require a respond within a limited time frame. The parallel computing system sends all the responds as soon as it is available, assuming that these responds are available at the time each request being received, then the parallel computing system will send a respond message to the application. The parallel computing system is not limited to a certain number of messages to be sent or received, but since the real-time interaction requires a time frame constraint, then only a limited number of messages can be in between during that time.

The figure shows that requests 1, 2, 3, and 4 are received, and responds 1, 2, 3, and 4 are sent back to the application without any obstacles. And during that the computation mode A served the requests and then switched to mode B.

The parallel computing system receives another four control requests from the application, numbered 5, 6, 7, 8. As soon as possible, the responding messages will be sent back, but for some reasons related to the network, responds 5, 8 are lost. The application only received 6, 7, the remote application may decide whether to ignore the lost responds and proceed or to perform a recovery method. Usually, this can be done by using the existing data to predict the missing ones. What determine that and what methodology should be used depend on the application type and the time frame size.

One of the last four requests that supposed to be sent from the application to the parallel computing system was lost, meaning that part of the controlling request is lost. Instead of receiving requests 9, 10, 11, 12 it received only 10, 11, and 12. Then the decision whether to proceed or to recover is also made by the receiver.

The main point is that even when lost requests and responds exist, the UDP effect on the concurrent control did not terminate its survival possibilities; it just couldn't perform an accurate controlling. And that shows that in the future when the network problems disappear the concurrent control could be still available and possible and its control accuracy returns to high, consider the option of running this methodology on private network.

Other saying, receiving anything during the time frame is better than receiving nothing or even receiving everything after. And basically depending on the remote application type and the computations being carried by the parallel computing system, the decision of whether it's required to have accurate concurrent controlling or not is based on that.

PRIMM does not eliminate the TCP option for other types of computations, but it does not support the TCP for concurrent control, because the TCP is not designed for that or in other words it is not customizable for that. A case description for that is shown bellow

In the bellow figure, the TCP protocol is applied to perform concurrent control on the parallel computing system.

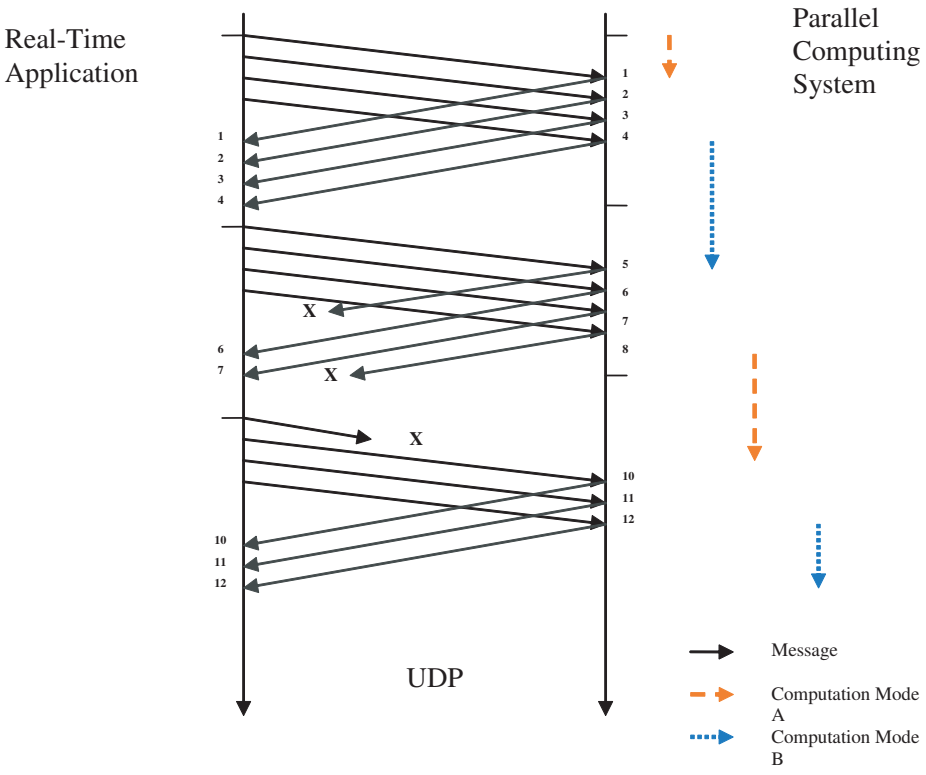

Fig. 3.7: UDP Streaming Control Failure

The first set of requests 1, 2, 3, 4 succeeded to perform an accurate concurrent control on the parallel computing system, four requests received and four acknowledgments are sent back to the remote application. The parallel computing system sends four responds as soon as these responds are available. The responds were received within the time frame constraint, and four acknowledgements were sent back to the parallel computing system.

The next sets of requests 5, 6, 7 and 8 were received by the parallel computing system, and four acknowledgements were sent back to the remote application. The parallel computing system sent responds 5, 6, 7 and 8 to the remote application, unfortunately because some issues with the network, respond number 7 was dropped, the parallel computing system waited for an acknowledgment for that respond and couldn't get it. It decided to re send it again, after the second attempt the parallel computing system got an acknowledgment; it proceeds to send respond 8. Because of the delay of resending the responds and waiting for acknowledgements, responds 7, 8 were received in a time exceeding the time frame for that request. This is considered the basic problem of TCP to be applied for concurrent control.

Be aware that the TCP protocol may send multiple internet packets at the same time and then wait for acknowledgments depending on the window management. But for simplicity, the example refers that for each request it must get an acknowledgement, or assume that the respond and request are group of packets.

The third set of requests 9, 10, 11, 12 were supposed to be sent during the time frame the remote application, but because request number 10 got lost, a resending process restarted again and another waiting time for an acknowledgement. Making request number 11 little bit late. Unfortunately, request 11 was lost also, since no acknowledgment was received. The remote application re sends request 11 and waits for an acknowledgement, the request was received by the parallel computing system but for some reason the acknowledgments got lost, the remote application decide again to resend request number 11 since no acknowledgment arrived during the waiting time. The resending process and time waiting for acknowledgment exceeded the time frame

that the four requests and responds should be sent and received. The connection-oriented feature of the TCP made the requests and responds to delay not only in one single time frame but also it affected the others.

The TCP couldn't carry on the concurrent requests and responds between the parallel computing system and the remote application, but instead it made data available at a wrong points of execution. TCP has very low survival possibilities to preserve the concurrent control.

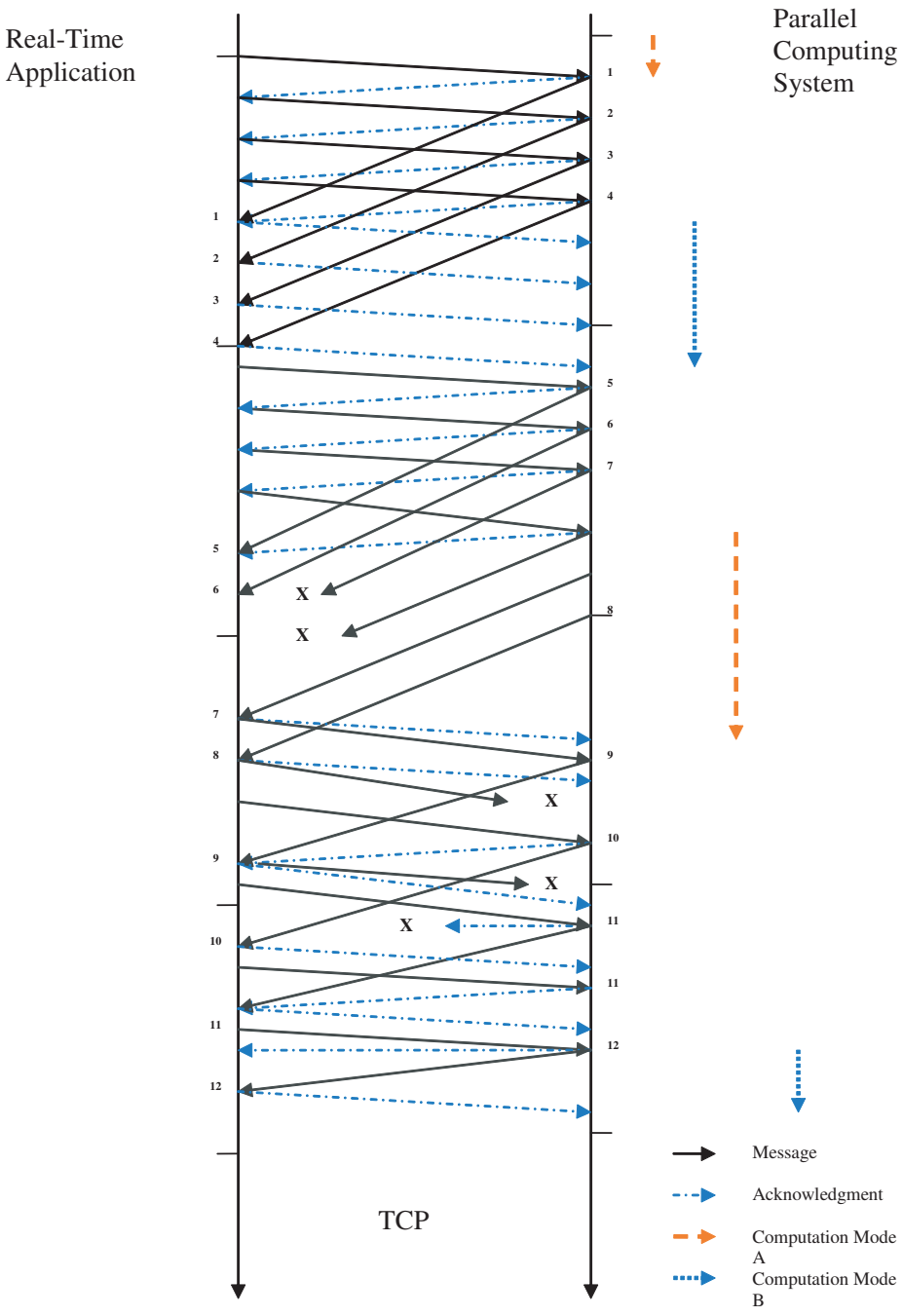

Fig. 3.8: TCP Streaming Control Failure

4. SHORTCUTTING IN PARALLEL COMPUTATIONS

4.1 Shortcutting Concept

Assume that you have just got home form work and opened your fridge looking for a bottle of milk, finding out that no more milk is left. So you decide to go shopping to the closest grocery store in your neighborhood. After a very short while of leaving your home, your roommate calls you telling you that s/he has already brought milk for the whole week. At that point, wouldn't it be wise to turn back home? Assuming the purpose of the trip was only to buy milk. Assume further that you didn't take any means of communication with you (for e.g. no cell phone or walkie-talkie), then your decision of buying milk would not change most likely. The extra bottle of milk might not be of much use. You might end up putting it in your fridge while its expiration date will be approaching soon.

Sometimes computations in parallel execution may have no sense or significance, or it may be expired and not needed [5]. Let's assume that a parallel sequential algorithm is developed to run into multiple processes, on multiple machines. Each process has a loop that will run sequentially comparing the key element with an element in a distinct input array. If the element is found then its loop will exit and the value will be showed up to the users. While the rest of other processes are still running and have no clue that the element is already found. At this point it is obvious that the computations of other processes are not useful, but even harmful, in which it has reserved resources for doing insignificant work.

If the process that found the search element can talk or yell at its neighboring workers to tell them to stop, then the parallel execution algorithm will run faster. This means no need to wait for all of them to end. Discovering the fact that interruptions

of current running computations may help not only in reducing the amount of time spent for execution, but also aims to solve the problem more efficiently with the teamwork spirit.

Your decision in the above example of buying milk is made basically depending on initial circumstances, that there is no milk left in the fridge, so the decision of hitting to the grocery store to buy milk is made based on that. And in which you couldn't predict or expect that your roommate will come over with a bottle of milk. Problems in parallel computations may start with an initial attempt to solve the problem, but at certain point it discovers that different direction must be taken. So some interruptions from other neighboring processes may be involved to assure the necessity of the work.

Implementation of shortcutting in parallel execution is tricky and deceitful [5]. Readers may say why it is so hard just to inform other processes that the search element is found already. Yes, it is possible to broadcast an interrupting message to all other processes, but sending a message requires a receiver, or simply the sending process requires a receiving process. In our current methodologies for implementing parallel execution, the process can perform one thing at a time, either looping or receiving. If the receiving action is not made at that moment then the sender will either not send the value or block or may put the value somewhere into its neighboring worker memory.

4.2 Shared Memory Structure for Shortcutting

Now introducing the shared memory structure for communication, it sounds a great idea but still has its own overhead [10]. Lets assume that a flag is put somewhere in memory to indicate an interruption [9] (puts and gets functions). At each looping iteration the processes has to check the flag before proceeding, if the flag is set true, then the processes exit otherwise proceed. If the loop has hundreds of iterations, which means hundreds of comparisons has to be made; now an extra hundred of comparisons should be made also to tell whether to proceed or to stop.

```
for i <-- 1 to M
   if(Interruption Flag)
       Compare(Array[i],Key)
```

The interruption condition above usually determined by memory check, the memory check usually takes more clock cycles than usual computations since it requires reading from processor main memory. It is obvious that m + m comparisons required completing the loop. The big-O for the above loop is O (2m) ≈ O (m), since '2' is considered a constant. Meaning that the additional m is completely an overhead added to the loop.

If it is required to consider the resource consumption, then shared memory structure for shortcutting may require more resource consumption, if the parallel execution is scattered on P processors, then m*P is the overhead of the total additional computations. Meaning that the resource did more work than before, be aware that this doesn't mean that the overall execution time is decreasing or either increasing, since the interruption could occur at any point during execution, if it happened early enough, it may decrease the overall execution time, otherwise, it most likely increase the execution overhead.

4.3 Preemptive Processes Approach

Other option is still feasible, let us assume that a superior process may exist to perform the task of monitoring and controlling the running process, and assume that that process has the ability to kill or terminate the looping process [18], then at the time the interruption is made, the looping processes will be killed or terminated by a superior process. And by this way it is possible to achieve that same goal but with more resources consumption.

Creating an extra process that handles the incoming events and interrupts is considered an exaggeration. This may achieve the shortcutting concept but the resources consumptions will be high. Lets simplify the idea, assume ten processes are running to perform a parallel search on a distinct array of integers, in best case

scenario of initiating the processes, is that at each processing element a single process will run, and an additional ten will run to listen to the interrupts coming from each other, then a total of twenty processes to perform a parallel search on an array, with the assumption that the additional ten will handle all the other requirements of coordination between them. Then this may allow one process interrupts the other and so, and getting to the point that shortcutting is possible.

Through all of the above explanations for achieving shortcutting, the thesis proposes a new architecture that allows more flexibility in design and management, and preserves the resources from insignificant work. Through building an interaction interface between the operating system and the parallel processes will help in controlling and managing the resources, and even customize threads purposes based on the execution plan, the following section will go into depth of the problem and present a thesis solution.

This section requires a complete understanding of the TCP - (Transmission Control Protocol) - and a basic knowledge about MPI - (Message Passing Interface) -communication and synchronization methodologies. The reason for that is that MPI uses the TCP protocol for communication, and MPI may inherit some issues because of TCP.

4.4 Implementing Shortcutting in Parallel Execution

Many issues could be taken in consideration when implementing any parallel algorithm. For simplicity, the MPI-Cluster architecture will be the basic execution framework.

The following assumptions are made:

1. There exists a variance between local machines clocks and the actual time, meaning that there is no fully synchronized clock between all of them.

2. There exists a variance in the starting point of each process, assuming the computational loads may not be completely balanced, making some processors interacts slower than the other.

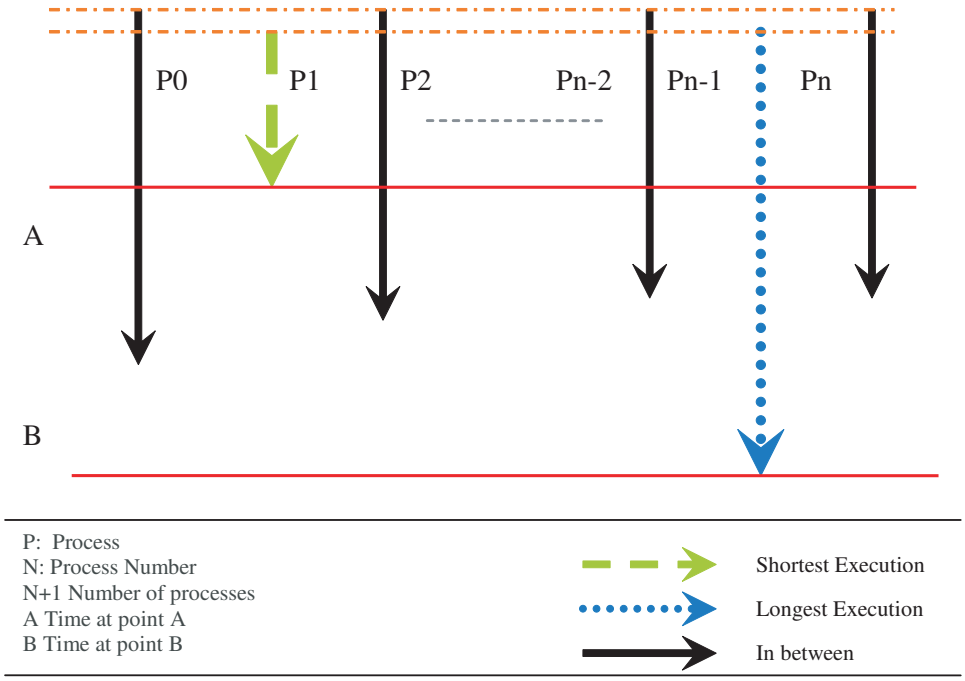

Fig. 4.1: Shortcutting Idea In Parallel Processing

3. There exists network latency.

4. There exist TCP sliding window problem. Packets may not be scheduled under optimal settings, meaning that the TCP protocol will not be adapted to the network latency during the beginning of the execution.

The figure 4.1 (N+1) processes running in an SPMD model, only one of them may find the solution; for example, the problem of sequential search, modified to run in parallel.

Process P1 was lucky to find the answer within the shortest time. Process P1 decides to broadcast the message to all other processes [10]. At point A, if shortcutting was enabled then the parallel execution will terminate somehow close to the point A. Otherwise the parallel execution will proceed to finalize state at point B. It is obvious that shortcutting may achieve less execution time by (B-A)

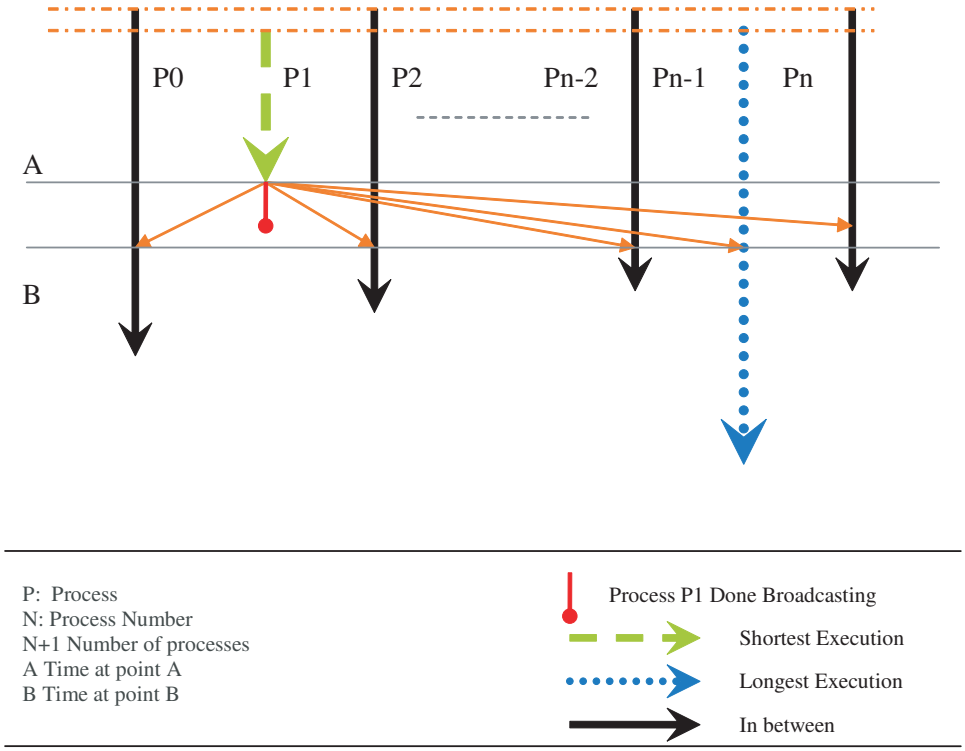

Fig. 4.2: Shortcutting Other Processes Will Result in Reducing the Total Execution Time

If shortcutting is designed based on send/receive architecture then process P1 will start a broadcasting process as multiple send/receives to all other processes, meaning that process P1 will need some extra time after point A to terminate. And due to network latency messages may arrive to other process at different times, be aware that the MPI-Clusters uses TCP as a communication protocol, meaning that in practical execution, the socket will send an interruption request and wait for acknowledgment. Packet may get lost, or may exceed the sliding window timeframe, or many other possibilities.

The bellow figure shows a safe mode shortcutting, where only process P1 could tell the others about the direction of computations, and it is only the one who performed the shortcutting, and there is no other process were interested in shortcutting.

Shortcutting will always happen after point A, since there must exist network latency. So synchronizing shortcutting interruptions is also another problem, if it is assumed that more that one process can shortcut others, because the network latency may lead to misunderstanding between them , in which who will perform the shortcutting first, and who has the most significant value. See figure 4.3.

Process P1 finds a significant indicator on the direction of the computations; it will perform broadcasting to all other processes, for example consider the prime numbers algorithms. But for some reasons process Pn tends to be also interested in shortcutting other processes, then Pn also start broadcasting to all other processes, while an interruption message from P1 to Pn was still in progress, and Pn has no clue that P1 is already in shortcutting mode.

The figure 4.3 shows that shortcutting may lead to network congestion, and an inconsistent state of execution, meaning that who is right, or who has the final decision or answer. This also may lead to blocking state between P1 and Pn, meaning that each of P1 and Pn is waiting for response.

The implementation of shortcutting in PRIMM has different approach, the computations is monitored by special monitoring thread, that can handle the issues of synchronization and consistency of the shortcutting process, the figure 4.4 shows a conceptual idea of how the monitoring thread works, the figure does not show all the details of the implementation, such as the operating system or the structure of the file system, but basically, it does perform shortcutting based on I/O streaming without the regular buffering process, meaning that an immediate shortcutting can occur and only from one process, eliminating the possibilities that are in figure 4.3.

The model carries its management through the operating systems, meaning that the process that run on different processing elements, is already running on top of the operating system, and the operating system is already performing the required management for that process, for example giving the process its time slices, or hardware resources and so. The operating system usually has higher privileges than the regular processes, meaning that operating systems can terminate processes

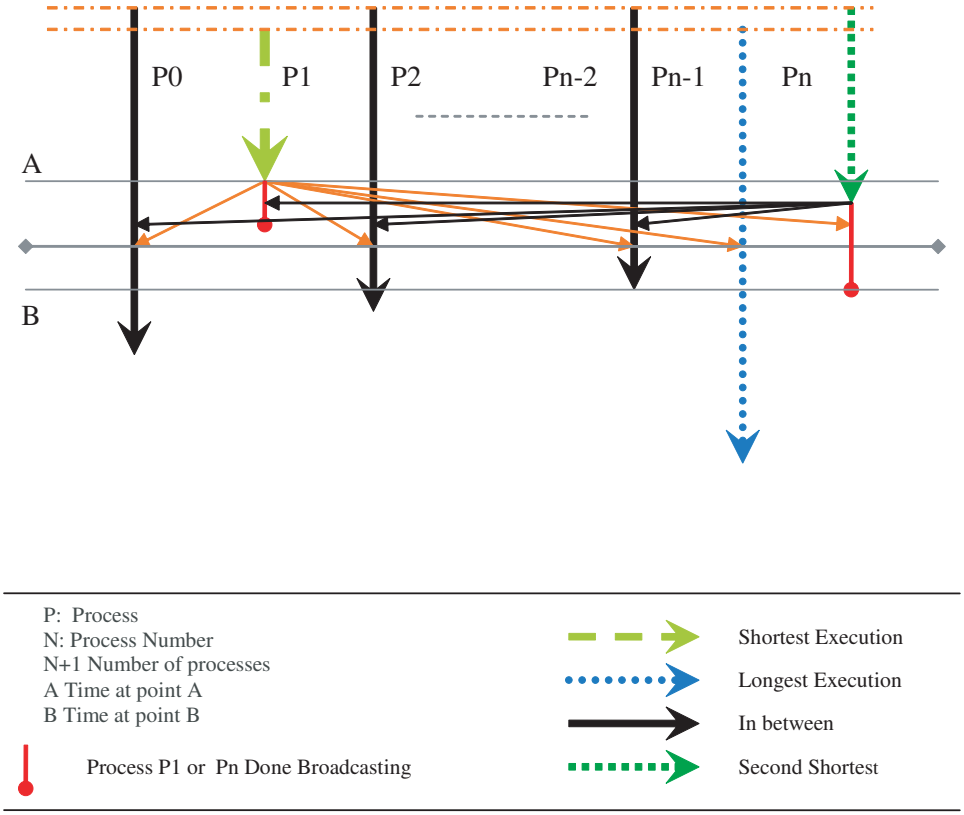

Fig. 4.3: Multiple Processes Shortcutting Condition

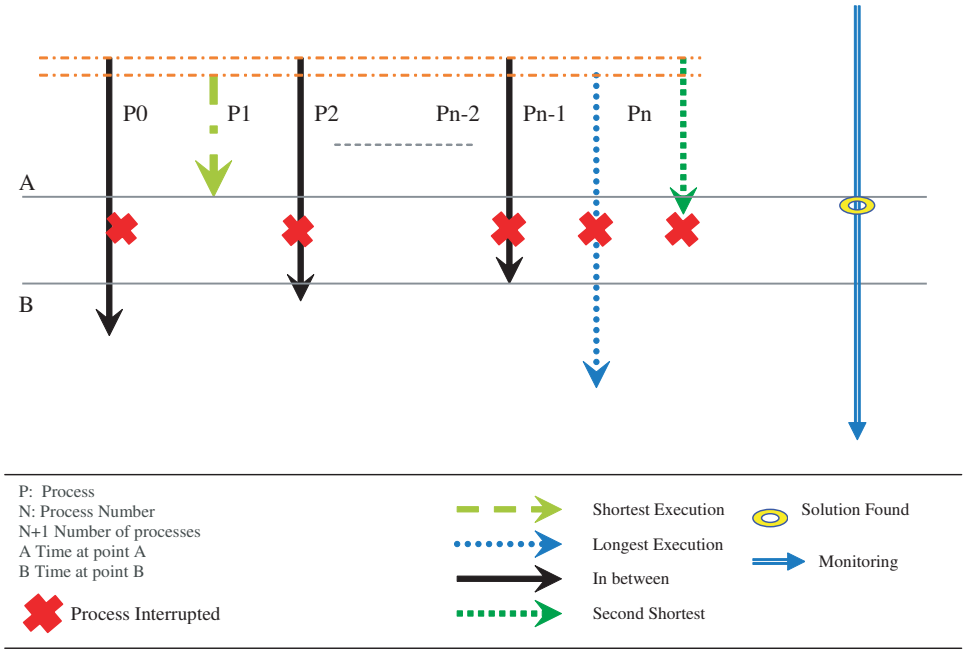

Fig. 4.4: PRIMM's Server Thread Shortcutting, Thread Monitoring the Parallel Execution

and interrupts its current execution. The PRIMM built a communication interface between the operating systems and the thread, meaning that the operating system stands as a bridge between the monitoring thread and the current running process, monitoring thread as mentioned before, has streaming capability, meaning that the thread can check what is going inside the process at any point of time, concurrent control. More specifically the thread reads a stream of bits, buffer them to one byte, and then perform the required byte conversions and concatenations operations. The thread itself is managed by another severing thread, which has also the responsibility to bridge between the thread to the outside world, so the main thread or the serving thread communicates to the monitoring thread to remote applications, in which remote application users or even remote applications can has direct access on what is going inside the parallel server.

The thread performs its operations through systems calls to the operating system, therefore the thread will perform, for example a process termination, by system call to the operating system.

Thread management is allowed, and management schemas are also available. So that the developers can write their own case scenario management plans, like determining when to proceed and when to shortcut, or what should be performed if one or more of the processing elements are down, and so on. Since during execution a number of non-deterministic conditions may occur, so the thread should be built to perform the correct action for that specific execution. The thread could not predict what kind of actions the developer needs, meaning that a developer may run matrix multiplication algorithm and other developer may run parallel sieve algorithm and so on, and in both two different monitoring thread must exit.

The PRIMM has shortcutting capabilities, the figure 4.4 shows that process P1 has the shortest execution time, and process Pn second shortest execution time, the thread has detected that P1 has already found shortcutting answer, then the thread will perform a termination operation to all other running processes. The time required to inform the thread that P1 has already found a shortcutting answer, is the streaming time between P1 and the thread. Assuming that the latency of this process is too low, practically it is, but theoretically it is required to mention it.

Only one process will return at a time to the operating system, meaning that one process can only stream to operating system at certain point of time, be aware that this doesn't mean that other processes can't, it just means that the operating system streams the processes output as soon as it has an output. So whenever a process output is available then it is streamed to the operating system. Well known process of input and output streams or I/O operations. For example, if you typed something from your keyboard, it will show up as the order you typed the letters. If you practically tried to type two letters at the same moment, then you will not see two letters overlap each other on the same display block. This gives readers hint about how the monitoring thread reads from the operating system, it reads

from shortcutting process directly, which eliminates the inconsistency state that was introduced before. Pn can not confuse the monitoring thread by anyway, since the monitoring thread will perform the suitable system call as soon as the streamed bits built a certain value or certain indication.

4.5 Analyzing Shortcutting Probability in Parallel Search

Roughly depending on the problem type, shortcutting may be considered a great option in decreasing execution time and managing resource consumption. Consider the problem of a sequential search again, if the search key exits within the distinct input array then for sure at least one of the processes will find the key before termination.

If the algorithm is designed to run on P even number of processing elements, assuming it has an input array A of size S divided into L sub arrays of size K each, if S is odd then last L will have K+1, with no overlapping, then the search key must be in one of the Ls sub arrays.

The probability of finding the key in a random position in A is

$$Pa = 1/S \tag{4.1}$$

The probability of finding the key in a random positon in L is

$$Pl = 1/K \tag{4.2}$$

Pl represents also the possibility of shortcutting in the above search algorithm. For example, assume S= 100, P=4 Then K=100/4 =25 so Pa = 1/100 while the Pl = 1/25

Pl is relatively good value to indicate that shortcutting possibility is significant rather than the Pa that shows shortcutting possibility is insignificant. Meaning

that in parallel algorithms the shortcutting possibility may have more significance than sequential algorithms. Also comparing parallel algorithm with shortcutting and without shortcutting, consider the bellow example:

Assume Ws is the work that has been preserved from insignificant computations, if the search key in the above example was found in first position then

$$\begin{aligned} W_1 &= (K-1) * P \\ W_2 &= (K-2) * P \\ &\vdots \\ W_k &= (K-k) * P \end{aligned} \tag{4.3}$$

Meaning that Ws = (1/K)*P is the work that shortcutting preserved from insignificant computations, and Ws also is the amount of computational load that was avoided,

If shortcutting is disabled then total resource work W is

$$W = (K) * P \tag{4.4}$$

$$Ws = 0 \tag{4.5}$$

This proves that shortcutting has higher potential to save ((1/K) * P - 0) work than the one without shortcutting.

5. CUSTOMIZATION

5.1 Customizing Parallel Implementation

PRIMM is language independent. It has the ability to perform parallel computations on many types of parallel implementations.

The computations can be customized remotely, which means that the parallel implementation may be coded on a remote machine and then loaded and compiled on the server machine. In other words, customization include transferring the code file from PRIMM client to PRIMM server, the server installs the file under a unique client directory in order avoid overwriting other client's work. PRIMM client then sends a control command to PRIMM server to ask for compilation. The compilation process will be managed by a lightweight thread on the server and the results of the compilation(s) would be shipped out to the client.

Many grid systems have something called task submission and task definition. The task definition is when the task is being defined by the developers on their local machine. The task can be defined in multiple ways. For example in JPPF, tasks are threads [1]; in GridRPC [21], tasks are functions; and in GLOBUS, tasks are JSDL jobs defined by Job Submission Description Language [2]. In PRIMM, tasks have no special definition language; any supported language should work. For example if PRIMM server has MPI-1 and MPI-2 installed then the developer could use any of them. The task submission process is when the implementation of the task is being received by the managing node. In PRIMM, the task is submitted when PRIMM client sends the appropriate control command.

Below is an example for the use of PRIMM client:

```
client.create("local filename path/search.c");
client.compile("search.c");
client.execute("search.c", "Execution Details");
```

PRIMM has re-usability feature. This means that developers do not have to define an already existing task; they just have to send control commands that initiate the implementation. It is unlike other systems; where developers, at each execution step, have to ship the whole task to the managing node.

Below is an example of re-usability. Notice that there is no compilation or creation processes needed.

```
client.execute("search.c", "Execution Details");
```

5.2 Customizing Buffers

PRIMM has capability to customize buffers and packets. This is a unique feature that adds more value to PRIMM. Buffers and packet sizes can be defined appropriately based on the implementation needs. PRIMM client can send control command to the server to specify what size of the buffer should be used to receive and send massages for a specific implementation. For example, the generation of an encryption key would most likely be transferred in a 256-byte packet size and the generation of a random array may need to be transferred in 1500-byte packet size. Packets and buffers size play an important role when data has to be sent within a pre-determined time frame [16]. For example, consider the situation where PRIMM client requests to generate an encryption key of size 256 bytes from the PRIMM server. In this case, the PRIMM server would accept the command and launch a monitoring thread for the parallel implementation. The monitoring thread would then start reading each byte until it reads 256 bytes. At this point, it would send the packet right away without the need to wait for the maximum capacity of the packet to be consumed.

5.3 Setting Requirements

The object oriented approach has been a decorative feature that many grid systems have. Unfortunately most O.O grid systems claim that they can contribute in application development level. That's true in theoretic view, but because of the sophisticated requirements that these grids need made the developers to avoid the use of grid computing. Grid systems usually have special installation and development requirements in order to be used in the application development, such as tutorials and classes should be taken in order for developers to start using them. PRIMM also has special requirements, but these requirements are very simple and affordable. For example to use PRIMM Client, you need to be registered user in PRIMM Server, know the IP address and port number of the PRIMM Server and finally to have PRIMM client object in you path.

6. ANALYSIS AND CONCLUSION

6.1 Introduction

In this chapter we will explore two experiments that highlight the differences between shortcutting and non-shortcutting techniques in grid and parallel computing. The chapter includes the explanation of two main experiments done to compare the execution times of different programs under the two techniques. The first experiment was based on a matrix multiplication problem and the other experiment was based on a parallel search for a key in a data set. The goal of the matrix multiplication experiment is to highlight the possibility to gain better performance by distributing problems to a parallel cluster. PRIMM shows that there are many types of problems that can be solved more efficiently using remote access to parallel resources. A typical problem that could benefit from added parallel resources is the matrix multiplication problem, thats why it was chosen for the first experiment. PRIMM with shortcutting could be used in problems where stopping the execution after finding the first good answer is useful; a problem of this nature is a parallel search for a key and that why it was chosen for the second experiment.

6.2 Matrix Multiplication

6.2.1 Problem Description

Consider the problem of multiplying two matrices $AB = C$. The calculations of multiplying the rows in matrix A to the columns in matrix B could be distributed over multiple parallel processes. Each process may contribute in finding the final result. Parallelizing the calculations may result in better performance and less time

consuming, this will be more efficient when the problem size or the number of computations involved is relatively huge to a single processors.

The distribution of matrix multiplication computations in this section is performed as rows-to-columns. It is based on the following equations that show a standard and traditional well-known formula for multiplying two matrices, for example the first row in matrix A and the first column in matrix B will be multiplied and summed to find the first element in the first row of the resulted matrix C. See figure 6.1

$$A = \begin{bmatrix} a11 & a12 & \dots \\ a21 & a22 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} A1 \\ A2 \\ \vdots \end{bmatrix} \tag{6.1}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots \\ b_{21} & b_{22} & \cdots \\ \cdots & \cdots & \ddots \end{bmatrix} = \begin{bmatrix} B_1 & B_2 & \cdots \end{bmatrix} \tag{6.2}$$

$$AB = \begin{bmatrix} A_1B_1 & A_1B_2 & \cdots \\ A_2B_1 & A_2B_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{6.3}$$

For example consider $A_{n,m}$ and $B_{m,k}$ Then

$$A_1B_1 = a_{11} * b_{11} + a_{12} * b_{21} + \cdots + a_{1m} * b_{m1}$$

Also the parallel implementation of matrix multiplication has been developed to have variable sizes, in which it could result in performing different amount of work in each process. For example, assume the distribution of matrix multiplication computations over five parallel processes. If the first matrix is A[4,4] and the second matrix is B[4,2], then the first row in matrix A and the first and second column in

matrix B will be assigned to process one, the second row in A and the first and second column in B will be assigned to process two, the third row of A and the first and second of B will be assigned to process three, and the fourth process will be assigned rows and columns in the same manner. But unfortunately in this example process five has not been assigned anything. All four processes have been assigned same and equal size of arrays (rows and columns) while process five has no work at all.

Each parallel process will be contributing in finding a row in the resulted matrix C, unless if there was no more tasks to be assigned to it as shown in the previous example, or if there are too many tasks assigned to it in which it may contribute in finding more than one row.

The master process is considered process number one, it receives all results from all others. The number of processes has been carefully picked to be five, because it has showed a satisfying execution time than other numbers. If the number of processes is much more higher than five, then the overall execution may be effected from the network latency and the MPI sending and receiving delays, and if the number is too small then the parallel computations are very limited. It is only the case at the time the experiments are taken and also it is highly dependent on raven cluster and it's situation at that time.

The extra number of computations may not be distributed equally. It is also straight forward that matrix B or the second matrix is shared between all of them. The rows are distributed according to the following equations:

$$\begin{aligned} Average_{Distribution} &= (NumberOfRows)/(NumberOfProcesses) \\ Extra_{Distribution} &= (NumberOfRows)MOD(NumberOfProcesses) \end{aligned} \tag{6.4}$$

For example, if the number of rows in A is 500 then each of the five processes will be assigned 100 rows, now assume the number of rows in A is 502, then each of the five processes will be assigned 100 and the extra two will be also assigned to process number one, so process number one will have 102 rows, and the rest will have

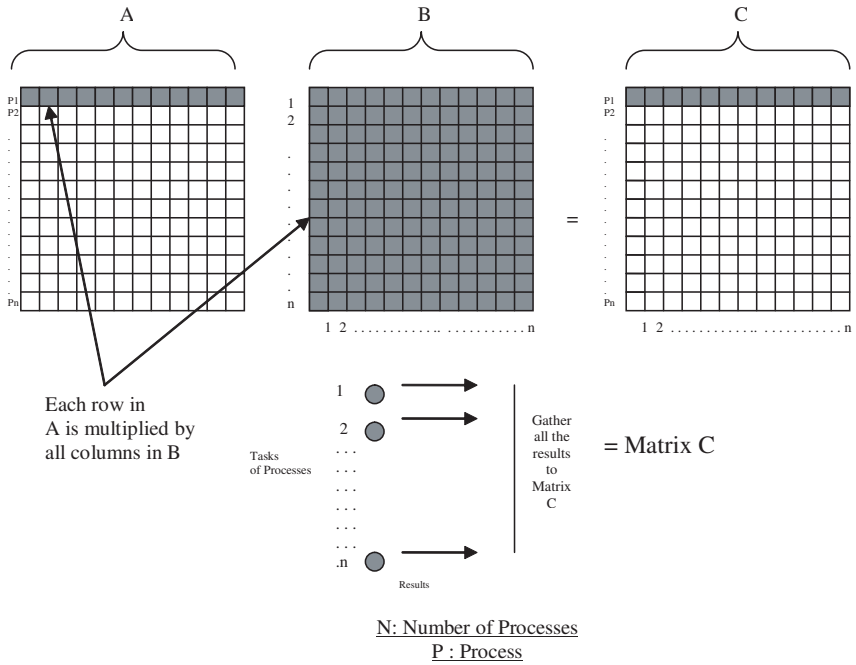

Fig. 6.1: Parallel Matrix Multiplication Distribution

100 rows each.

Matrix multiplication is considered to be a run to finish problem, in both parallel and sequential implementations , which means that the result is always available at the end. The shortcutting methodologies are not applicable in this example. There is no shortcutting overhead, since the shortcutting technique is not used at all.

Matrix multiplication is applied on three types of implementations. All of these three implementations have been tested for the same problem sizes and on the same machines.

The first one is the sequential implementation in which no network or synchronization delays exist. The second one is the parallel implementation in which the MPI-Cluster is performing the computations locally (i.e., performing the communications without any communication with the outside world). The third one is the PRIMM implementation, which includes all other delays.

The parallel computing system used here is named Raven; it is a cluster of 13 Compaq Proliant DL-360 G2 machines. Each node has two Pentium III processors running at 1.4 GHz, and 512MB of SDRAM. Nodes are connected via switched 1000GB Ethernet. The Internet is the communication media between the PRIMM client and server. PRIMM client has 1.7GHz processor, it communicates to the internet via Wi-Fi and it runs a Microsoft Windows XP operating system. The local network latency in both directions between the client and the server is around 25 ms. MPI-Cluster is used on the parallel computing system (Raven) to perform the computations. The time in PRIMM is the time between the first UDP packet sent to the server to the last UDP packet carrying the result received. The time of processing the packets is included in both the server and the client.

6.2.2 Results and Discussion

The matrix multiplication problem is applied to prove that PRIMM has a potential of decreasing the execution time by distributing the problem to a centralized parallel computing system.

An order of magnitude increase in this number due to traffic is not unreasonable, which accounts for the approximated 100,000 ms of delay of PRIMM from the parallel execution, which was run locally.

The resulted matrix C increases with the increase in the number of rows in A and the number of columns in B. Specifically, this increase results in an increase in the number of UDP packets transferred and the overall communication overhead. Benchmarks that help in decision making by providing sample data files with predetermined performance thresholds could be provided as a reference point to assist users in determining whether the PRIMM approach is a feasible option for their problem size. For example, sample data files could include matrices of varying sizes that are to be run in user's environments in order to obtain certain performance results that are dependent on user's environment. These results could then be compared to benchmarks to decide if PRIMM is a suitable approach.

Problem Size	Rows in A	Columns in A	Columns in B	Time in Seq. (ms)	Time in Parallel (ms)	Time in PRIMM (ms)	Packets
1	50	50	50	1000	1000	1200	20
2	100	50	50	2000	1000	1700	39
3	100	100	50	1000	2000	2040	40
4	100	100	100	3000	3000	2670	81
5	150	100	100	7000	4000	4510	122
6	150	150	100	8000	2000	5140	124
7	150	150	150	13000	5000	6510	187
8	200	150	150	17000	9000	9430	251
9	200	200	150	18000	9000	1020	255
10	200	200	200	25000	13000	13800	340
11	250	200	200	32000	18000	19000	427
12	250	250	200	37000	16000	19400	432
13	250	250	250	50000	18000	25400	542
14	300	250	250	56000	27000	36500	653
15	300	300	250	67000	19000	36200	659
16	300	300	300	78000	36000	44400	792
17	350	300	300	91000	29000	58400	926
18	350	350	300	94000	30000	58800	932
19	350	350	350	128000	34000	89000	1090
20	400	350	350	134000	35600	109600	1249
21	400	400	350	155000	39000	110600	1259
22	400	400	400	153000	41000	128570	1442
23	450	400	400	171000	47000	155900	1626
24	450	450	400	185000	71000	157000	1636
25	450	450	450	199000	65000	182000	1844
26	500	450	450	250000	69000	216600	2052
27	500	500	450	270000	76000	217900	2059
28	500	500	500	323000	89000	249300	2290
29	550	500	500	356000	98000	293900	2522
30	550	550	500	394000	118000	296000	2533

Tab. 6.1: Matrix Multiplication Results for the Different Problem Sizes

Notice that the matrix multiplication is not an ideal problem to show PRIMM's performance; many other implementations that have relatively smaller output may show better results. However, implementing the matrix multiplication problem is very familiar to developers and well-known parallel computing problem.

NRA is the number of rows in matrix A. NCA is the number of columns in matrix A. NCB is the number of columns in matrix B. The resulting matrix should have the same number of rows in A and the same number of columns in B. Many input combinations could result in the same output size; for example, consider A100,2 and B2,50. The product would result in C100,50, but the amount of computations spent to find C100,50 is not the same as in A100,1000 and B1000,50. Therefore, the problem's size is defined by {NRA, NCA and NCB}.

Both the parallel and PRIMM implementations used five MPI processes, distributed across five machines in the raven cluster. Time in sequential is the time in the sequential implementation with no communication. Time in Parallel is the time in parallel implementation on the MPI-Cluster. Time in PRIMM, is the time in the parallel implementation on the MPI-Cluster plus the PRIMM's client and server overhead.

The number of UDP packets transferred is an indication of the size of the data transferred. Some of these UDP packets are consumed for protocol administration. Since PRIMM server and client runs a communication protocol on top of UDP that will handle the communication issues, for example PRIMM client has to send a protocol packet to PRIMM's server to let it knows that the client process still running. Each UDP packet is 1500 bytes in size. For example, in case 1, the final result costs around 30000 bytes of transferred data.

The table 6.1 shows the problem size for each case, the problem size column is used in the graph 6.2 as X-axis.

Notice that the processing in case number 29 has a slightly less overhead than that in case number 30. This is because the formal has 50 columns more, which translates into more computations. Also the overhead of the communication between number 29 and 30 is almost the same. The small variance between the number of packets is a result of PRIMM's protocol overhead.

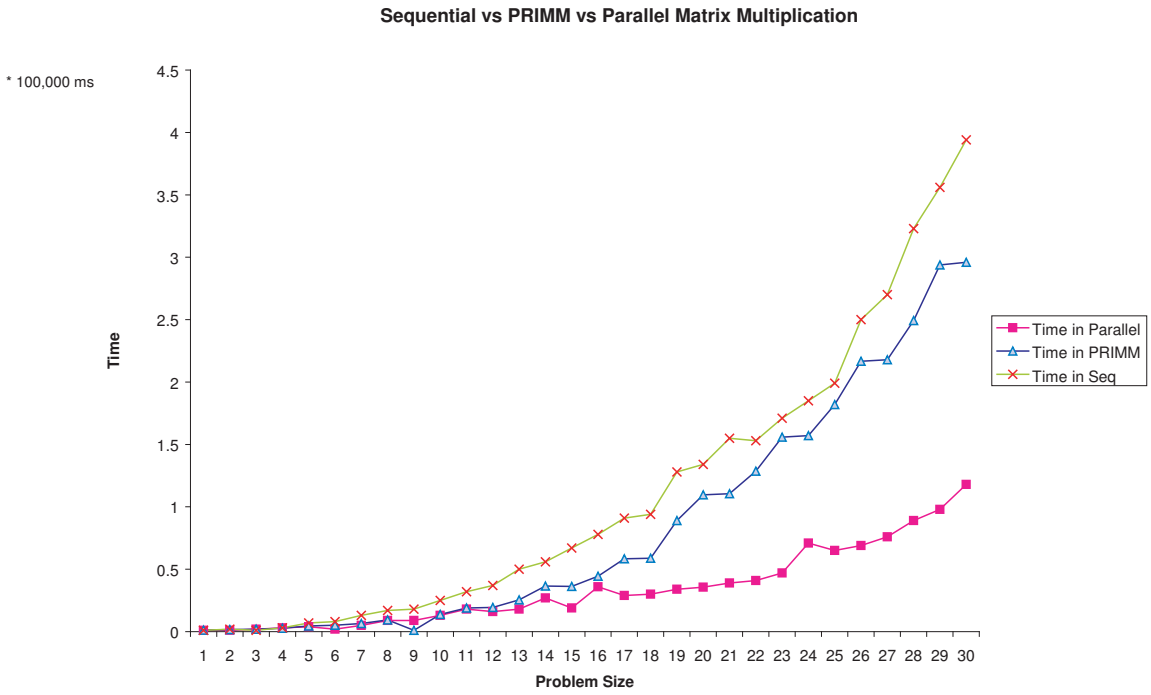

Fig. 6.2: The Three Matrix Multiplication Implementations

Graph 6.2 shows all three implementations. Notice that PRIMM has less execution time than the sequential implementation and higher execution time than the parallel implementation. The overhead of PRIMM is considered high depending on the graph 6.2 and also it is considered better than the sequential implementation.

The local implementation of matrix multiplication on a parallel cluster is considered the baseline for performance, because PRIMM could have equal or lower performance given the added overhead at the same problem size. Since the matrix multiplication problem is considered run to finish problem that is not suitable for shotcutting. This fact makes PRIMM non-shortcutting implementation typically have lower performance in comparison to the local parallel implementation.

The approximate total time spent in network latency between PRIMM server and PRIMM client is shown bellow, there is a possibility that the UDP packets overlap each other. The values are approximate numbers based on the time of the experiments.

Given that $time_{transmit} = 30ms$, $time_{transmit} = .04ms$, the number of packets to transmit is

$$packets \quad = \quad matrices \times \frac{rows \times columns \times bytes_{element}}{bytes_{packet}}, \tag{6.5}$$

and that an update is sent when an element is calculated, thus

$$updates \quad = \quad rows \times columns, \tag{6.6}$$

roughly the time consumed by the network from the cluster's perspective, with no collisions, is

$$
\begin{aligned}
ms_{net} &= ms_{start} + ms_{status} + ms_{answer} \qquad (6.7)\\
&= (ms_{transmit} + packets \times ms_{send}) \\
&\qquad + (updates \times ms_{send}) \\
&\qquad + (ms_{transmit} + packets \times ms_{send}) \\
&= \left(35ms + 2\frac{500^2 \times 8}{1500} * .04ms\right) \\
&\qquad + \left(500^2 \times .04ms\right) \\
&\qquad + \left(35ms + \frac{500^2 \times 8}{1500} * .04ms\right) \\
&\approx 145ms + 10000ms + 90ms \\
&\approx 10235ms
\end{aligned}
$$

6.2.3 Conclusion

PRIMM gains speed up in most experimental cases as shown bellow 6.8. The very beginning experiments show an inefficient performance of PRIMM, while the very end experiments show more satisfying results.

$$
\begin{aligned}
&\tfrac{T_{Sequential}}{T_{Grid}} \geq 1 \\
&\text{No: } 1 \neq \tfrac{10000}{12000} < 1 \\
&\text{No: } 2 \equiv \tfrac{20000}{17000} \geq 1 \\
&\text{No: } 3 \neq \tfrac{10000}{20400} < 1 \\
&\text{No: } 4 \equiv \tfrac{30000}{26700} \geq 1 \\
&\text{No: } 5 \equiv \tfrac{70000}{45100} \geq 1 \\
&\text{No: } 6 \equiv \tfrac{80000}{51400} \geq 1 \\
&\text{No: } 7 \equiv \tfrac{130000}{65100} \geq 1 \\
&\vdots \\
&\text{No: } 29 \equiv \tfrac{3230000}{2493000} \geq 1 \\
&\text{No: } 30 \equiv \tfrac{3560000}{2939000} \geq 1
\end{aligned}
\tag{6.8}
$$

$$
\hookrightarrow \tfrac{\sum_{i-1}^{30} t_{Seq_i}}{\sum_{i-1}^{30} t_{Grid_i}} = 42.81893323 \Rightarrow 42.81893323 \geq 30
$$

The problem size in table 6.2 is the sum of all the elements that have been included in the computations. The time in table 6.2 is the time of the execution in milliseconds. The weighted time is measured based on the problem size.

The non weighted speedup is measured by summing all the thirty experiments in both the sequential and PRIMM implementation. The table 6.2 shows the sum of the total execution time in the weighted and non weighted cases. PRIMM was able to gain speed up around 30%.

$$
\begin{aligned}
\text{Non Weighted Speedup} &= \tfrac{\sum Time_{Seq.}}{\sum Time_{PRIMM}} \\
&= \tfrac{33170000}{25604700} \approx 1.3
\end{aligned}
\tag{6.9}
$$

The weighted time is the time divided by the problem size, basically we are normalizing the time by the problem size. The normalization gives a measure of how efficient the program is at each size and does not disregard smaller problems as an

Problem Size	Time in Seq	Weighted Seq	Time in PRIMM	Weighted PRIMM
5000	10000	2.00	12000	2.40
7500	20000	2.67	17000	2.27
15000	10000	0.67	20400	1.36
20000	30000	1.50	26700	1.34
25000	70000	2.80	45100	1.80
37500	80000	2.13	51400	1.37
45000	130000	2.89	65100	1.45
52500	170000	3.24	94300	1.80
70000	180000	2.57	102000	1.46
80000	250000	3.13	138000	1.73
90000	320000	3.56	190000	2.11
112500	370000	3.29	194000	1.72
125000	500000	4.00	254000	2.03
137500	560000	4.07	365000	2.65
165000	670000	4.06	362000	2.19
180000	780000	4.33	444000	2.47
195000	910000	4.67	584000	2.99
227500	940000	4.13	588000	2.58
245000	1280000	5.22	890000	3.63
262500	1340000	5.10	1096000	4.18
300000	1550000	5.17	1106000	3.69
320000	1530000	4.78	1285700	4.02
340000	1710000	5.03	1559000	4.59
382500	1850000	4.84	1570000	4.10
405000	1990000	4.91	1820000	4.49
427500	2500000	5.85	2166000	5.07
475000	2700000	5.68	2179000	4.59
500000	3230000	6.46	2493000	4.99
525000	3560000	6.78	2939000	5.60
577500	3940000	6.82	2960000	5.13
Sum	33170000	120.35	25604700	87.38

Tab. 6.2: Weighted Values Based on the Problem Size for both the Sequential and PRIMM Implementations

unweighted sum does. The speedup is smaller than the unweighted case as the cost of the initial setup is more visible in weighted smaller problems, but it does show that PRIMM is more efficient. For large problems asymptotic speedup can be expected to exceed 30%, as is seen in the unweighted case.

Bellow is the weighted speed up 6.10, PRIMM can gain approximately around 40%.

$$\begin{aligned} \text{Weighted Speedup} &= \frac{\sum WTime_{Seq.}}{\sum WTime_{PRIMM}} \\ &= \frac{120.35}{87.38} \approx 1.4 \end{aligned} \tag{6.10}$$

6.3 Parallel Search

6.3.1 Problem Description

PRIMM uses the implementation of parallel search to perform shortcutting remotely. This implementation shows that PRIMM is capable of reducing the execution time and performing better resource management than many other systems that do not have the remote shortcutting technique.

The sequential search could be parallelized to some extend by distributing the search ranges in a controlled manner. Many processes could execute a search in which each process will run for specific range that is different from the other neighboring processes. The search array is randomly generated and the search key must exits always in the array. The master process or process number one declares the result.

6.3.2 Results and Discussion

The table 6.3 shows the execution time of parallel search over half million elements. The search key is variable to show it's effect on shortcutting and to be able to compare the three executions. The search key is passed to the parallel implementation. In PRIMM, the search key is passed to the parallel implementation through PRIMM's server, the key is carried in a UDP packet from PRIMM's client with the

Category	Key	Number of Processes	PRIMM Non Shortcutting (ms)	PRIMM Shortcutting (ms)	Parallel (ms)
1	2	2	31	15	7
2	2	4	39	24	7
3	2	8	50	24	8
4	2	16	47	11	7
5	2000	2	42	13	7
6	2000	4	45	20	7
7	2000	8	62	25	7
8	2000	16	89	22	7
9	70000	2	44	16	8
10	70000	4	31	14	7
11	70000	8	78	23	7
12	70000	16	45	22	7
13	155002	2	19	22	8
14	155002	4	32	32	7
15	155002	8	62	59	7
16	155002	16	63	41	7
17	240000	2	20	34	8
18	240000	4	37	40	7
19	240000	8	39	39	7
20	240000	16	98	47	8

Tab. 6.3: Parallel Search 250000 Elements. PRIMM Non Shotcutting , PRIMM Shortcutting and Parallel Implementation, Time in Milliseconds

program name (parallel implementation) and the number of parallel processes desired.

This thesis concentrates on showing that shortcutting methodologies in grid computing has a high potential of decreasing the execution time by a significant factor. Based on the results from the tables 6.3 and 6.4, PRIMM shortcutting saves more time and resources than PRIMM non-shortcutting; this could be also the case for many other grid computing systems that run their communication over the World Wide Web. The PRIMM shortcutting technique outperforms the PRIMM non-shortcutting implementation significantly.

The results in the two tables 6.3 and 6.4 represent two different series of experiments at different times. The difference in values between the non-shortcutting PRIMM and shortcutting PRIMM show a better performance in both **P**roblem sizes, P = 0.25M and P = 0.5 M. Notice the differences in results between the PRIMM shortcutting in the P = 0.25M array and P = 0.5 M array, this difference could be based on some factors that can affect the computations or the communication between

Category	Key	Number of Processes	PRIMM Non Shortcutting (ms)	PRIMM Shortcutting (ms)	Parallel (ms)
1	2	2	32	12	16
2	2	4	42	22	17
3	2	8	55	28	17
4	2	16	61	12	17
5	2000	2	32	12	17
6	2000	4	22	21	17
7	2000	8	19	19	17
8	2000	16	47	33	17
9	70000	2	44	20	17
10	70000	4	31	14	17
11	70000	8	77	13	17
12	70000	16	47	30	17
13	155002	2	46	36	17
14	155002	4	50	23	17
15	155002	8	65	23	17
16	155002	16	88	30	17
17	240000	2	54	34	17
18	240000	4	46	34	17
19	240000	8	45	36	17
20	240000	16	68	29	17

Tab. 6.4: Parallel Search 500000 Elements. PRIMM Non Shotcutting , PRIMM Shortcutting and Parallel Implementation

PRIMM's client and PRIMM's server.

One expected factor is the Internet (ISP) connection. Since it is well known that the Internet is an unreliable network that can be affected by many hidden factors, it is assumed that the packets transferred between PRIMM's client and server take multiple routes. It is also important to consider the possibility that the parallel cluster may have been busy with other jobs from different users as this adds some randomness to the availability of resources. If one of the nodes in the parallel cluster has been running slow then that may contribute in slowing down the whole implementation. This is also be the case in comparing PRIMM non-shortcutting at P = 0.25 M array and P = 0.5 M array.

The worst case scenario is when shortcutting doesn't improve the execution time or even adds an overhead to the parallel execution itself. This is possible when the key is always found at the end of one of the distributed arrays. However, in the case of finding the key in the middle of one of the distributed arrays, the execution

time may be decreased by a satisfying value. The best case scenario is finding the key at the beginning of one of these distributed arrays. In this case the execution time may be dramatically decreased. See table 6.3 and 6.4.

However, PRIMM concentrates on showing the usefulness of shortcutting methodologies in grid computations. It does not contradict the fact that PRIMM shortcutting still has a potential to reduce the execution time of parallel implementations. However, PRIMM with shortcutting had a worse performance in many cases than the parallel implementation, refer to tables 6.3 and 6.4. Realize that the five cases that PRIMM shortcutting performed better than the parallel implementation were in the P = 0.5 M parallel search, refer to the second table 6.4. The results give merit to the notion that the increase in the number of computations promotes the possibility of saving time. For example shortcutting a parallel search problem with size of a half-million, may save time over the same search with a problem size of a quarter-million. In conclusion, the shortcutting technique in parallel and grid implementations could be very useful in some cases where the problem size is bigger than a certain threshold that overshadows the intrinsic overhead of the commutations occurring.

In the tables 6.3 and 6.4 the differences between PRIMM shortcutting and PRIMM non-shortcutting could be a result of multiple variables, for example the variance may be caused because the shotcutting has spent short amount of time buffering, while the non-shortcutting has been buffering until all the processes finish. The shortcutting result is carried after 24 byte streaming from the operating system. While the non-shortcutting packet is carried after the stream has completely finished and terminated. Another important factor is the number of UDP packets transferred between the client and the server. In the shortcutting mode it is lesser compared to the number of packets in the non-shortcutting mode (in most cases) , because PRIMM client and server transfer protocol-based packets every short period of time. These protocol-based packets may cause a short delay in sending the packet that carries the parallel implementation result.

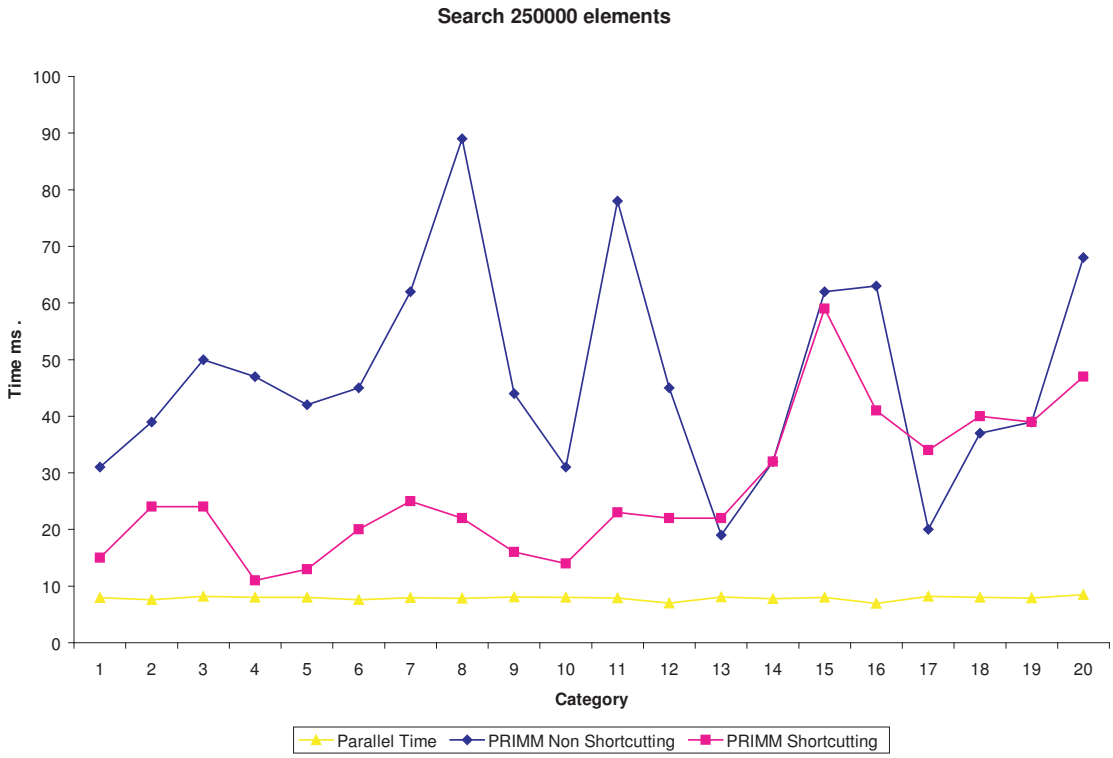

Fig. 6.3: Search Results for the Three Implementations on 250000 Elements Array

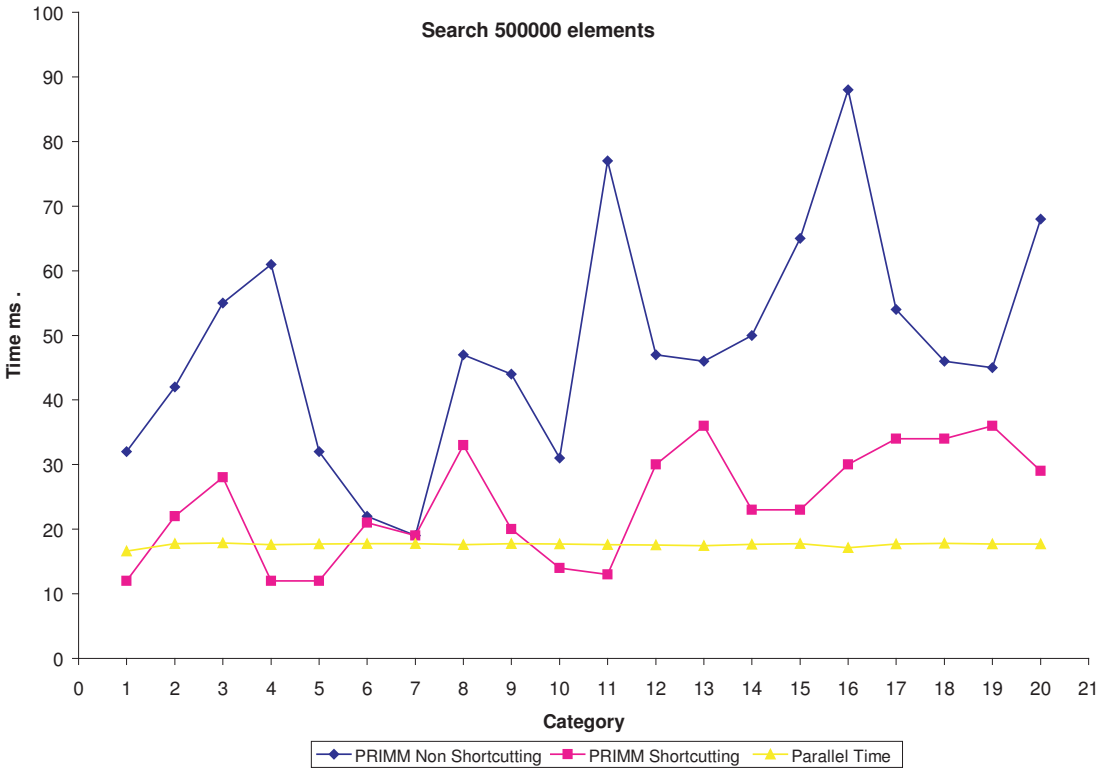

Fig. 6.4: Search Results for the Three Implementations on 500000 Elements Array

The factors discussed lately may also affect the system performance. The streaming process may cause a little overhead on the local machine itself. That could also include all other processes that PRIMM is running. The longer the streaming process stays the more resource cost will be consumed. The PRIMM client also has overhead on the other side, even if the client is not involved in the calculations itself, it still consumes time to receive, validate or parse the PRIMM header and finally declaring that the final result has been received.

6.3.3 Conclusion

The speed up is computed based on the results in the summation of all experiments. Speed up is compared with a grid implementation that has shortcutting methodologies with another one that doesn't. The problem that has been used is considered a straight forward problem, in which the idea of shortcutting can be understood easily.

For 250000 elements array:

$$\begin{gathered} \sum Shortcutting = 543ms \\ \sum NonShortcutting = 943ms \\ Speedup = \frac{\sum NonShortcutting}{\sum Shortcutting} \\ \hookrightarrow \frac{24938}{2077} \approx 1.7 \end{gathered} \tag{6.11}$$

For 500000 elements array:

$$\begin{gathered}\sum Shortcutting = 481ms \\ \sum NonShortcutting = 971ms \\ Speedup = \frac{\sum NonShortcutting}{\sum Shortcutting} \\ \hookrightarrow \frac{971}{481} \approx 2.0\end{gathered} \tag{6.12}$$

The speed up for all the parallel search experiments, the following is based on equal ratios for problem sizes, the half million array is two quarter a million arrays:

$$\begin{gathered}\sum Shortcutting = 0.5 * 481 + 543 \\ \sum NonShortcutting = 0.5 * 971 + 943 \\ Speedup = \frac{\sum NonShortcutting}{\sum Shortcutting} \\ \hookrightarrow \frac{1428.5}{783.5} \approx 1.82\end{gathered} \tag{6.13}$$

6.4 Conclusion

The PRIMM's throughput rate is defined as the average output of data per unit of time. This rate includes all the variables in T_{Grid} formula 6.14. It does include PRIMM's server processing time **S** and PRIMM's client processing time **C**. The relationship between the server processing time and the client processing time is tricky too, be aware that the server or the client may not participate in the same rate of processing. The variable **P**acket **V**alidating Time **PV** in the server side may be insignificant if compared to the **PV** in the client side, because the client has to validate all resulted output packets from the server while the server has only limited number of packets that carries the control commands.

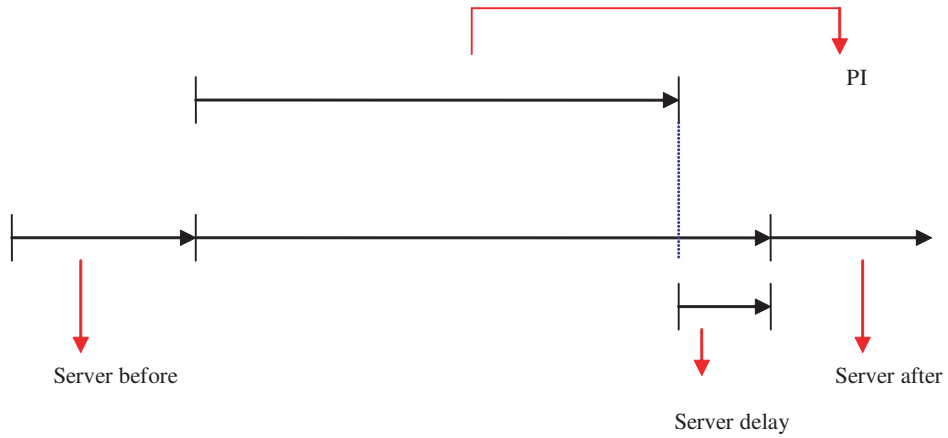

Fig. 6.5: Server Overhead Time

Consider **P**arallel **I**mplementation time PI is the parallel implementation time without PRIMM server being running, **N**etwork **L**atency NL, PRIMM's **S**erver latency **S** and PRIMM's **C**lient overhead C. **S** is the time consumed from the server before the parallel implementation starts and after the termination of the parallel implementation. This variable does not include the parallel implementation time **PI** So **S** is defined by the extra time consumed on top of the parallel implementation time. S_Before is the time when the packet carrying the request received to the time when the parallel implementation started. S_After is the time when the parallel implementation terminated to the time when the final result is sent to the client. S_{Delay} is the delay time that PRIMM server may cause when running at the same time while the parallel implementation is still in execution. This mean that PRIMM server could slow one node that has been participating in the parallel implementation. In the experiment it is Ravan0. Total execution **T**ime for the grid T. Please see figure 6.5. This could help in concluding when PRIMM can be more efficient and feasible for a general problem, for example the variables that contributes in performance overhead could be insignificant if the computation load is large enough.

$$T_{Grid} = PI + NL + S + C$$
$$S = S_{After} + S_{Before} + S_{Delay} \tag{6.14}$$

In order to realize PRIMM's feasibility, it is required to isolate the **PI** variable. **PI** represents the parallel time for the parallel cluster implementation, in which does not include any communication with the outside world. Control commands transferred between the client and the server are not included in **PI**. PI here will roughly represent the time for the MPI-cluster implementation.

$$NL + S + C = T_{Grid} - T_{Parallel} \tag{6.15}$$

PRIMM becomes more feasible system if the variables NL,S and C are insignificant to PI. It means that when the amount of computations is very high compared to the total overhead of NL,S and C.

PRIMM is capable of reducing the execution time of the sequential implementations. One technique, shortcutting, could be utilized to reduce the execution time in grid computations by a significant factor. Example, PRIMM shortcutting and PRIMM non shortcutting.

PRIMM is not the only grid system that has to deal with considerably higher network latency; many other grid systems have the same issue when running their communication over the Internet (ISP). PRIMM may have better potential for better performance in some cases than others, because it uses UDP rather than TCP. PRIMM also provides methodologies to allow the server to perform real time streaming to the client, which could be significant to such applications that require data for managerial decisions.

PRIMM is a pure OOP implementation, which translates into cost savings, relative to other implementations; especially, when frequent adaptations are required. In addition, it supports shortcutting and runtime execution management.

REFERENCES

[1] Java parallel processing framework. *http://www.jppf.org*.

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

[3] George Karypis Ananth Grama, Anashul Gupta and Vipin Kumar. *Introduction to Parallel Computing*. Addison-Wesley, 2nd edition, 2003.

[4] Apple. Mac os x - x grid. *http://www.apple.com/macosx/features/xgrid/*.

[5] Sammy Raymond D'Souza. Parallelizing a non-deterministic optimization algorithm. Master's thesis, California Sate University, San Bernardino, 2007.

[6] I. Foster. What is the grid? a three point checklist. *Argonne National Laboratory*, 2002.

[7] Gigi Karmous-Edwards Franco Travostino, Joe Mambretti. *Grid Networks and TCP Services, Protocols, and Technologies*. John Wiley and Sons, Ltd, 2006.

[8] Sun Grid from Sun Microsystems. *http://www.sun.com/service/sungrid/*.

[9] Ernesto Gomez. The sos library. *http://csci.csusb.edu/egomez/html-res/sos.html*.

[10] Ernesto Gomez and L. Ridgway Scott. Overlapping and shortcutting techniques in loosely synchronous irregular problems. *LNCS*, 1457:116–127, August 1998.

[11] Yunhong Gu and Robert L. Grossman. An application level transport protocol for grid computing. *Laboratory for Advanced Computing / National Center for Data Mining*.

[12] Innovative Computing Laboratory. Gridsolve and netsolve. *http://icl.cs.utk.edu/netsolve/*.

[13] Michael Miley. The grid. *Oracle Magazine*, 2003.

[14] Message Passing Interface official organization forum. Mpi documentation. *http://www.mpi-forum.org*.

[15] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, third edition, May 2003.

[16] David Reilly and Michael Reilly. *Java(TM) Network Programming and Distributed Computing*. Addison-Wesley Inc, first edition, Mar 2002.

[17] Matti A. Hiltunen Richard D. Schlichting Ryan X. Wu, Andrew A. Chien and Subhabrata Sen. A high performance configurable transport protocol for grid computing. *ACM*, 2:1117–1125, 2005.

[18] Mukesh Singhal and Niranjan G.Shivartri. *Advanced Concepts in Operating Systems*. McGraw-Hill, 2002.

[19] NorduGrid System. *http://www.nordugrid.org/middleware/*.

[20] UNICORE Grid System. *http://www.unicore.eu/*.

[21] Takemiya H. Nakada N. Tanaka, Y. and Sekiguchi S. Design, implementation and performance evaluation of gridrpc programming middleware for a large-scale computational grid. *IEEE*.

[22] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, fourth edition, 2002.

[23] Michael Philippsen Vladimir Getov, Gregor von Laszewski and Ian Foster. Multiparadigm communications in java for grid computing. *ACM*, 44(10):118–125, 2001.