PEER-TO-PEER SUPPORT FOR MATLAB-STYLE COMPUTING

A Thesis

by

RAJEEV AGRAWAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2004

Major Subject: Interdisciplinary Engineering

PEER-TO-PEER SUPPORT FOR MATLAB-STYLE COMPUTING

A Thesis

by

RAJEEV AGRAWAL

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

May 2004

Major Subject: Interdisciplinary Engineering

ABSTRACT

Peer-to-Peer Support for Matlab-Style Computing. (May 2004)

Rajeev Agrawal, B.E., University of Pune

Chair of Advisory Committee: Dr. Riccardo Bettati

Peer-to-peer technologies have shown a lot of promise in sharing the remote resources effectively. The resources shared by peers are information, bandwidth, storage space or the computing power. When used properly, they can prove to be very advantageous as they scale well, are dynamic, autonomous, fully distributed and can exploit the heterogeneity of peers effectively. They provide an efficient infrastructure for an application seeking to distribute numerical computation. In this thesis, we investigate the feasibility of using a peer-to-peer infrastructure to distribute the computational load of Matlab and similar applications to achieve performance benefits and scalability. We also develop a proof of concept application to distribute the computation of a Matlab style application.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A. Introduction

The technological race has made it easier for the common man to own a computer and connect to remote computers to access resources available on them at an affordable price. The types of machines vary from the desktop personal computers and supercomputers to wireless phones and handhelds. Not all these resources are fully utilized at all times. This has led to the drive to make efficient use of the unused resources for scientific purposes, which gave rise to parallel computing. Traditionally, the parallel computing infrastructures require special-purpose machines like supercomputers and they do not solve the problem of resources lying at different locations. Gradually, the growth in processing power of the computers and the increasing bandwidth has paved the way to distributed computing in which several interconnected computers share the computing task assigned to the system [1]. Most distributed computing systems have been using the message passing libraries like PVM and MPI, remote procedure calls (RPC), remote method invocation or distributed-shared memory paradigms for inter-node communication. The issues common to all the distributed computing infrastructures are heterogeneity, scalability, transparency, security and fault-tolerance. Researchers in many field use distributed computing infrastructures to process their data faster using some numerical computing application. We want to use the peer-to-peer distributed computing paradigm that will be used by numerical computation software to distribute the computation.

B. Objectives

The research objectives of this thesis are to investigate the feasibility of porting existing large-scale numerical and mathematical software to a peer-to-peer infrastructure and satisfy the following three requirements:

The journal model is *IEEE Transactions on Automatic Control.*

1. The migration to the peer-to-peer platform should be scalable, manageable, easy to perform (i.e. an application should be easily deployable),
2. It should be easy to migrate a large number of existing applications to the peer-to-peer platform, and
3. The migrated computation should make efficient use of the underlying computational infrastructure and give the expected performance improvements.

The approach in this thesis will be empirical: we will build and evaluate a proof-of-concept that will indicate how the three requirements above can be satisfied.

C. Large-Scale Numerical and Mathematical Software

Distributed computing infrastructures are used in the scientific research field where a huge amount of data has to be processed. People working in these fields use some numerical computation applications for data processing, which lets them concentrate on the main issues of the research. There are several numerical computation software available in the market to help researchers process data faster. There are commercial software like Matlab, Mathematica, Maple and freely available software like Octave and Scilab. MATLAB (by The MathWorks) is one of the most common languages for numerical computing in the scientific and engineering community. It provides features like data acquisition, data analysis and exploration, visualization and image processing, algorithm prototyping and development, modeling and simulation, and programming and application development [2]. According to The MathWorks [3], there are currently more than 500,000 users of MATLAB worldwide and is used by more than 2000 universities.

Using commercial numerical computation software comes with its own price. For example, as of January 2004, it costs $1900.00 for Matlab Release 13 with Service Pack 1 CD, $500.00 for academic use and $59.00 for the student version. The high price of Matlab makes its use difficult on some distributed computing infrastructure where each node needs a local copy of the software. Also, the source code for the commercial numerical applications is not available, i.e., they are not open-source. To overcome these limitations, researchers have been looking at many open-source options to Matlab that offer much better cost-to-performance ratio even though they are not fully compatible with Matlab m-files. Some of these Matlab like applications are Scilab, Octave, Rlab and Euler [4].

Euler is one such MATLAB clone that can handle real, complex and interval numbers, vectors and matrices [5]. It is available for free under the GNU general license and is written in C. It provides a programming language similar to Matlab. It is a good representative of a MATLAB-like application.

The limitation of many of these numerical computation applications is that they can be run only on a single machine. The exponential increase in the amount of data to be processed still exceeds the processor speed increase and so there are many fields where lots of time is spend on numerical computation. A rational approach to solve this problem is to distribute the load of one processing element to others using any of the distributed computing technologies. Ideally this distribution should be such that it minimizes the time taken to process the work and additionally allows the maximum utilization of each processing element.

D. Distributed Computing Systems

The distributed computing systems can be divided into two classes depending upon the hierarchical relationship among the participating nodes [6]:

- Client-server
- Peer-to-peer

E. Peer-to-Peer Distributed Systems

According to [6], "the term peer-to-peer refers to a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner". The main feature, which distinguishes a peer-to-peer from the client-server system, is the transient nature of the participating machines. Any peer can act as a client of a server and can join or leave the network dynamically. Peers share resources that mainly are computing power, space, information and bandwidth. Peer-to-peer technologies can be used to make efficient use of these resources. Peer-to-peer computing was crowned by Fortune as one of the four technologies that will shape the future of the Internet [7].

A peer-to-peer system can be thought of as a Very Large Virtual Machine (VLVM). Each individual peer is plugged into this virtual machine and forms a small component of this virtual machine. When the resources of all the peers are aggregated, it takes the form of a VLVM as shown in Figure 1.

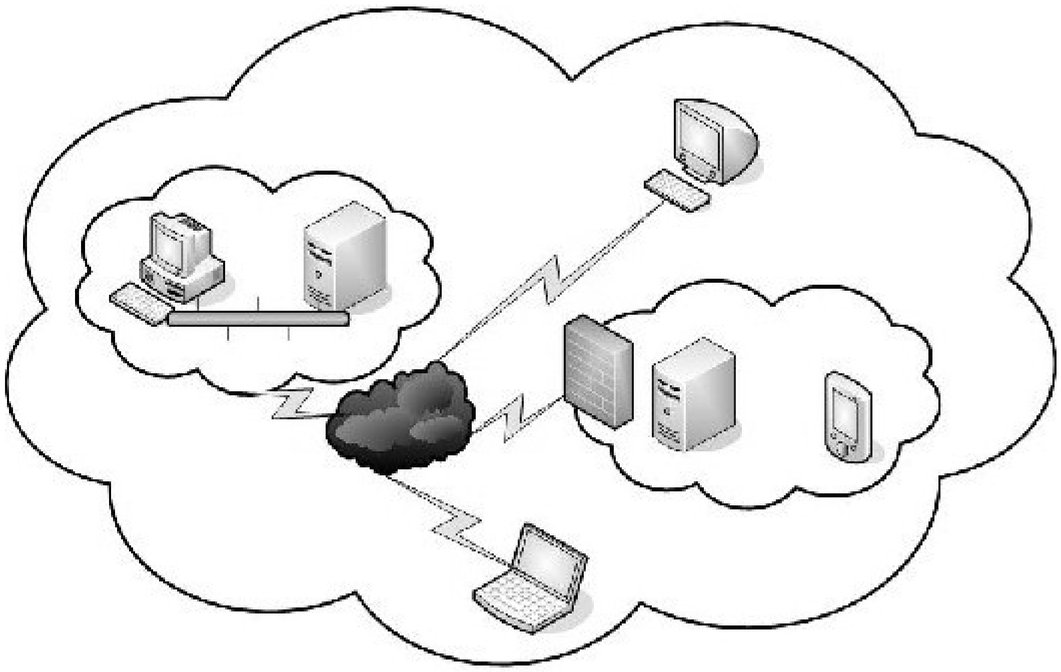

Fig. 1. A Peer-to-Peer System as a Very Large Virtual Machine (VLVM)

F. Advantages of a Peer-to-Peer System

The advantages of the peer-to-peer system lie in their high scalability, high decentralization and high performance when resources of all the peers are aggregated [6]. This makes peer-to-peer systems a good candidate for being used as a platform for high performance numerical computation applications [8]. Also, with time, the edge devices (desktop computers, handhelds, cell phones) have become powerful and when aggregated, their resources can be very helpful. A peer-to-peer framework helps in increasing the involvement of these edge devices to work on bigger problems. A peer-to-peer system provides dynamic ways to locate peers and resources.

G. Disadvantages of a Peer-to-Peer System

The disadvantages of a peer-to-peer system are that due to its dynamic nature, the peer-to-peer infrastructure is less reliable, as there is no control over the peers joining and leaving the network. But this problem can be overcome by the redundancy in the data and peers. For example, in a collaborative computing peer-to-peer infrastructure, instead of giving a part of job to a single peer, it can be given to multiple peers thus ensuring the higher probability of return of results. In the same way, in a data sharing peer-to-peer infrastructure, the data can be stored with multiple peers instead of one. Due to the lesser reliability of a peer being available in a peer-to-peer system, there are lesser guarantees about the QoS (Quality of Service). Also since the peers in a peer-to-peer system are more autonomous, it becomes difficult to have a central system to manage them.

H. Outline

This thesis report is organized as follows. In Chapter II, we outline the various issues related to the design of a peer-to-peer distributed computing infrastructure and distributed computing in general. Chapter III presents the related work done in this field. Chapter IV gives an introduction to how JXTA works and explains how JXTA deals with the issues related to our objectives. In Chapter V, we explain the implementation of our infrastructure. In Chapter VI, we provide the results of our implementation. Chapter VII presents a summary and final conclusion of our work.

CHAPTER II

PEER-TO-PEER DISTRIBUTED COMPUTING

In the design of an efficient distributed computing infrastructure, a lot of issues have to be taken into consideration. In this chapter, we discuss the issues related to peer-to-peer distributed computing.

A. Requirements Related to the Objectives

According to the objectives laid out in the previous chapter, a peer-to-peer platform we choose for this work should satisfy the following requirements.

The migration to the peer-to-peer platform should be scalable, manageable and easy to perform. The peer-to-peer should be chosen in such a way that if the scale of the system changes, its performance does not deteriorate (scalability). It should be easy for a user to install the software without bothering about the technicalities of the software (manageability) and work in restrictive conditions (for example, behind the firewall). The migration to the peer-to-peer platform is justified only if the gain by porting the application to the peer-to-peer platform exceeds the effort to port the application. The gain can be realized in terms of increased scalability and performance, anonymity, decentralization etc.

It should be easy to migrate a large number of existing applications to the peer-to-peer platform. The peer-to-peer platform should be such that it is easy for most of the common existing numerical applications to be ported to the peer-to-peer platform. Our application should provide standard features and interfaces and it should be easy to add new functionalities. The peer-to-peer platform should make the application interoperable.

The migrated computation should make efficient use of the underlying computational infrastructure and give the expected performance improvements. The new application with the peer-to-peer base should be efficient and utilize all the resources of the available peers. It should also show some perceivable gains like the speedup and bandwidth utilization and should be fault tolerant.

B. Issues with Peer-to-Peer Distributed Computing

Peer-to-peer systems share most of the issues in designing a distributed computing system. According to [6] [9] [10], an efficient peer-to-peer system (and a distributed system) should address the following issues: decentralization, scalability, heterogeneity, anonymity, cost of ownership, dynamism, performance, security, peer management, fault-resilience, self-organization, resource discovery, administrative issues and transparency. In context with our objectives, we can group these issues according to the objectives they affect:

1. The migration to the peer-to-peer platform should be scalable, manageable and easy to perform.

   The issues affecting objective 1 are:

   Scalability: A good peer-to-peer system should scale well with the number of nodes and increasing resources. Some resources could be in high demand. To increase the scalability, these heavily used resources could be replicated or cached [10].

   Peer management: The identity of a peer and the ability of a peer-to-peer system to group the peers into meaningful groups to perform specific functions are critical to the peer-to-peer system. The management of peers should be easy to perform.

   Self-organization: With the increase in the number of peers and their dynamic nature, the chances of failure of node or communication also increase. In those cases, the peers should be able to adapt to new changes and not fail.

   Resource discovery: The peers should be able to find other peers on the network and share resources with them easily and should not require any interference by the user. The overhead required in doing this should not deteriorate the performance of the system.

   Transparency: According to [10], an ideal distributed system should hide the separate components from the user and the applications to view the system as a whole. The advantage of transparency is that it eases the manageability of the peer-to-peer system.

2. It should be easy to migrate a large number of existing applications to the peer-to-peer platform.

The following issues affect objective 2:

Heterogeneity: Any network aware device can act as a peer in the peer-to-peer system and so the peer-to-peer infrastructure should take care of the difference in network topologies, operating systems, hardware and programming languages [10].

Cost of ownership: With the peer-to-peer system, there is no need for specialized machines to act as servers to all the peers as other peers also provide that service. This reduces the load on that particular peer without reducing the quality of service and driving the cost of ownership down drastically. A peer-to-peer system should be cost effective so that a user can be benefited from migrating to a peer-to-peer system.

Administrative issues: According to [9], the combined use of computers spread across several countries calls for an acceptable use of these resources for a good cause and the involved parties should agree on the common definition of "good cause". An acceptable use means that if a user is logged on a machine, his interactive session should not suffer from heavy use by other peers.

Security: Since all the peers have access to all the other peers, the security in a peer-to-peer system proves to be a real challenge as peers are more vulnerable than the traditional computing systems. To eliminate this problem, reliable means for authentication and authorization should be used.

Anonymity: In a peer-to-peer system, the participating nodes are not dependent upon some particular peer for a service since each peer is both a client and a server. This provides an added feature of anonymity to the peers. An ideal peer-to-peer system should provide good anonymity for its peers.

3. The migrated computation should make efficient use of the underlying computational infrastructure and give the expected performance improvements.

   The issues relevant to objective 3 are:

   Performance: The use of a peer-to-peer system is advantageous only if the peers give an acceptable performance when collaborating in the peer-to-peer network. For example, if the messaging overhead in a peer-to-peer system is high, this may overburden the network and increase network latency, deteriorating the performance of the peer-to-peer system and this is undesirable.

Decentralization: Decentralization is the main feature that distinguishes a peer-to-peer system from other traditional distributed computing architectures. A good peer-to-peer system should be fully decentralized so that there are no centralized points in the system that will fail. This increases the reliability of the peer-to-peer system and gives better performance.

Dynamism: It is common for peers to dynamically join or leave the peer-to-peer network. A good peer-to-peer system will know about the availability or unavailability of the peer as soon as it happens and take measures to prevent the failure.

Fault-resilience: Fault tolerance is important to detect node and communication failure. It is also important if we need to provide reliability and QoS (Quality of Service) guarantees. A peer-to-peer system should have mechanisms to provide fault tolerance (providing checkpoints, replication and logging), which will increase the performance.

Prevention of free riders: Free riders use the resources provided by other peers but do not provide any resources or services to other peers. A peer-to-peer system should be able to identify free riders and discourage this behavior as it deteriorates the performance of the peer-to-peer system.

CHAPTER III

RELATED WORK

This chapter cites the work done in this field and approaches used to parallelize numerical computation application. Since Matlab is the most commonly used application for numerical computation, most of these approaches focus on it. Approaches used to parallelize other Matlab like and Matlab compatible applications have also been discussed in this chapter.

A. Approaches to Parallelize Numerical Computation Applications

There have been many attempts to parallelize Matlab and Matlab like numerical computation software, both commercial and freeware. Most of them achieve the coarse-grain "outer-loop" parallelization [11]. Some of the other Matlab like software are Mathematica, Maple, Scilab and Octave. Matlab proves to be a difficult candidate to parallelize. According to [12] [13], the various reasons for there not being a parallel version is:

1. Most of the Matlab functions have fine granularity that cannot be effectively parallelized on the distributed architecture.
2. Matlab's sophisticated memory model and architecture conflicts with the small number of computers in the shared memory computers.
3. Not many consumers own a parallel machine to effectively take advantage of the parallel Matlab functionalities.
4. More time is spent on the Matlab parser, interpreter and the graphics routine and it is difficult to parallelize these functionalities.
5. To parallelize the easy for loops, a basic change in the Matlab architecture will be required which is not feasible.

The approaches to parallelize numerical computation can be divided into several categories that are as follows.

Based on the communication paradigm, the approaches to parallelize the numerical computation software can be divided into three categories:

1. Using message passing routines

   This approach to parallelize numerical computation applications uses message-passing libraries like MPI (Message Passing Interface) and PVM (Parallel Virtual Machine) to talk to more than one instance of the numerical computation software. A few of the implementations that try to parallelize Matlab in this way are MultiMATLAB, MPITB (MPI Toolbox), PVMTB (PVM Toolbar) and MatlabMPI. This approach requires an overhead of first setting up and installing the message passing libraries. The use of message passing libraries create some additional overhead but for situations where numerical computation time exceeds the communication time by a few order of magnitudes (coarse granularity), these overheads can be neglected. In case of Matlab, the interface between Matlab and the message passing libraries can be through MEX files. These files contain sub-routines written in C or Fortran. They are compiled using the Matlab compiles and the subroutines can be called as built-in functions form Matlab.

2. Using distributed shared memory (DSM) model

   This approach is used by MatMARKS to provide a higher level (shared memory) abstraction between numerical computing applications and message passing [14]. The application behaves as if it is executing on a shared memory multiprocessor and accessing shared data and the responsibility of passing messages to other processes is left to the distributed shared memory system [15]. This allows the programmer to use the SPMD (single program multiple data) programming model. MatMARKS is based on the TreadMarks DSM. The communication overhead in this case is little more than the message passing libraries.

3. Others communication paradigms

   These approaches use inter-node communication using remote procedure calls (RPC) and pipes. Pipe is used by Matlab Parallelization ToolkIt 1.20 [16] and is based on the Master/slave paradigm. The master can start remote Matlab sessions on local/remote slaves and they communicate with each other using TCP/IP pipes. This paradigm is suitable where low inter-process communication is expected. The inter-node communication in MATLAB*P is using RPC.

Depending upon the type of software used by the remote servers, these attempts can be divided into:

1. Use of the standalone numerical applications.

   In this setup, the remote servers to which tasks are sent use the same numerical computational application as the local server for working on a task. This approach is used by MultiMATLAB, MPITB/PVMTB, MATmarks, MatlabMPI, Cornell MultiTask Toolbox [17] and DistributePP [18] to parallelize Matlab. One of the methods to distribute computation used by Scilab is to invoke instances of Scilab processes on the remote machines and communicate through message passing libraries [19]. In case of commercial applications, this could prove to be expensive as many of the commercial applications require a per machine license fee. The service requestor has the capability to spawn the instances of the numerical computational application on the remote machine.

2. Use of parallel numerical libraries.

   This approach requires a single instance of the numerical computation application and uses parallel libraries like ScaLAPACK on remote servers to provide a parallel backend. This approach is used in Matpar, Paramat and MATLAB*P. This approach additionally requires the availability of parallel libraries. This approach is limited to the availability of parallel libraries for the specific platform but is cost effective as it requires only a single instance of the numerical computation application.

3. Use of network computational servers.

   In this setup, the remote servers are standalone Internet servers that provide some type of computational service. Scilab uses this approach to submit the tasks to the NetSolve network solver. NetSolve consists of 3 types of machines: clients, agents and the servers. The clients submit jobs to the agents. The agents have information about the available servers and they dispatch jobs based on the available resources.

Netsolve has a built-in load-balancing scheme and uses the greedy MCT (Minimum Completion Time) algorithm [20]. This approach is also used by MATLAB*P.

According to [17], other approaches to parallelize the numerical computing applications can be grouped under the following categories:

1. Provide routines to split up scripts among multiple Matlab sessions.

   This approach takes advantage of the for loop and other simple repetitive blocks in the MATLAB routines, splits them and distributes it to multiple instances of Matlab running either on the same computer or different computer. Examples of this implementation are MULTI Toolbox, Paralize, PLab, Parmatlab. This approach also requires setting up the message passing libraries to pass the work units between different machines and keeping track of the work units. Again the cost of ownership is high because a user with a single use Matlab license cannot use these systems.

2. Translate and compile Matlab scripts into parallel code.

   This approach works by translating Matlab scripts into native languages like C and FORTRAN and then compiling that native language code and linking it with parallel libraries. This approach also uses one instance of Matlab and is implemented in Otter, FALCON, and CONLAB. This approach is not interoperable as the code has to be compiled for each platform and might require the user to have a sound technical knowledge of using (compiling and linking) parallel libraries.

B. Advantages and Limitations of the Related Work

The following work discusses how the above-mentioned approaches deal with the issues related to distributed computing, their advantages and their limitations.

According to our categorization of these issues in the previous chapter, we can again list these issues according to the objectives as follows:

1. The migration to the peer-to-peer platform should be scalable, manageable and easy to perform.

   Scalability: Since most of the approaches are client-server based, they do not scale very well. This is because the bandwidth proves to be a limitation. Too much bandwidth is required at the server end as the number of clients increase. Hence they do not satisfy the first requirement of our objectives.

Resource discovery: In most of the approaches, the client has prior knowledge of the servers available. In some cases, a master spawns new processes on remote slaves and in this case too, the master should have prior knowledge about the slaves.

2. It should be easy to migrate a large number of existing applications to the peer-to-peer platform.

   Cost of ownership: The cost of ownership in the client-server based architecture (where each node required a local copy of the application) is high, as specialized machines are needed to act as servers.

   Security: Using the message passing libraries like PVM and MPI may make the system less secure depending upon how a machine communicates with the remote machines (rsh, ssh). But schemes have been proposed to provide authentication, data integrity and privacy support in PVM [21].

   Anonymity: The machines involved in task computation are not anonymous, as the master (or client) needs prior information about the slaves (or servers).

   Firewall friendliness: None of the approaches are firewall friendly. They are not designed with these advanced issues in mind.

3. The migrated computation should make efficient use of the underlying computational infrastructure and give the expected performance improvements.

   Decentralization: Most of these systems are client-server (master-slave) based which is a hierarchical computing paradigm and so these approaches are not decentralized. They are vulnerable to a single point of failure at the master node.

   Performance: The communication overhead in these approaches is less. In the message passing approaches, the user decides when the tasks should communicate and hence the design can be very optimal.

   Fault-tolerance: Schemes are proposed to guarantee consistent checkpoints and message logging for the message passing libraries ([22] [23]). This increases the fault tolerance of these applications and help in task migration in case of some failure on the host machine.

In the next chapter, we show how the peer-to-peer networks solve some of these issues efficiently to make heterogeneous distributed computing more scalable and decentralized.

CHAPTER IV

PEER-TO-PEER COMPUTING AND JXTA

A. Introduction to JXTA

As we saw in the previous discussions, the approaches to parallelize numerical computation applications do not scale very well and do not use dynamic resource discovery schemes. This is because the traditional distributed computing models (client/server) do not scale well, are more prone to failure because of dependence on some specific nodes (server, master) and have the bandwidth limitations. To avoid these problems, we turn to other ways to decentralize and scalable computation. For this reason we used a peer-to-peer development framework called JXTA [24].

JXTA is a collaborative research initiative by Sun Microsystems to develop a freely available infrastructure to help develop industry strength peer-to-peer services easily. It is a set of open protocols that allow any connected device on the network to communicate and collaborate in a P2P manner [25]. The connected devices could be any device ranging from cell phones and wireless PDAs to PCs and servers. The JXTA protocols are independent of the programming language. Currently, full implementation of JXTA protocols exist in Java and work is going on to implement it in J2ME, C, Python, Perl and Ruby.

The main components of the JXTA network are peers, peer groups, advertisements, resources, services and pipes (communication mechanism). An identifier string that is guaranteed to be unique over time identifies all these components.

B. JXTA Components

The JXTA infrastructure consists of the following main components:

1. Peers: Any device connected to the JXTA network is called a peer and it can either provide or consume a service.

2. Peer groups: Peers providing similar services/resources and be logically divided into groups that share common interests.
3. Advertisements: Advertisements are used by peers and peer groups to make their services known to other peers. Advertisements are XML based documents describing the details of the service.
4. Resource: Resource could be anything from storage space, information, bandwidth or the computing power.
5. Services: Any functionality that a peer makes available to other peers is called a service. Service can either be offered by a peer to any remote peer (peer services) or by a peer group to the members of the group (peer group services).
6. Pipes: A pipe is a communication mechanism between peers irrespective of the underlying infrastructure and network topology where information is put in at one end of the pipe and received at the other end [26]. The endpoint indicates the input and output points of communication and channels are the connection between the endpoints. The JXTA specification has three pipe types:

   Unicast: Used for one-way, unreliable and insecure communication.

   Unicast secure: Used for one-way, unreliable and secure communication.

   Propagating: Used for one-to-many insecure, and unreliable communication.

C. JXTA Model

The JXTA model works in the following way:

1. Peers join the JXTA network and advertise its resources or services to other peers.
2. The peer sends a query to the JXTA network if it wants to search for some resource/service.
3. If the resource is available, the response is sent back to this peer letting it know how it can access the resource.
4. The peer then contacts the resource/service and uses it.

Each resource in the JXTA network is assigned an ID, which is guaranteed to be unique for any resource in the JXTA network.

D. JXTA Architecture

The JXTA architecture can be divided into three layers [27] as shown in Figure 2.

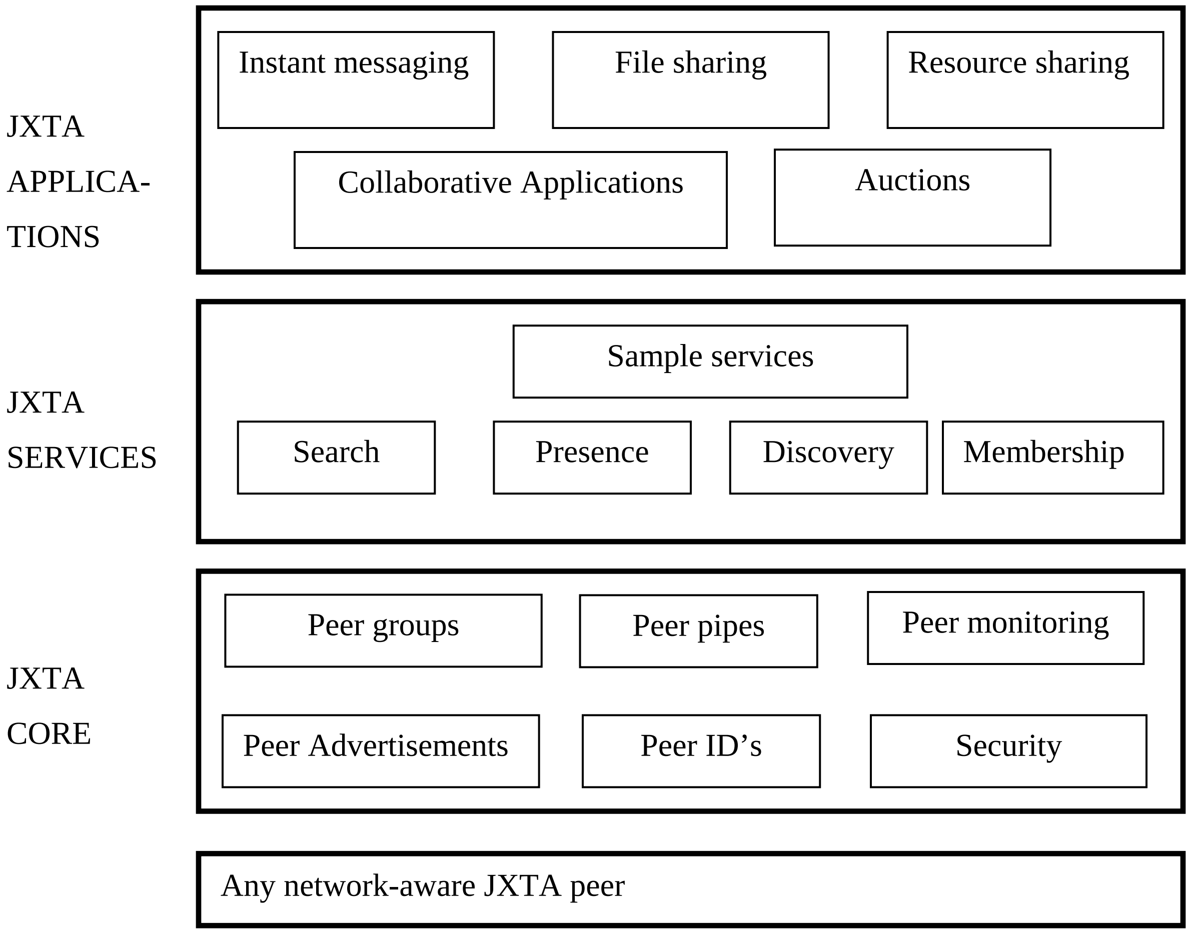

Fig. 2. JXTA Layered Architecture [28]

The three layers are as follows:

1. The core layer

   The core layer provides functionalities that are fundamental to any peer-to-peer infrastructure. These include peers, peer naming, peer groups, protocols for communicating, communication endpoints (pipes) and security.

2. The Services layer

   This layer provides the higher-level functionalities that are advantageous for a peer-to-peer infrastructure but not essential. It uses the core layer to provide these functionalities. A few examples are sharing resources with a peer, discovering other peers etc.

3. The Applications layer

   This layer is built on top of the service layer and helps the developers in providing the functionalities needed by a particular peer-to-peer application. For example, a peer-to-peer application to instant message other peers can be built on this layer.

E. Core JXTA Protocols

JXTA provides a set of 6 core protocols [25] as shown in Figure 3.

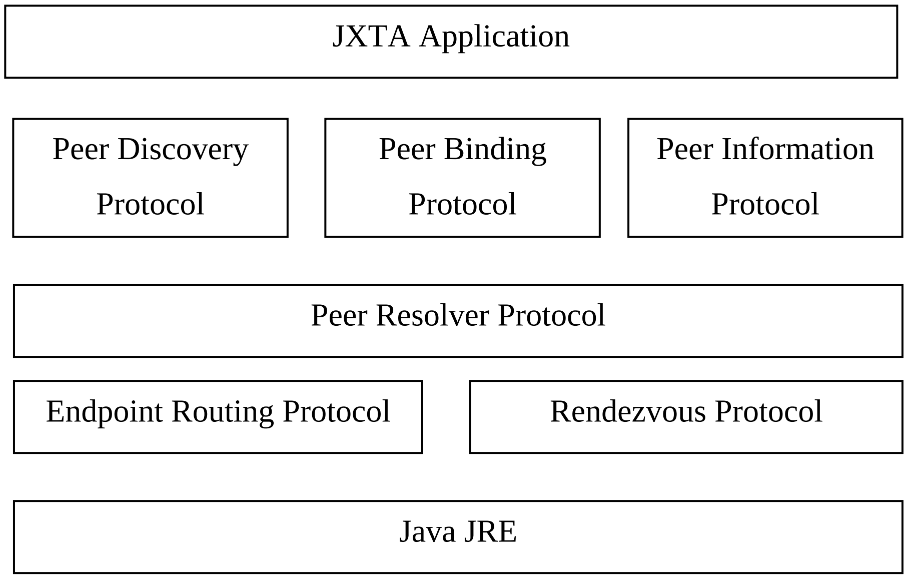

Fig. 3. JXTA Protocol Hierarchy [26]

These 6 protocols are described as follows:

1. Peer Discovery Protocol: Used for discovering peers, peer groups, and any other advertisements. A peer sends a Discovery Query message to discover an advertisement and gets a Discovery Response message in return. This protocol helps in the dynamic discovery of peers and resources and makes the infrastructure more scalable and decentralized.

2. Peer Resolver Protocol: Enables a peer to send and receive generic queries to find or search for peers, peer groups, pipes, and other information. A Resolver Query message is sent to the remote peer and this peers gets back a Resolver Response message. This protocol also helps in the dynamic resource discovery and provides a common platform for peers to share information.
3. Peer Information Protocol: Allows a peer to learn about other peers' capabilities and can obtain status information (one can send a ping message to see if a peer is alive).
4. Rendezvous Protocol: Allows a peer to form and administer (apply for membership, authentication) peer groups and propagate a message within the scope of a peer-group. This protocol helps in making the peers and the whole peer-to-peer infrastructure more manageable. Peers can be organized into groups to work collectively on some common interest.
5. Pipe Binding Protocol: Allows a peer to bind a pipe advertisement to a pipe endpoint, thus indicating where messages actually go over the pipe. This is independent of the transport mechanism. This makes it more manageable and easy to deploy, as the user does not have to worry about the network specific issues.
6. Endpoint Routing Protocol: Allows a peer to ask (Route Query message) a peer router for available routes for sending a message to a destination peer. The router peer responds with a Route answer message.

F. JXTA and High-Performance Distributed Computing

The main advantages of using JXTA are that it is very generic and addresses most of the issues related to the design of a peer-to-peer system. Any peer using JXTA protocols can talk to any other peer using the same protocols be it a workstation, a server, a cell-phone, a PDA or a laptop. In chapter two, we mentioned the issues related to peer-to-peer distributed computing. JXTA addresses the issues related to the design of a peer-to-peer system in the following ways.

1. Decentralization: Decentralization is one of the most important features of peer-to-peer networks. JXTA uses several schemes to achieve decentralization. The

first is to use broadcasting as a means to know about all the peers in the local network. JXTA also uses the concept of rendezvous and router peers. A rendezvous peer is a regular peer with an additional feature that it caches the services advertised by other peers. A router peer stores the route information between the peers. Even in the worst case, JXTA allows every peer to be configured to act as a rendezvous and the relay peer, which eliminates the dependency of peers on other peers for some specific functionality and single points of failure. This increases the manageability and ease of deployment in accordance with our objectives.

2. Scalability: JXTA has good scalability, which can be concluded from the fact that there are no central servers to provide specific functionalities, and each JXTA peer can handle the essential role of discovering peers and resources if a need may arise. This fulfills our first objective (scalability).

3. Resource discovery: JXTA provides the basic infrastructure so that peers can discover resources and services dynamically. This is very different from the client/server mechanism where the client has to have the prior information about server's presence. The peers in a JXTA network can discover other peers on the local area network using broadcasting. There are rendezvous peers who cache the resources advertised by other peers and router peers who cache the route to other peers.

4. Heterogeneity: This is achieved because JXTA protocols can be implemented in any language that a particular peer supports. Moreover, JXTA protocols are network, hardware, operating system and programming language independent. Thus, the user need not worry about the ease of migration and other network and operating system specific issues and results in an increase in manageability.

5. Anonymity: JXTA takes care of peer's anonymity because a unique string identifier identifies the peers and network endpoints. The real world peer identity cannot be resolved from this identifier.

6. Cost of ownership: The cost of ownership is definitely low with JXTA as it eliminates any need for a specialized machine to act as a server. Also the load on

a peer is less as there are other peers providing the same service, which decreases the bandwidth requirement at the server end.

7. Dynamism: Dynamism is achieved in JXTA by the means of advertisements. A peer advertises about its resources to other peers. Every advertisement has an expiration time. A peer continuously sends new advertisements to override the expired advertisements. The peers that receives an advertisement first checks for the advertisement expiry. If the advertisement has expired, the peer drops the advertisement. This expiration time can be set depending upon frequency with which peers join/leave the network. If peers leave and join the network frequently, the expiration period can be short.

8. Performance: The JXTA performance depends upon the advertisement overhead and network latency. If all the peers use broadcasting as a means to discover each other, there will be too much network latency because of increased traffic. Choosing an efficient policy will lead to good performance.

9. Security: Though peer-to-peer networks are less secure, using JXTA protocols, a peer can have strict group membership rules so that only the trusted peers can join a group. JXTA also provides secure pipes as a secure means to communicate with other peers. JXTA supports Transport Layer Security (TLS) that is capable of providing reliable private connections between peers [28]. JXTA also provides password based logins.

10. Peer management: A unique identifier string identifies each peer in a JXTA network. The peers can easily join or leave a group. If a peer starts a group, it can set group membership rules, which have to be followed by any peer interested in joining the group. If a peer doesn't join a particular group, it joins a default group.

11. Fault-resilience: If there are very few rendezvous and the relay peers in a JXTA network, other peers can be drastically affected by their failure. But in the worst case, all the peers can be made a rendezvous peer but this might increase the network latency. The system can be made more resilient through resource and service replication.

12. Interoperability: JXTA defines only a set of protocols. These protocols and be implemented in any programming language. As long as a peer or a device talks in this protocol (no matter what programming language the protocols are written in), that device can talk to the other JXTA peers.

13. Self-organization: In a JXTA network, peers can easily organize themselves into groups. Also if there are network failures, the router peers can learn about new routes to reach a peer.

14. Firewall friendliness: It is possible for the peers who are behind the firewall and NAT to connect to peers outside the internal network using the relay peers. The relay peers collect the advertisements for the peers behind the firewall and make this information available to the other peers. The peers that are behind the firewall initiate the communication and collect messages for them by contacting the relay peers. This satisfies out first requirement (easy deployment).

G. Why Euler and Peer-to-Peer Using JXTA

As mentioned in our objective, we want to investigate the feasibility of porting existing large-scale numerical and mathematical software to a peer-to-peer infrastructure. We chose an open-source numerical computation software called Euler [5] because of the availability of source code and its cost-effectiveness.

The reason we choose Euler is that Euler is a true representation of a typical numerical computation application and it provides some of the features (it can handle real, complex and interval numbers, vectors and matrices as mentioned earlier) of a large-scale computational application. According to [29], the main characteristics of the applications that can be deployed over a P2P implementation are where:

1. Centralization is not possible or desired
2. Massive scalability is desired
3. Relationships are transient or ad-hoc
4. Resources are highly distributed
5. Resilience is desired

For parallelizing numerical computation, we want the system to be decentralized (to avoid single point of failure), scalable (as specified in our objectives), dynamic (the joining and leaving of peers should not deteriorate the performance), fault tolerant with a short response time. Since the above-mentioned factors meet our requirements and also the numerous benefits of peer-to-peer system, we chose to use JXTA for our work. JXTA provides us with a robust framework that meets most of our requirements (scalability, ease of deployment, easy migration, manageability).

CHAPTER V

PEER-TO-PEER EULER (P2PEULER) IMPLEMENTATION

In the previous chapter, we discussed the advantages of using JXTA as the peer-to-peer infrastructure and about its suitability for parallelizing numerical computation. In this chapter, we explain our implementation of the parallelizing process. We name this implementation the P2PEuler system

A. Peer-to-Peer Euler Architecture

The P2PEuler consists of the JXTA peers and the services they provide. To provide different numerical computation services, we split up Euler into its user interface (UI) and the computation engine (CE).

Our P2PEuler architecture consists of two kinds of peers as shown in Figure 4: P2PEuler service consumers (PSC) and P2PEuler service providers (PSP).

P2PEuler service consumer (PSC): The PSC is a peer that has the Euler user interface (UI) and the Euler computational Engine (CE). Through this kind of peer, a user can submit jobs to other JXTA peers or compute the task locally (for trivial tasks).

P2PEuler service provider (PSP): The PSP is a peer that accepts tasks from PSPs and either directly executes them or sends it to other PSPs and sends the results back to the origination peer. It has got only the Euler CE.

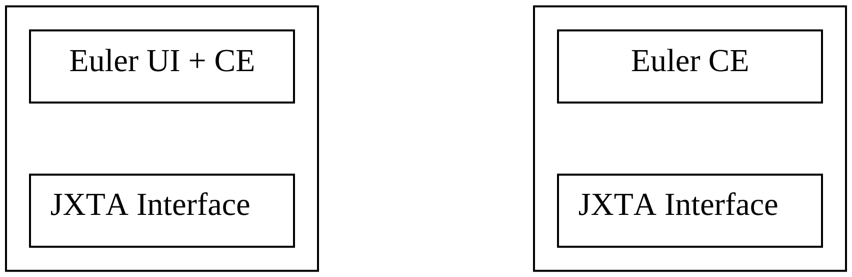

Fig. 4. Peer-to-Peer Euler Service Consumer and Peer-to-Peer Euler Service Provider

In the simplest terms, a PSPs boot up and waits for the tasks to arrive. A user on a PSC submits a task to a PSP. The PSP either executes the task or partitions it amongst other PSPs. It then combines the sub-results and sends the final result back to the originating PSC.

In both types of peers, the communication between the Euler component (UI or CE) and the JXTA Interface component is done via Java Native Interface (JNI). A machine can have both the PSC and the PSP running. It can even run multiple instances of PSCs or PSPs running at the same time.

B. Terminology

Task

A task is a complete unit of work but not necessarily atomic. Throughout this thesis, we use the term "task" and "job" interchangeably. The following are the characteristics of a task:

1. They can be decomposed into small units (subtasks).
2. They have no priority.
3. They have no deadline associated with them.

Subtask

When a PSP partitions a task into smaller units to be distributed amongst the available peers, each of those units is called a subtask. The task from which the subtasks were made is the parent and the subtasks are the children. They have all the characteristics of a task. A subtask is complete when PSPs CE has finished computing the subtask. After a subtask is computed, it produces a sub-result.

Task/Subtask Execution

Execution is the state when a PSP is working on a task/subtask. It has 3 states:

1. Computation: In this state, the CE computes the task/subtask and a result/sub-result is produced.
2. Partition: In this state, the PSP divides a task into subtasks.
3. Merger: In this state, the PSP combines the sub-results into a result.

Result

The final output of an executed task is called the result. This is the value returned by a PSP to a PSC. When a PSPs CE combines all the sub-results of the subtasks of a task, it produces a result of that task.

Task submission

A task submission happens when a user submits a task through the PSCs User Interface (UI).

Task completion

We define task completion as the event when a PSPs CE combines all the sub-results of a task successfully.

P2PEuler Service

We define P2PEuler service as the ability of a PSP to accept Euler tasks or subtasks, work on it (task execution, task assignment or sub-results merging), send subtasks to other peers and return the final results. Each PSP advertises this service to other PSP/PSc when it boots up.

Communication cost

The sum of the time required by a peer to send the data to another and the time required by the results to come back from that peer.

C. Decision Making Algorithms

The following are the decision-making algorithms in P2PEuler:

1. How a PSC chooses a PSP to dispatch a task to from all the available PSPs?
2. How do we achieve load balancing and decide if a task has to be executed locally or remotely?
3. How is task partitioned and scheduled and into how many parts?
4. Where is the task partitioning done?
5. How are sub-results combined?
6. How is peer’s metrics calculated?

D. Algorithm Details

Algorithm 1: How a PSC chooses a PSP to dispatch a task to from all the available PSPs?

Purpose: The main purpose of this algorithm is to choose the best PSP to dispatch a task to. This is done to increase the efficiency of the system (Requirement #3). The first criteria to decide a suitable PSP is the cost of communication and the second criteria is the performance of the PSP. The cost of communication is decided as the first criteria because in JXTA, the cost of communication is high and so we would like to minimize this cost.

This algorithm is run when a PSC has a list of available PSPs and wants to choose one of them to submit the job to. If there are PSPs available, it picks up a single PSP who is sent the whole task. This PSP could be either running locally or remotely. But to a PSC, they are all the same.

Time taken to open an input pipe strictly depends upon the network distance [30]. If there is a local PSP, there are very good chances that it will know about the good performance peer (since this and local PSP are on the same subnet and might be using the same rendezvous peer). If the good performance peer is available, there are good chances that the local PSP will dispatch the job to him too. Because of this reason, we first decide to send the task to the local PSP so that the cost to open input pipe is minimized.

The algorithm calculates the cost to reach each PSP from this PSC. The cost is calculated as shown in Figure 5:

```
for all available PSP
{
   if (local_IP ==  PSP_IP)
   {
        cost=0
        local_found = true
        return;
    }
    else {
        cost =1
        local_found = false;
    }
}
if(local_found)
  send_job_to_local_PSP()
else
  send_job_to_highest_metrics_PSP()
```

Fig. 5. Algorithm Used by PSP to Calculate the Cost to a PSC

The algorithm works as follows:

If a local PSP is found, the whole task is sent to him. If no local PSP is found, we use a greedy algorithm in which we choose a PSP with the highest metrics and send the task to him. In the case where more than one local PSP are available, it picks up the one with the highest metrics.

We give first preference to the local PSP because we assume that the cost to reach a local PSP will be considerably less than the cost of reaching any other remote PSP. It is also assumed that all the available peers will have roughly the same number of peers. The downside of this algorithm is that if a remote PSP is really superior and knows a lot of other superior peers, the efficiency of our peer-to-peer framework will suffer as we lose a chance to minimize the execution time of a task.

Advantages:

1. This scheme gives highest priority to a locally available PSP to minimize the communication cost.
2. This scheme picks the best peer (the one with the highest metrics) to ensure that the task execution time is the least, giving us higher efficiency (requirement #3).

Disadvantages:

1. This comes at the cost of anonymity of peers because we know the IP address of every peer. We assume that anonymity is not a major concern for the PSPs.
2. In case the PSP to which we have assigned the task fails, this PSC will be waiting forever because the user interface (UI) will wait till a result is returned by PSP and the user cannot use the interactivity of the UI.

Algorithm 2: How do we achieve load balancing and decide if a task has to be executed locally or remotely?

Purpose: According to the requirement #3, we want to make efficient use of all the resources so that some peer's resources are not wasted (idling). In an unbalanced system, the peers with high load might become unavailable either due to insufficient processing power or bandwidth (in the scenario where there is fine granularity). Such a system may become unstable and fail. A balanced system will be more scalable (requirement #1) because the availability of peers will increase. The algorithm used for load balancing should itself be scalable. If there are n available peers (processing elements), then the algorithm's complexity should be O(n).

Our load-balancing algorithm is run when a PSP receives a task and wants to achieve load balancing. Every time a peer boots up or communicates with another peer, it sends its queue size, which gets updated at recipient's end. This information is used by PSPs for load balancing. One of the goals of the load balancing policy is to minimize the time taken to complete the execution of all the tasks globally. Our load balancing policy is dynamic and decentralized. At the time of arrival of a task, each PSP decides whom to assign the task depending upon the information available with it.

The 3 basic policies of our load-balancing algorithm (as suggested in [31]) are:

Information policy: This policy specifies what constitutes the load information.

According to [32], in a general purpose distributed system, the best single workload descriptor is the number of tasks in the run queue of the operation system scheduler and that major improvements can hardly be expected when more complex (more than one workload descriptions) workload descriptions are used.

In our case, we assume that if the major part of a peer's load is due to that peer being a PSP then the PSP run queue will be proportional to the OS run queue.

Transfer policy: This policy determines the condition under which the job transfer should be made.

Ideally, this policy should consider the current load of the system and the task size. In keeping with the above information policy, we do not consider the task size in deciding whether to execute the task locally or remotely. To decide this, the PSP first checks its job queue. If the job queue is less than or equal to a threshold, we execute the task locally otherwise we execute it remotely.

Placement policy: This policy specifies the peer(s) to which the task should be transferred.

Using the greedy algorithm, we choose the best peers among the available peers and assign the task to them. We assume that the communication cost is independent of the task size and is constant.

We chose a dynamic scheme because of its flexibility, adaptability and high scalability (requirement #3). Also, in a peer-to-peer system, we cannot depend upon some centralized point as a source of information and hence a dynamic is more suitable.

Algorithm 3: How is task partitioned and scheduled and into how many parts?

Purpose: The peers in a peer-to-peer network can have diverse capabilities ranging from cell-phones to supercomputers. According to requirement #3, for the efficient use of the peer-to-peer resources, each peer should be assigned an optimal load specific to its capabilities. No peer should be overloaded as it will make that peer unavailable for other purposes and affect the efficiency and scalability of the peer-to-peer system. This algorithm helps us achieve this goal by assigning optimal load to a peer.

This algorithm is run when a PSP, after deciding to run the job remotely, wants to partition and schedule task execution among the available PSPs. When deciding a partitioning and scheduling scheme, we have to take into consideration the varying capabilities (metrics) of peers. The peer distributing tasks to other peers should be able to exploit this heterogeneity to make effective use of all the resources. This plays as key role in determining the efficiency of a peer-to-peer network. Also, the overhead of splitting a task into more number of subtasks than desired (and other overheads like communication cost) may overweigh the gain achieved by including more peers for that task computation [31] and this can deteriorate the performance of the system.

We assume that all the communication times are constant (except the one in which PSC communicated with a local PSP) and are independent of the task size and peer location. Each peer is assigned a task proportional to the metrics it advertises. For example, in a matrix multiplication task of two `nxn` matrices, the first matrix was divided in proportion to the metrics of the available peers. So for example, if there are 3 peers available and their peer metrics are `m1`, `m2`, `m3`, then the sum of their metrics is

`M = (m1+m2+m3)`

So the three subtasks in the matrix multiplication example will each have

`m1xn/M, m2xn/M, m3xn/M` rows of the first matrix and the whole of second matrix.

This is a very simple minded algorithm to achieve fair task partitioning and we believe that this algorithm will make efficient use of the available peers, if not the best. Since we have assumed earlier that the tasks are independent, the tasks are dispatched to the available PSPs in the order in which they occur in the available peer vector of a PSP. This approach is reasonable considering that each subtask is dispatched in a new thread. Also, for each task type, there is an upper bound on the number of subtasks in which it can be divided (depending upon its size) so that their distribution can be justified.

Algorithm 4: Where is the task partitioning done?

Purpose: The main idea behind this algorithm is to make the infrastructure more efficient, easy to manage and reusable.

Only the ECE of PSP has the ability to find what kind of computation has to be performed on the task. And the JTXA interface has the information about the peer's metrics. The task partitioning can be done at two places:

1. JXTA interface: This component has both the task and the information about the available peer's metrics but does not have the parsing capability to divide the tasks into subtasks. If the task were to be partitioned here, the whole parsing algorithm (which is already present in the ECE) has to be duplicated.
2. ECE: The ECE has the parsing capability but no information about the peer's metrics. But according to [12], a considerable amount of time is spent in parsing the data. If the task is partitioned here, only the peer metrics information has to be passed.

In accordance with the requirement #1 and #3 (efficiency, manageable, easy to migrate), we perform the task partitioning at the ECE. Parsing at ECE (which is written in C) will be more efficient then parsing at JXTA interface (written in Java). The JXTA interface passes the peer metrics information to the ECE, which then divides a job roughly in proportion to the available peers metrics. This also frees the JXTA interface from knowing about the Euler language and makes the code more reusable.

Algorithm 5: How and where are the sub-results combined?

Purpose: This algorithm is run when all the sub-results of a task have arrived at a PSP and the sub-results need to be combined to obtain the final result.

When a task is partitioned by the ECE and returned to the JXTA interface, additional information is passed about how to combine the sub-results. This information is a list of function calls (in the ECE) to perform the sub-results merging. This information is kept with the JXTA interface. When all the sub-results have arrived, the sub-results along with the list of function calls are passed back to the ECE. The ECE performs the merger and sends the results back to the JXTA interface that sends it back to the originating PSC.

The advantage of this approach is that the JXTA interface only does what its functionality is (to perform task management and information passing) and does not need to know about the intricacies of the data it is passing (how to perform merger). And the

ECE handles all the task manipulation functionalities. This keeps the code modular (more manageable) and provides clean interfaces.

Algorithm 6: How is peers metrics calculated?

Purpose: To make efficient use (requirement #3) of each peer's capabilities, a peer's capabilities should be known to other peers so that they can be assigned task in that proportion. One way to do this is to run a benchmark. Since the capabilities of a peer will not change over time in a single join-leave session, the benchmark is run only when a PSP joins the JXTA network. [33] uses SPECmarks benchmark to assign tasks to machines.

We use SciMark 2.0 to calculate a peer's metrics. SciMark is a benchmark tool developed by NIST and is based on 5 numerical kernels to measure the performance of numerical codes occurring in scientific and engineering applications [34]. It calculates a composite score based on 5 numerical kernels which are: FFT, Gauss-Seidel relaxation, Sparse matrix-multiply, Monte Carlo integration, and dense LU factorization. SciMark is chosen because it provides a true measure of the performance of a machine undergoing scientific and engineering computation.

E. The Peer-to-Peer Euler Process

The whole process from job submission to result display can be divided into the following steps:

1. A PSP boots up and waits for the tasks to arrive
2. The PSC boots up
3. User submits a job
4. Job is sent to the jxta network
5. Job received by PSP
6. Job execution decision: local/remote
7. Job partitioning and assignment
8. Job execution

9. Sub-results Merging
10. Result retrieval
11. Job complete

F. Peer-to-Peer Euler Process Explained

1. A PSP boots up and waits for the tasks to arrive

The first thing that a PSP does after it starts is to run the benchmark (metrics) as shown in Figure 6. The benchmark we chose was SciMark 2.0. It joins the default peer group and initializes its discovery and pipe service. It then starts the local and remote discovery to search for advertisements. This is an asynchronous process.

After this, it prepares an advertisement (`doAdvertise`) and adds the following information to it:

- The peer's metrics
- The current queue length
- The service it provides.
- Input pipes ID

For this setup, the only service a PSP provides is the P2PEuler service. It sends the advertisement so that it is available to other peers and creates an input pipe based on the advertisement. It starts the service (`startService`) and waits for the tasks to arrive (using `waitForMessage` method). This blocks indefinitely till a message arrives.

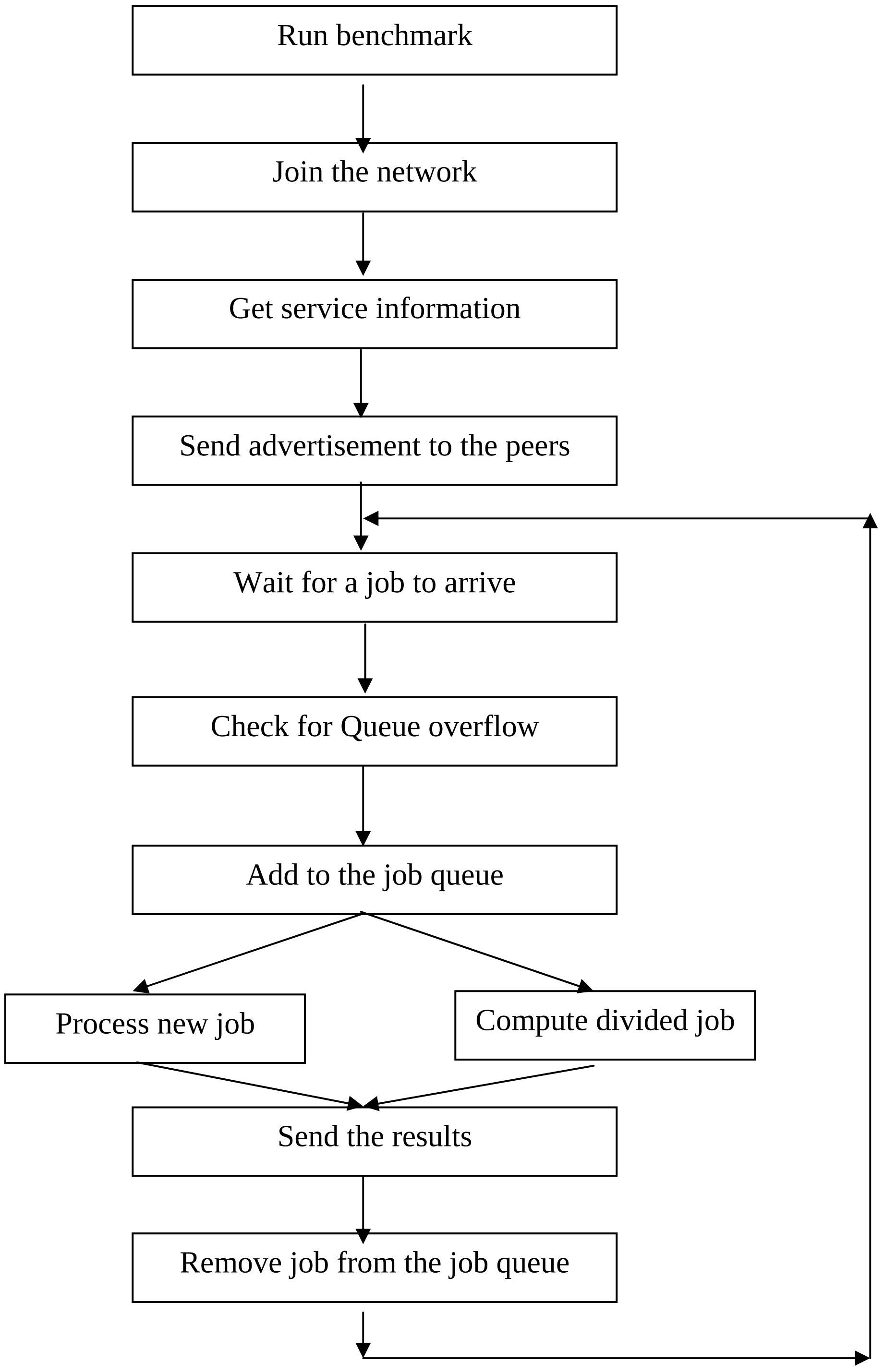

Fig. 6. Peer-to-Peer Euler Service Provider

2. The PSC boots up

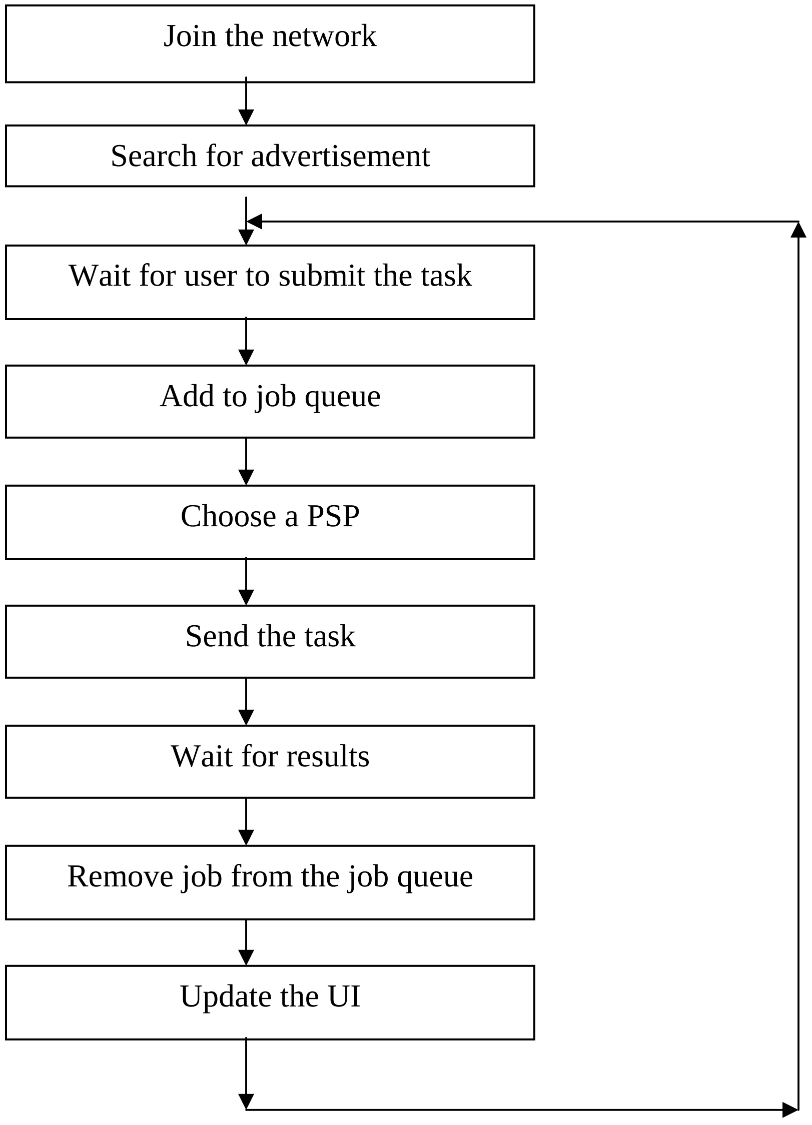

Fig. 7. Peer-to-Peer Euler Service Consumer

The user starts a PSC process as shown in Figure 7. This is similar to how he would start the Euler process but this time he runs a script:

```
./p2peulersc
```

This causes the PSC to run the JXTA initialization functions (`jxta_init`). This function creates a EulerClient and a WorkManager object and initializes it (by calling its constructor). The EulerClient object is an interface between the Euler C code and the Java WorkManager. The WorkManager constructor calls the function (`startJxta`) to join the worldpeergroup (which is a default peer group joined by any peer which boots up in the JXTA network) and initializes the discovery and the pipe service for this PSC. The constructor then starts the local and remote advertisement discovery process (to search for services/resources advertised by other peers) that is asynchronous. It then gets a handle on the other WorkManager object fields and methods necessary to submit and obtain the results and returns the control to PSC User Interface (UI). The P2PEuler UI shows up along with the graphics window. Now the user is ready to submit the job.

3. User submits a job

He types in his task. At this point, a user has two options:

1. He can compute the result locally, or
2. He can submit the task to be computed by other JXTA service providers (JSP).

This type of approach has the advantage that the user had control over the task submission, as he is the one who knows best what kind of tasks he is submitting. If the task is really simple (like adding some quantities), he has the option to do the computation locally. If the task requires lots of computation, he can submit the task so that it is computed remotely. This also keeps the Euler session more interactive. This also allows the PSC to compute a task locally in case there are no available peers.

If the user wants to submit a job to be computed remotely, he precedes the statement with

`submit_to_p2pEuler`:

Consider an example when a user wanted to multiply two matrices, he types the following to submit the job to remote JSP:

```
a=[...]
b=[...]
submit_to_p2pEuler:a.b
```

We call this event “Task Submission”. This is a blocking call that waits till the user gets the result back from the jxta network.. This causes the p2peuler to invoke the `submitToWorkManager` function of the EulerClient object. The control then transfers to the WorkManager's `processJob`.

4. Job is sent to the jxta network

The WorkManager's `processJob` function accesses the list of available peers (through the asynchronous advertisement discovery process discussed above). If no PSP is available, the function returns with an error and the control transfer back to the Euler part that then processes the job locally using its own CE.

If there are PSPs available, it picks up a single PSP who is sent the whole task. It uses algorithm 1 to select a PSP. After a PSP is selected, the PSC adds the following information to the task (`Message`) being sent to the PSP:

1.	Input pipes ID:	This information is for the PSP to send the final result back.
2.	Task ID:	A string identifying this job (a random number)
3.	Task Status:	Job is new (`JOB_NEW`)
4.	Task String:	The whole task

It then sends the message to the PSP. It is the responsibility of the PSP (to whom the job was sent) to return the final result back to the requestor.

5. Job received by PSP

The PSP is waiting for a task to arrive (`waitForMessage`). This process blocks till a message arrives at the input pipe. When a task arrives at a PSP (in the form of a JXTA `Message` object), it extracts the `Message` from it. It checks if its job queue is full. If the job queue is not full, the task is added to the job queue otherwise a reply is sent to the PSC indicating that this job was rejected because the job queue is full. The JSC makes a note of the full job queue. After this, the PSP checks the job status. If the job status is new (`JOB_NEW`), the function `processNewJob` is called otherwise if the job status is a divided job (`JOB_DIVIDED`), it knows that it has to compute this job and so calls the `computeDividedJob` function.

6. Job execution decision: local/remote

When a new task arrives, the PSP makes a decision to either execute the task locally or remotely according to the algorithm 2. If executing locally, it computes the task. If running remotely, it partitions the task.

7. Job partitioning and scheduling

After the PSP decided to run the task remotely, it partitions the task into subtasks and dispatches each of them one after the other according to algorithm 3. A new thread is created to dispatch each subtasks.

8. Job execution

If the task (or subtask) that arrived is the one this PSP has to compute, it passes the task to the CE. A task is the one sent by a PSC to execute and this PSP decides to compute locally. A subtask is the one sent by another PSP to this peer to compute. The CE computes the task (or the subtask) and outputs the result (or sub-result). The result is sent to the requesting PSC and the sub-result is sent to requesting PSP.

9. Sub-results Merging

After all the sub-results of the particular task have arrived, the sub-results are combined using algorithm #5. The task is complete now and the results are sent back to the originating PSC.

10. Result retrieval

The PSC is listening (`pipeMsgEvent`) through the input pipe for the results. When a message arrives at this pipe, the results are extracted from it and passed on to the UI.

11. Job complete

The UI displays the result to the user and the job is complete now.

CHAPTER VI

EXPERIMENTAL SETUP

A. Experimental Setup

Experiments were conducted to test the P2Peuler application on a LAN. The task was to calculate the prime numbers from 1 to 1000000 with the lower boundary of 1. The prime numbers were calculated with the following upper boundary: 10, 100, 1000, 10000, 100000, 200000, 600000, 800000, 1000000. The number of peers in the experiments varied from 1 to 6 for each set. Figure 8 shows the experimental setup of peers.

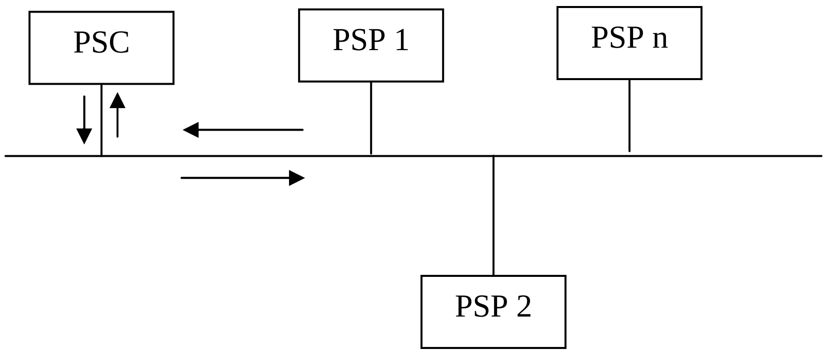

Fig. 8. Experimental Setup

In the experimental setup, there was one PSC and more than one PSPs. PSC chooses one PSP, submits the task to it and waits for the result. The chosen PSP takes care of task partitioning and gives the final result back to the PSC. Table 1 shows our experimental results.

Table I. Time Taken (in Milliseconds) by Peers to Compute Prime Numbers

Prime number Upper boundary	Number of Peers				
	1	2	3	4	6
10	2459.8000	4315.5000	5120.2500	7531.0000	9795.1667
100	1822.2000	3556.7500	4734.2500	5808.5000	7913.7500
1000	1922.8000	3323.5000	4974.7500	5846.7500	7644.2500
10000	2090.8000	3264.2500	4454.5000	6002.4000	7828.5000
100000	7261.1667	4712.2500	4963.7500	6263.5000	8010.5000
200000	19271.6667	8468.5000	7443.7500	7209.7500	8859.7500
400000	61396.5000	16722.2500	13900.6000	12586.5000	14068.4000
600000	149193.2000	29109.7500	22579.7500	19887.5000	16661.7500
800000	288622.6000	48426.5000	34196.5000	29519.8000	20163.2500
1000000	457166.4000	74676.0000	48867.0000	43745.0000	26500.0000

The experiment was repeated to compute the prime numbers locally.

B. Experiments with Computing Prime Numbers

The following graphs summarize the results.

Figure 9 shows a graph between the time taken to complete the task and the number of peers available for prime numbers calculated from 1 to 1000000.

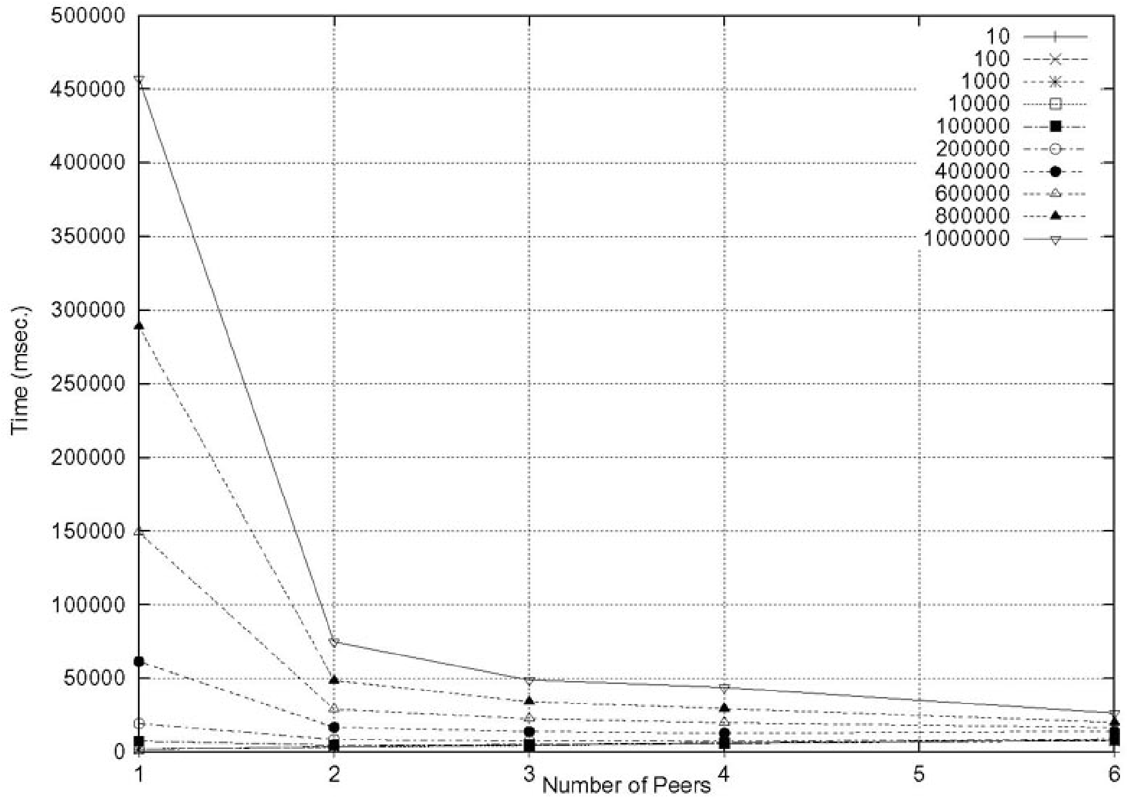

Fig. 9. Time Taken (in Milliseconds) vs. Number of Peers (for Prime Numbers up to 1000000)

We can conclude from the graph that the benefits for this application are more visible for bigger task sizes. This behavior is to be expected as the ratio to computation time and communication time is higher for the bigger tasks.

Figure 10 shows the time taken by peers for prime numbers less than 200000. The purpose of this figure is to show the behavior not visible in the graph above. In this graph, we see that for smaller tasks (<100000), the completion time increases as the number of peers increase. This behavior can be attributed to the fact that for smaller tasks, the communication overhead is more than the computation time and it would be beneficial not to split the tasks into parts.

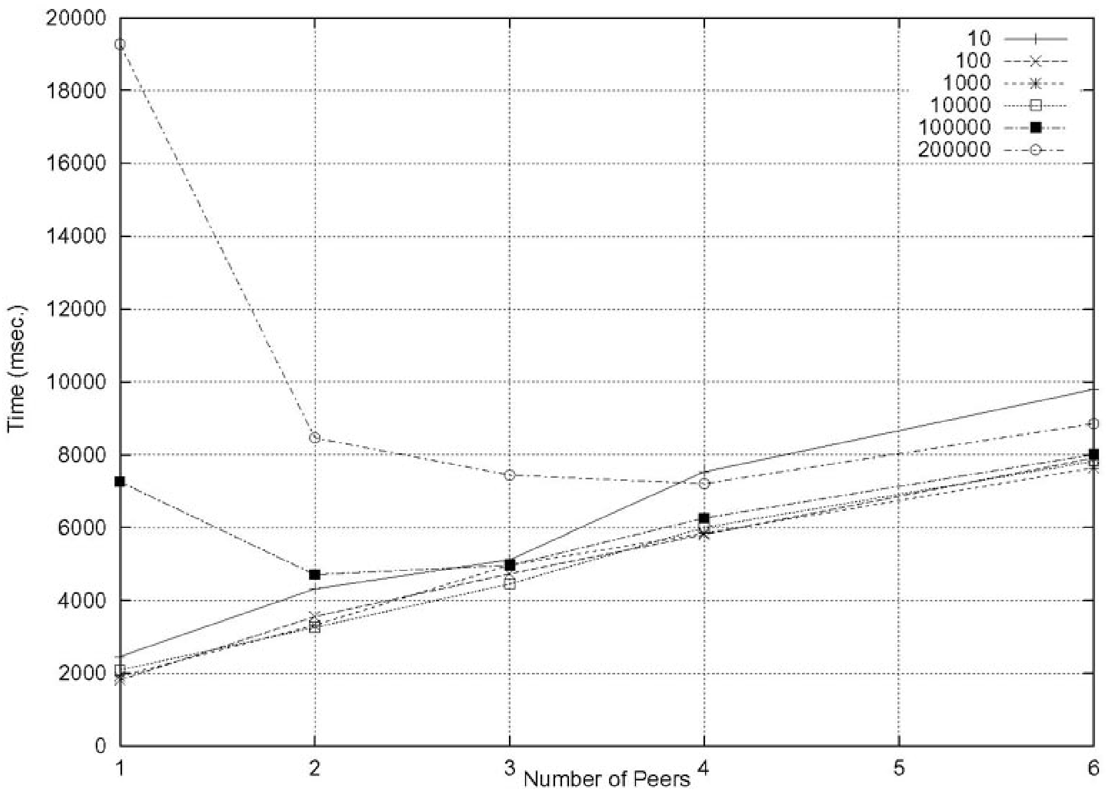

Fig. 10. Time Taken (in Milliseconds) vs. Number of Peers (for Prime Numbers < 200000)

Speedup: Speedup was measured as the ratio of time taken to compute the task locally and remotely.

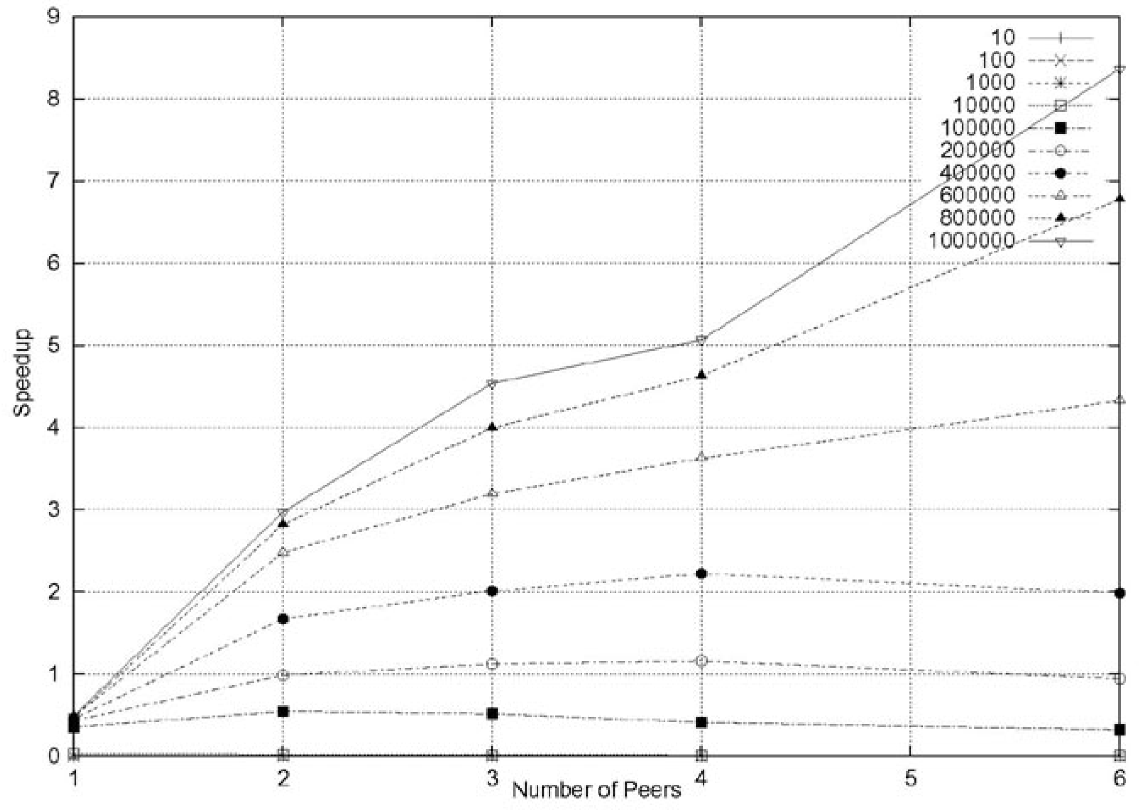

Fig. 11. Speedup vs. Number of Peers

It is evident from Figure 11 that for smaller tasks (<100000), the speedup is not significant. But as the task size increases, the speedup increases.

C. Experiments with Different Task Partitioning Algorithms

Experiments were conducted to calculate the prime numbers between 1 and 1000000 using three different algorithms to partition a task submitted by a PSC among the available peers. The algorithms used are as follows:

1. Random task partitioning: The tasks were randomly partitioned and assigned randomly among the available peers.
2. Equal task partitioning: The tasks were partitioned equally among all the available peers.
3. Proportional task partitioning: The tasks were partitioned using algorithm three described earlier.

The results of this experiment are summarized in the Table II.

Table II. Experiments with Three Different Task Partitioning Algorithms

Task Partitioning Algorithm	Average Time taken (in millisecond)
Random Task Partitioning	180665
Equal Task Partitioning	131369
Proportional Task Partitioning	112795

The results of this experiments show that the proportional task partitioning algorithm gives us the best results. This is to be expected because in the proportional task partitioning scheme, we assign tasks proportional to a peer's capability which minimizes the task completion time makes the most efficient use of a peers resources among the three algorithms tested.

CHAPTER VII

CONCLUSION

In accordance with our objective, we have successfully developed a numerical computation software (P2PEuler) that uses a peer-to-peer framework to distribute its computation. The application is at the development stages and runs on Linux and is compatible with Solaris. We have met the objectives set out for this thesis in the following ways:

1. The experiments conducted show that the numerical computational software developed is scalable to a modest degree and is easy to manage and deploy.
2. The developed software is compatible for Matlab-style computation and is very cost effective.
3. The results of our experiments showed that the developed software makes efficient use of the peer resources (using an efficient task partitioning scheme) and gives us acceptable performance improvements in terms of speedups.

As of this writing, the P2PEuler system is still under development. The system can be improved in various ways. First, the algorithms used to load balance, partition the task and schedule it can be improved to add complex features. For example, in order to be able to use this infrastructure for complex problems, the task-partitioning algorithm can analyze the dependencies. A better estimate of communication cost to a peer can be made by pinging and peers with lower communication cost will be given preference (in case their metrics are same). Second, when partitioning the task, script level parallelism can be implemented to achieve outer-loop coarse grain parallelism. Third, when deciding to execute a task locally or remotely, the task size can be taken into account. Four, peers can advertise multiple metrics. Some tasks might be IO bound where as others might be CPU bound. For each type of the task, the metrics best suited for that particular task can be used.

REFERENCES

[1] Institute of Electrical and Electronics Engineers, *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*, IEEE, New York: 1990.

[2] The MathWorks, Inc., “MATLAB 6.5.1 product description,” 2004, http://www.mathworks.com/products/matlab/description1.html (accessed April 23, 2004).

[3] The MathWorks Inc., “MATLAB acceleration, innovation and development,” 2003, https://tagteamdbserver.mathworks.com/ttserverroot/Download/10300_9215v02_MATLAB_Brch.pdf (accessed April 25, 2004).

[4] Iowegian International Corporation, “Matlab clones,” http://www.dspguru.com/sw/opendsp/mathclo2.htm (accessed April 23, 2004).

[5] R. Grothmann, “The EULER software,” http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/euler/euler.html (accessed April 23 2004).

[6] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja1, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, *Peer-to-Peer Computing*, Technical Report HPL-2002-57, HP Lab, 2002, http://citeseer.ist.psu.edu/milojicic02peertopeer.html (accessed April 23, 2004).

[7] L. Gong, *Project JXTA: A Technology Overview*. Sun Microsystems, Inc., April 2001, http://www.jxta.org/project/www/docs/TechOverview.pdf (accessed April 25, 2004).

[8] A. Lumsdaine, “CAREER: High-performance computing for computational science and engineering,” October 17, 1997, http://www.siggraph.org/education/nsfcscr/projects/gen/lumsdaine.html

(accessed April 23, 2004).

[9] J. F. Mehaut and Y. Robert, "Algorithms and tools for (distributed) heterogeneous computing: A Prospective Report," 1999, http://citeseer.ist.psu.edu/mehaut99algorithms.html (accessed April 23, 2004).

[10] G. Coulouris, J. Dollimore and T. Kindberg, *Distributed Systems Concepts and Design.* 3rd edition, Addison-Wesley, Upper Saddle River, NJ: 2001.

[11] J. F. Baldomero, "Message passing under Matlab,". http://atc.ugr.es/~javier/investigacion/papers/ H010FernandezBaldomero002.pdf (accessed April 23, 2004).

[12] C. Moler, "Why there isn't a parallel Matlab," MathWorks Newsletter, Spring, 1995, http://www.mathworks.com/company/newsletter/pdf/spr95cleve.pdf (accessed April 23, 2004).

[13] S. Pawletta, W. Drewelow, P. Duenow, T. Pawletta, and M. Suesse, "A Matlab toolbox for distributed and parallel processing," in *Proceedings of the MATLAB Conference 95*, Cambridge, MA: 1995, http://citeseer.nj.nec.com/pawletta95matlab.html (accessed April 23, 2004).

[14] G. Almasi, C. Cascaval, and D. A. Padua, "MATmarks: A shared memory environment for Matlab programming," *HPDC 1999*, http://citeseer.ist.psu.edu/255289.html (accessed April 23, 2004).

[15] H. Lu, S. Dwarkadas, A.L. Cox, and W. Zwaenepoel, "Message passing versus distributed shared memory on networks of workstations," in *Proc. of Supercomputing '95,* San Diego, CA: December 1995.

[16] E. Heiberg, "Matlab parallelization toolkit 1.20," Parallel Matlab Interface, November 2003, http://hem.passagen.se/einar_heiberg/index.html (accessed February, 2004).

[17] R. Choy, "Parallel Matlab survey," http://supertech.lcs.mit.edu/~cly/survey.html (accessed April 23, 2004).

[18] M. D. DeVores, “DistributePP - A distributed parallel processing tool for Matlab,” http://www.people.virginia.edu/~md9c/DistributePP/ (accessed April 23, 2004).

[19] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubil, S. Steer, F. Suter, and G. Utard, “Scilab to Scilab - the Ourangan project,” *Parallel Computing,* vol. 11, no. 27, pp. 1497-1519, 2001, http://citeseer.ist.psu.edu/caron02scilab.html (accessed April 23, 2004).

[20] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund,” Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,” *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107-31,1999.

[21] V. Nair, “The design, implementation and evaluation of cryptographic distributed applications: Secure PVM,” Technical Report, University of Tennessee, 1996, http://citeseer.ist.psu.edu/55679.html (accessed April 25, 2004).

[22] G. Stellner, “Consistent checkpoints of PVM applications," in *Proceedings of the First European PVM User's Group Meeting*, Rome, 1994, http://citeseer.ist.psu.edu/stellner94consistent.html (accessed April 25, 2004).

[23] G. Bosilca, A. Bouteiller, F. Cappello, S. Djailali, G. Fedak, C. Germain, T Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov, "MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes," 2002, http://citeseer.ist.psu.edu/bosilca02mpichv.html (accessed April 25, 2004).

[24] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov, “Framework for Peer-to-Peer distributed computing in a heterogeneous, decentralized environment,” *Grid Computing - GRID 2002: Third International Workshop*, pp. 1-12, July 2002.

[25] JXTA, “The JXTA project homepage,” http://www.jxta.org/ (accessed April 25, 2004).

[26] J. Gradecki, *Mastering JXTA: Building Java Peer-to-Peer Applications*. John Wiley & Sons, Inc., New York: 2002.

[27] B. J. Wilson, *JXTA*. New Riders, Berkeley, CA: 2002.

[28] Sun Microsystems, “Security and project JXTA, “ Sun Microsystems, Inc., http://www.jxta.org/project/www/docs/SecurityJXTA.PDF (accessed April 25, 2004).

[29] J. C. Soto, “Introduction to project JXTA,” October 2003, www.internet2.edu/presentations/fall-03/20031014-P2P-Soto.pdf, (accessed April 23, 2004).

[30] E. Halepovic and R. Deters, “The costs of using JXTA,” *Third International Conference on Peer-to-Peer Computing (P2P'03)*, September 01 - 03, 2003, Linköping, Sweden, http://bosna.usask.ca/pub/P2P03_Halepovic_CostsOfUsingJXTA.pdf (accessed January 17, 2004).

[31] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Society Press, Los Alamitos, CA:1995.

[32] T. Kunz, “The influence of different workload descriptions on a heuristic load balancing scheme,” *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 725-730, July 1991.

[33] Ö. Babaoglu, L. Alvisi, A. Amoreso, R. Davoli and L. A. Giachini, “Paralex: An environment for parallel programming in distributed systems,“ in *Proceedings of the 6th ACM International Conference on Supercomputing*, Washington, D.C., pp. 178-187, July 1992.

[34] R. Pozo, and B. Miller, “SciMark 2.0,” http://math.nist.gov/scimark2/

(accessed April 25, 2004).

VITA

Name	Rajeev Agrawal
Address	D3/3245, Vasant Kunj New Delhi 110 070, India
Education	B.E. in Mechanical Engineering, University of Pune, India, July 1998.
	M.S. in Interdisciplinary Engineering, Texas A&M University, May 2004.

The typist for this thesis was Rajeev Agrawal.