CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Studying the Behavior of Classical Parallel Algorithms with Message Passing

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

By

Ankit Pandey

MAY 2012

The Graduate Thesis of Ankit Pandey is approved:

_____

Prof. Richard Covington Date

_____

Prof. Robert McIlhenny Date

_____

Prof. Gloria Melara, Chair Date

CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

ACKNOWLEDGEMENTS

This thesis would not have been complete without the help of my committee chair Prof. Gloria Melara. She has guided me and helped me in many ways. I am also grateful to my family for always supporting me and giving me the courage to move forward in difficult times. I would also like to express my appreciation to my colleagues who have corrected a lot of my mistakes. I thank Prof. Robert McIlhenny, and Prof. Richard Covington for agreeing to be a part of my thesis committee. Last, but not the least, I thank all the people who I was associated with during my course at CSUN.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

STUDYING THE BEHAVIOR OF CLASSICAL PARALLEL ALGORITHMS WITH MESSAGE PASSING

BY

ANKIT PANDEY

Master of Science in Computer Science

In this thesis, we propose a distributed memory parallel computer-testing tool called the sandbox. The evaluation of parallel programs runs in the controlled environment with cloud computing instances, where a single instance is used to monitor and control all the instances of the sandbox. This allows the development of applications and teaching of parallel programming without the expenses of supercomputing resources. We use a standard message passing system, a reliable java network application, so that when desired the can be run on a truly parallel system without any changes.

Several features in the sandbox do not exist in other simulation system. Support for multi-tasking more processes than processors can be done, allowing the evaluation of load balancing schemes.

1. INTRODUCTION

Parallel processing is important in the entire field of computer science. [5] Earlier major processor manufacturers were increasing the clock speed mainly, to increase the computational strength of processors and to minimize their execution time. This approach leads them to end up in increasing straight-line instruction and heating up the processor. However, they are adopting the Multiprocessor systems to increase the computational strength with less execution time, less heat up, less complexity. Multiprocessor system comes in many different categories, but there are two primary categories: parallel computers and distributed computers. These two terms are used with some overlap, but usually a parallel system is one in which the processors are closely connected; while a distributed system has processors those are more independent of each other. In a distributed system, each processor has its own independent memory. This makes it difficult to use shared memory for inter processor communication, as it will increase the additional overheads and would make it too complex to implement. [6] This is why processors communicate by sending messages. In a cluster, these messages communicate via network. Though message passing is relatively slower than shared memory, it scales better for many processors. In this thesis we are developing a sandbox, which analyze algorithms in distributed environment. I have chosen distributed environment, because distributed programming is a potentially powerful tool that has only become broadly available for developers at large in the past few years. The power of distributed programming is not in the fact that a bunch of processors are scattered across the network. The power lies in that any processor in the system can directly interact with an object that "lives" on a remote host distributed programming.

1.1 Purpose of the thesis

The purpose of this thesis is to develop the Sandbox in the cloud environment to research and analyze the behaviors of classical parallel algorithms. The Sandbox will be a tool in controlled environment, so that users can get hands on experience with parallel processing. This environment will allow users to analyze message passing parallel algorithms. This thesis will provide a method to replicate the sandbox for different sets of parallel algorithms. This will allow users to do further research on parallel processing. The Sandbox will provide a Graphical user interface with cluster environment. This Graphical User Interface would provide an interactive tool, to allow users run and debug complex algorithms. Parallel processing involves some known and some hidden overheads. This sandbox would provide an opportunity to do research on the parallel processing overheads. This Sandbox would make users accurately run parallel algorithms, without using the super-computing resources.

1.2 Environment

Widows' server is used as to implement the sandbox. There are five active instances of windows 2003 on Amazon clouds with the following configuration. 20 GB Hard drive, 4GB Ram and Windows 2003 server. Net Beans IDE is a free, open-source, cross-platform integrated development environment with built-in support for the Java programming language. It is used to develop the sandbox Graphical User Interface, java network application, and algorithms. It offers many advantages over coding with a text editor.

2. Message Passing

Message passing is the paradigm of communication where messages are sent from a sender to one or more recipients. [1-4] Forms of messages include method invocation, signals, and data packets. Primarily there are two types of message passing, Asynchronous and Synchronous message passing. Synchronous message passing systems require the sender and receiver to wait for each other to transfer the message. That is, the sender will not continue until the receiver has received the message.

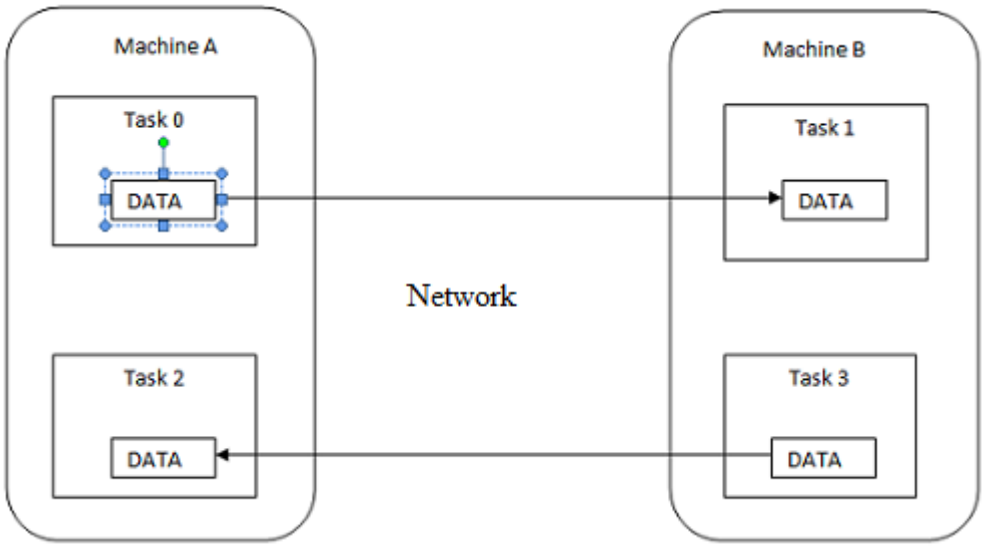


Figure 1 Message Passing Communication

Asynchronous message passing systems deliver a message from sender to receiver, without waiting for the receiver to be ready. The advantage of asynchronous communication is that the sender and receiver can overlap their computation because they do not wait for each other. In

figure – machine “A” Passes message to machine “B” and “B” to “A” here both machines A and B does not have any prior notification or message.

2.2 What is a message in message passing?

A message is a structured piece of information sent from one processor to another over a communication channel. Some messages are requests made to one processor by another, other messages deliver data or notification to another processor. In most applications, a message consists of a message identifier and, if needed, a set of message arguments. The message identifier tells the receiver the purpose or type of the message. The arguments to the message contain additional information that is interpreted and is based on the type of message. Message identifiers are usually simple, unique tokens that differentiate one type of message from another. They may even be simple integer values, where the processor on either end use a look-up table of some sort to match the value with its meaning. Message arguments, on the other hand, can be of many types. Some simple message protocols get away with using only basic data types, like integers, strings, and floating-point values, for message arguments. Other protocols need to use more complicated data types and objects as message arguments. These complex data types can be sent as an ordered sequence of simple data types.

2.2 What are the Advantages of Message passing?

Message passing have the several advantages that helps programmer to develop the distributed application, which require inter-processor communication. Message passing applications are less dependent of operating system and specific hardware. Message passing applications can be developed in one environment and can be enhanced in other environment.

2.2.1 Non-Blocking Communication

Non-Blocking communication is a type of message passing which helps in parallel processing. There are lots of overheads and communication breaks when a parallel program is intended to make inter-processor communication with number of processes back and forth. During the process sender has to wait for the reply from the receiver. Working on the parallel processing without Non-blocking communication is unreliable, because there is chain of processes one process is dependent on other process and if one fails due to some kind of communication blocking and delays the chain of process will break.

With the message-passing solution, the resource is not exposed, and an associated process makes all changes to it, so that the resource is encapsulated. Processes wishing to access the resource send a request message to the handler. If the resource or subsection is available, the handler makes the requested change as an atomic event, that is conflicting requests are not acted on until the first request has been completed. If the resource is not available, the request is generally queued. The sending program may or may not wait until the request is completed.

3. Technologies used

3.1 Why Java is used?

The parallel processing is all about connecting processors together. [18] One of the most exciting aspects of Java is that it incorporates an easy-to-use, cross-platform model for network communications that makes it possible to explore distributed programming without years of study. This opens up a completely new class of applications to programmers. Java technology's versatility, efficiency, platform portability, and security make it the ideal technology for network computing. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere, Java has been tested, refined, extended, and proven by a dedicated community. It has the ability to run in any platform and rich API for message passing programming. Some of the libraries are inherited taken from the package Parallel Java. Parallel Java is an API and middleware for parallel programming in 100% Java on shared memory multiprocessor (SMP) parallel computers, cluster parallel computers, and hybrid SMP cluster parallel computers.

3.2 Why Java swing ?

Java swing GUI design focuses on the layout of controls, the presentation of data, and the mechanics of completing various tasks with parallel processing application. [18] [19] However, even a program that has a great-looking, intuitive interface can be virtually unusable if it does not respond well to user actions. Performance is a key aspect of GUI design that is often overlooked until it is identified as a problem late in the development cycle. Using the concurrency within the client properly is the key to creating a responsive user interface with Swing. Because support for local concurrency was built into the Java programming language, maintain concurency is relatively easy; however, using them correctly can be difficult. Swing provides a timer that enables a task to be executed either periodically or at a specific time. Timers are important in parallel processing, albeit specialized, tools for the GUI programmer because they simplify the job of scheduling activity that results in a screen update.

3.3 Why Java core networking is used?

Distributing objects is a good idea, but distributed object systems like CORBA and,[18] to a lesser degree, Java RMI is so big and complicated. The core Java API, which is used in this thesis; has a package named java.net and java.io packages, this gives an easy access to the network and key network protocols. These packages enable the layer application-specific operations on top of the network conveniently. To use these java.net packages all we need to extend these packages to allow objects to invoke each other's methods over the network, and we have a basic distributed object system.

3.4 Why UDP/IP Protocol ?

UDP provides a minimal, unreliable, best-effort, message-passing transport to applications and upper-layer protocols. Compared to other transport protocols, [4] UDP messages are unique that they do not establish end-to-end connections between communicating end systems. UDP communication consequently does not incur connection establishment and teardown overheads and there is minimal associated end system state. Because of these characteristics, UDP can offer a very efficient communication transport to some applications, but has no inherent congestion control or reliability. A second unique characteristic of UDP is that it provides no inherent On many platforms, applications can send UDP datagram at the line rate of the link interface, which is often much greater than the available path capacity, and doing so would contribute to congestion along the path, applications therefore need to be designed responsibly

3.5 IP

IP, which stands for Internet Protocol, is a [4] Network layer protocol that is responsible for delivering packets to network devices. The IP protocol uses logical IP addresses to refer to individual devices rather than physical (MAC) addresses. A protocol called ARP (for Address Resolution Protocol) handles the task of converting IP addresses to MAC addresses. Because IP addresses consist of a network part and a host part, IP is a routable protocol. As a result, IP can forward a packet to another network if the host is not on the current network.

4. Challenges in the parallel programming:

Parallel processing involves countless challenges, [22] but there are classic parallel programming challenges; for example making a cross platform application that has interactive graphical user interface, debugger and a support for concurrency. There are some reputed organizations, working on parallel processing for years and has come out with some parallel processing tools, which is used to implement and observe the parallel algorithms. Solaris provides some of the most important industry standards such as OpenMP, MPI, and Grid Computing, and describe the current state of parallel application software development. OpenMP programming model is specified for single process. The scalability of OpenMp is ultimately restricted by the number of processors threads in a single machine. The scalability is limited by the programming logic complexity and programming style as explained in the above example. Current OpenMP semantics may not be flexible enough to handle some applications. For example, current OpenMP specification only allows limited dynamic thread creation in parallel region. The OpenMP language committee, whose members include Sun Microsystems, Intel and many other companies, are in the process of specifying a dynamic task queue construct to remove this critical limitation. The other more general problems when programming an OpenMP application are avoiding the race condition errors from multiple threads and achieving good load balance among the concurrent working threads.

MPI constraints in general MPI offers a good solution for parallel application on computer clustering, but it is also a difficult programming model for many developers. Because MPI has longer communication latency, the program core logic must be partitioned well to justify the distribution overhead. It is not an intuitive task to analyze and partition an application problem

and map it to a set of distributed processes. Because of the complexity of interaction among many MPI processes, it is also quite challenging to debug and tune a MPI application running on a large number of nodes even with the right tools.

Some other challenges for the parallel processing according to intel, are as follows: Finding concurrency in a program - how to help programmers think parallel?; There are very few and almost no soft-wares are available for the support of concurrency. Scalability support in hardware; bandwidth and latencies to memory and interconnects between processing elements. Scalability support in software libraries. Scalable algorithms, and adaptive runtimes to map high-level software onto platform details. Synchronization constructs and protocols that enable programmers write programs free from deadlock and race conditions. Lack of tools, API's, and methodologies to support the debugging process. Error recovery and support for fault tolerance. Support for good software engineering practices: Incremental parallelism and code reuse. Support for portable performance. What are the right models or abstractions so programmers can write code once and expect it to execute well on the important parallel platforms?

5. Methods to Implement Parallel programs

5.1 Machines with Parallel Processors

There are some traditional and new methods to run and analyze [2] the parallel programs. One of them is supercomputing machine that has number of processors. The cost of hardware for super computing is always high. It is almost unaffordable to get hundred-processor machine for the purpose of analyzing and researching the parallel processing. However, it is convenient and beneficial to get a single processor machine then a parallel processor machine in an era of cloud computing. Even if we get the machine with two or three processors, we will not be able to test the scalability of the parallel programs.

5.2 Simulation of Parallel Programs

Simulating the parallel programs could make users run the parallel [16] algorithms with rounds of inputs. Simulation software for parallel programs is specific for particular type of algorithms. If we have MPI simulator for sorting and searching then it is specific to sorting and searching only we can do Image processing or any other parallel algorithms. Other issue with the simulation software is the validation and accuracy of the results. For example : if simulation is done for an Image processing with four processor, and the size of an image is 1000x1000. Each processor gets 250x250 pixels and completes image enhancement in 0.5 sec. This analysis still lacks the overheads, as there is few or no inter-processor communication during the process. Simulation of parallel processing leads users to superlative knowledge.

5.3 Distributed Programming

This thesis provides a comprehensive method to analyze the parallel processing and the procedure to setup a distributed networking environment with socket programming. This procedure can be implemented with cloud computing or with the machines connected on LAN. Here users can develop their application for the algorithms they want to implement. This thesis provides a tool known as sandbox, which is implemented using UDP/IP protocol in the Amazon clouds for message passing algorithms. Users can use the same framework and enhance it for the type of algorithms they want.

6. Cloud based web services

Cloud computing services comes different form Saas, Iaas, Haas. These different versions are the type of services they provide. We have used Hardware as a service that is Amazon EC2 as the requirement of the project was parallel processing.

6.1 Why Amazon Elastic cloud computing?

Amazon Elastic Compute Cloud EC2 is a web service that provides resizable compute capacity in the cloud. With Amazon EC2 applications can be built, that starts small but can scale up rapidly as demand increases. Amazon EC2 offers many features that makes it possible to build such an applications, including:

- Ability to increase or decrease capacity within minutes
- Ability to commission one, hundreds, or even thousands of server instances simultaneously
- A web service API to control the scaling of instances depending on your needs
- A “pay only for what you use” pricing model

Amazon Simple Queue Service Amazon SQS is a complementary web service that enables users to build highly scalable EC2 applications easily using message passing. Amazon SQS is a highly reliable, scalable message queuing service that enables asynchronous message-based communication between distributed components of an application. The components can be computers, EC2 instances, or a combination of both. With Amazon SQS, users can send any

number of messages to an Amazon SQS queue at any time from any component. No message is ever lost its always in the SQS which can be retrieved.

7. Technical Approach

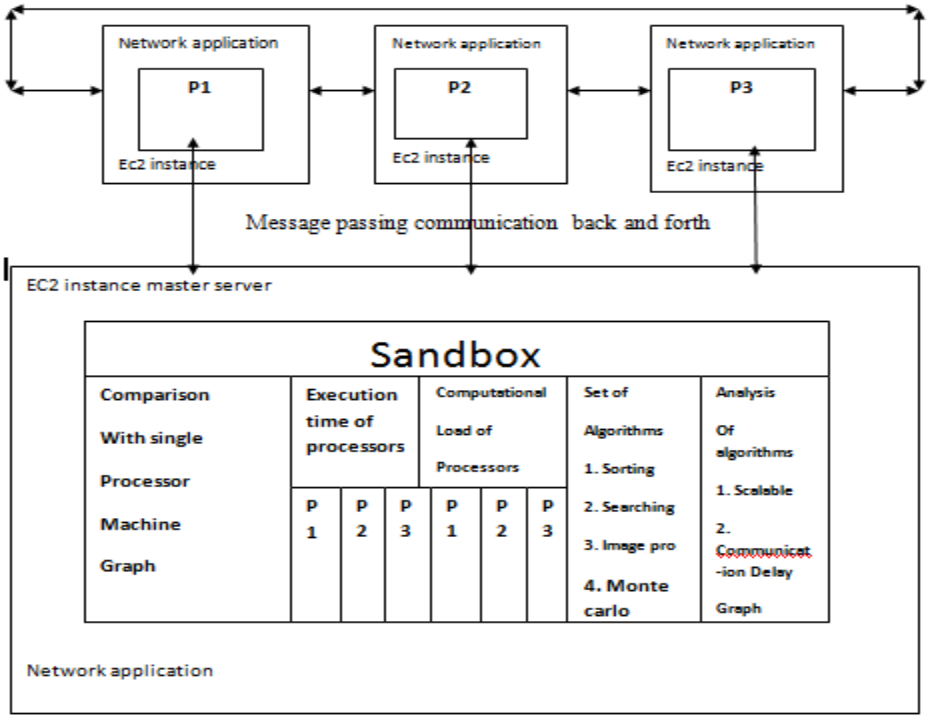


Figure 2 Schematic view of Sandbox.

7.1 Prior study related to Sandbox

Prior to this study, IBM conducted a research on the performance of parallel algorithms and parallel processing languages. This research was conducted on parallel programs for concurrency, computational load, time and computational complexity of an algorithm. This research helped to implement message-passing algorithms. Thus, I developed the sandbox to go deep into the parallel algorithms. The first step in this thesis was to create few Amazon Elastic cloud-computing instances as shown in the schematic figure above. One of the Ec2 instances is

acting as a master server. Here users can select the algorithm and pass the appropriate input. This instance is the sandbox. The sandbox has a comparison of execution time and computational load of each processor. Sandbox gives an analysis of scalability of the algorithms and shows us what the communication overhead.

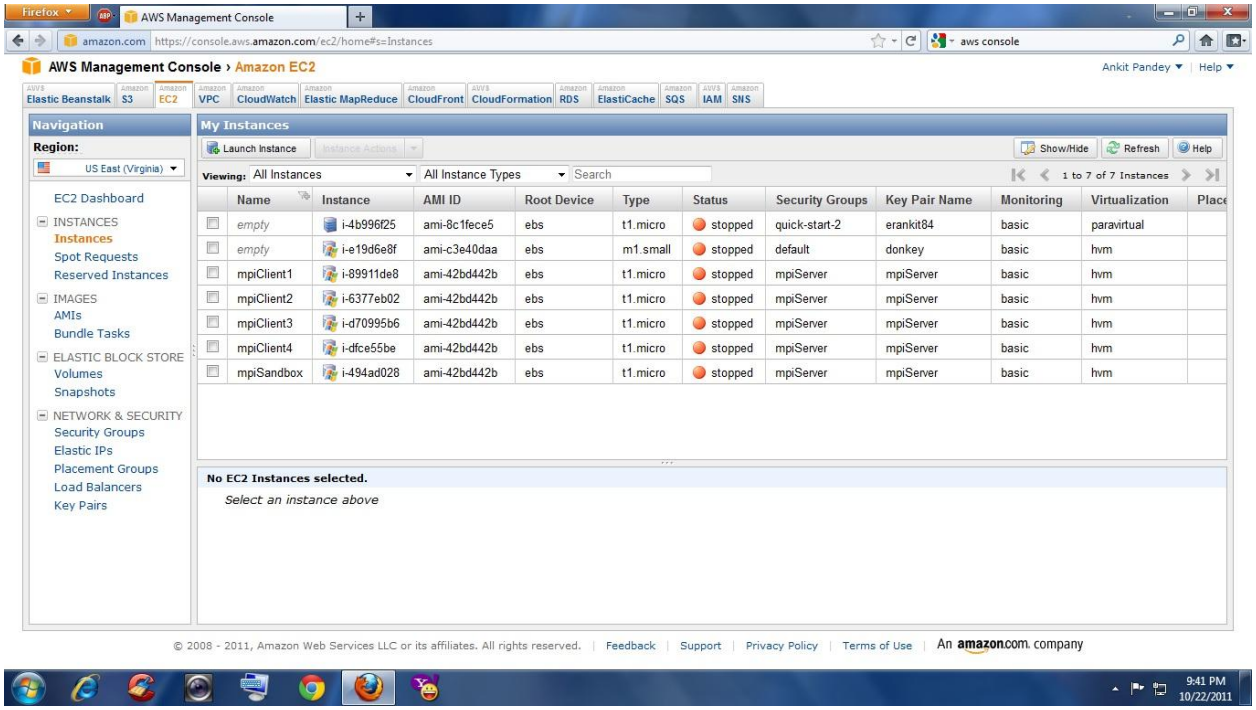


Figure : 3 Instance View Amazon Console

In the figure 2, there are EC2 Instances with the configuration of windows 2003, 4 GB RAM. To create an instance on EC2 user have to click on the launch button and follow the on screen

instructions.

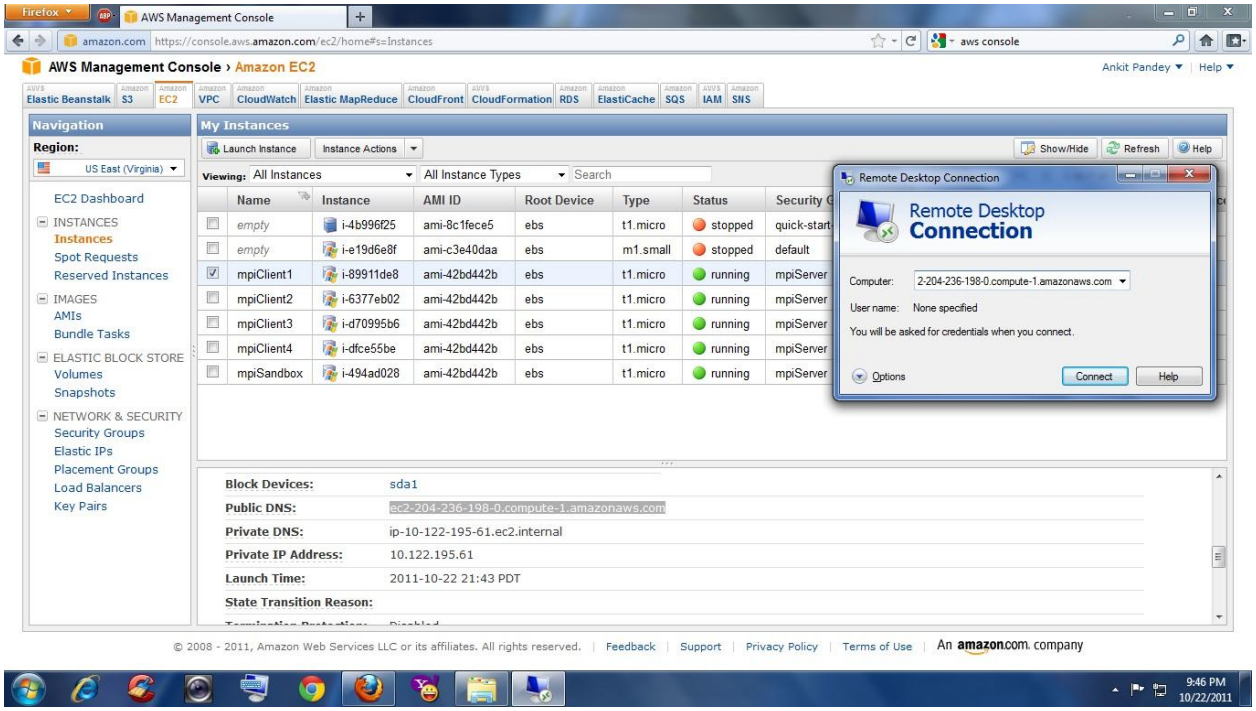


Figure 4 connecting to Ec2 to instances

This figure shows the view of EC2 instances that is already created. This figure explains how we can connect to a particular instance with the help of its IP address assigned to it. EC2 SQS server assigns the IP address. In the figure the highlighted portion have the IP address like ecc2-204-236-198-0.compute-1.amazon.com. This can be used to connect to the instances with the help of OS utility that is remote desktop connection or we can use ssh programs like putty.

7.2 Sandbox Functionalities

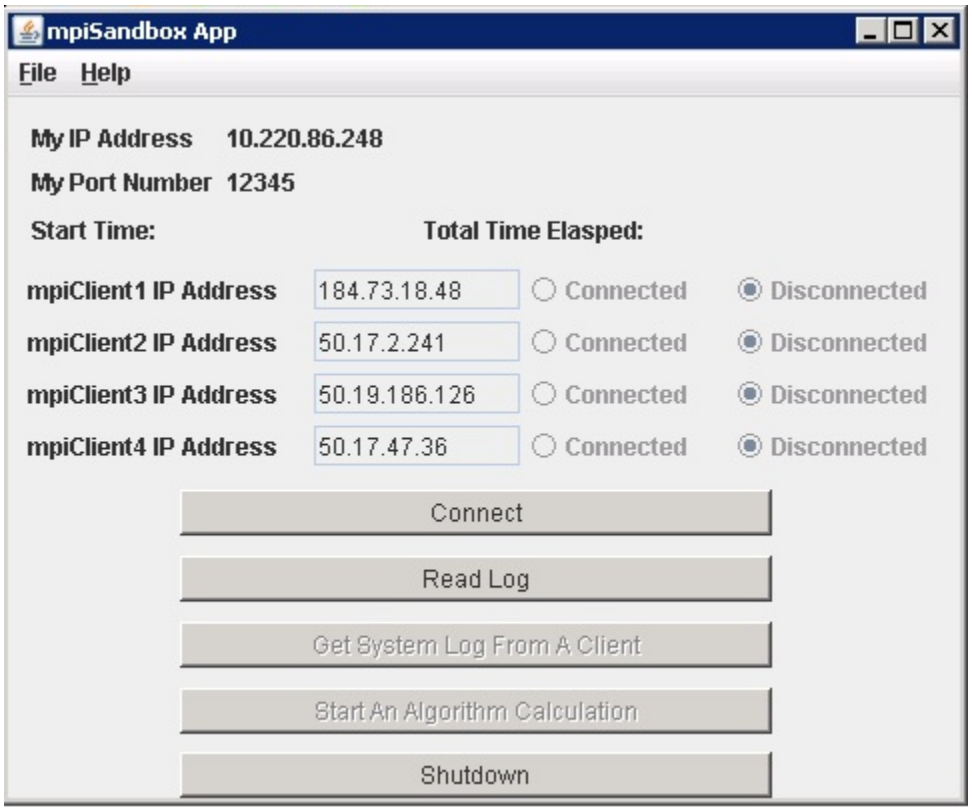


Figure 5 Sandbox GUI

The sandbox has a GUI created using Java Swing. The sandbox can be looked at as the server, which monitors the mpiClients, sends out computation requests and notifies the mpiClients about the location of all other parallel servers. Once started the sandbox has the following options:

- Connect to mpiClients: here the sandbox gets the ip addresses of all participating parallel mpiClients, sends the location of other mpiClients to each one, along with its position, which is used for, know how to split computation work and communicate amongst themselves.

- Request for mpiClients logs: mpiClients have logs that store information about connection requests and computations that they are working on. The sandbox can request the logs of specific mpiClients
- Read sandbox log: in addition to being able to obtain the logs of all mpiClients, the sandbox has a log that stores connection requests, computation requests and the results of computations performed by mpiClients.
- Ask mpiClients to perform a calculation: the sandbox can request all connected mpiClients to perform a calculation and get the result back from the MpiClients. The sandbox is quite an elaborated system, utilizing various java classes and networking protocols in supporting the message-passing interface. The project relies heavily on networking and UDP packets are the primary transport mechanism. The sandbox creates byte streams and uses a custom-made frame class for this project to populate a packet, which is sent to mpiClients for whatever request. The sandbox also limits the packets sent or received to the specific size of the buffer allowed for the networking socket on the underlying platform.
- Concurrency within the clients is maintained in separate the sending and receiving functions of the sandbox. A function is created to handle sending of packets, reading logs and basic functions within the sandbox. Another function is also created that handles receiving packets, logging the information about the packets and performing a function based on the contents of the packet.

7.3 Classes in the Sandbox

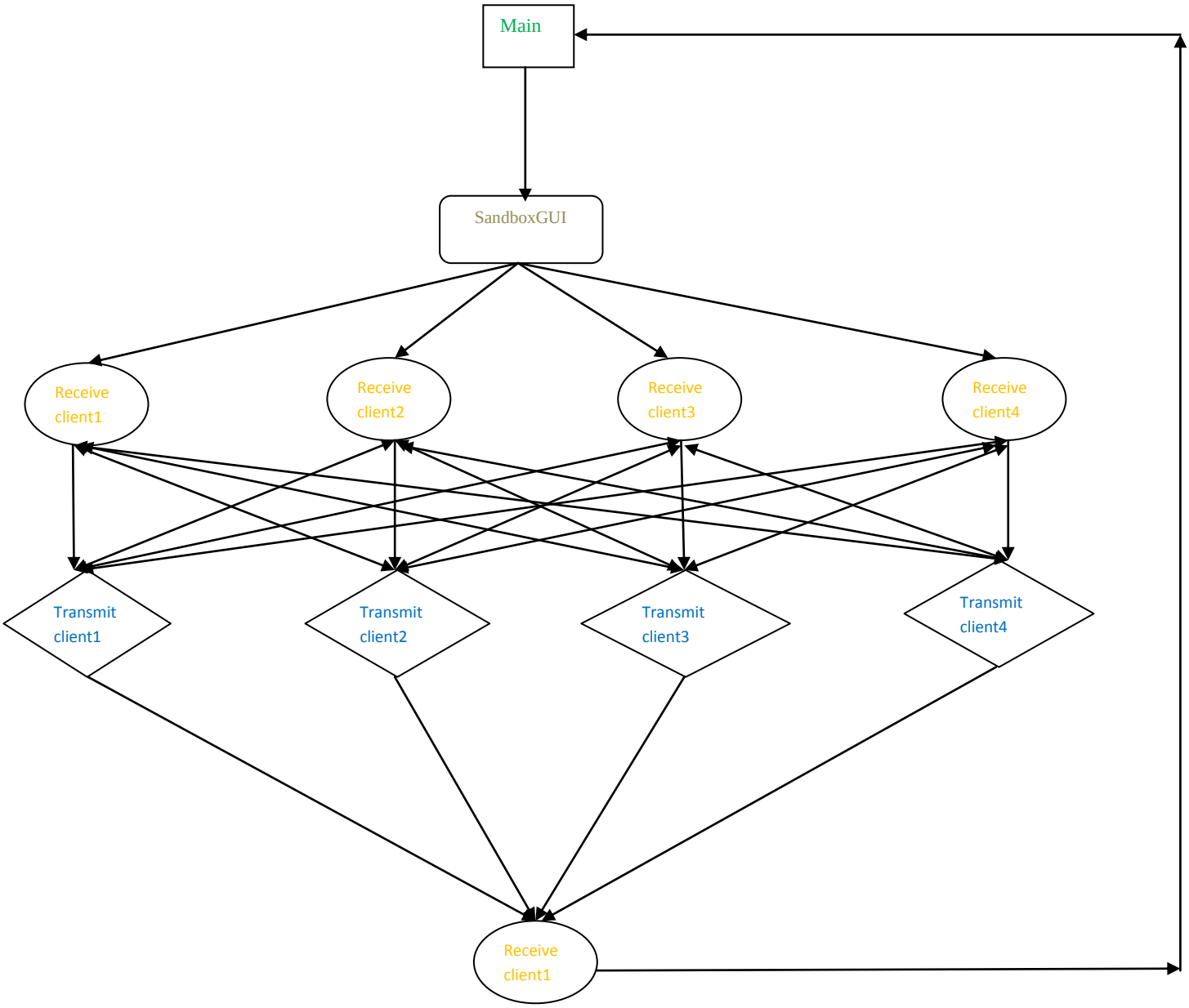


Figure shows the classes interactions in the sandbox

Classes in the sandbox interact the way it's shown in the figure above. Main class is called for the first time and then it calls the SandboxGUI class, which opens up the GUI of the sandbox.

Sandbox GUI provides the functionalities with the help of which users can connect to the number of instances they want. Sandbox GUI interacts with the receive functions and then. The receive functions of client interacts among them self to process the parallel algorithms. Finally, when the input is completely processed by clients they send their results to client1. Client1 combines the results from other clients and send it to Sandbox.

7.3.1 Classes common in Sandbox server and Mpiclients Servers

- NetworkException.java: this is the class that handles exceptions when sending or receiving data using UDP sockets
- MyFrame.java: this class simulates a UDP packet. It contains fields for source/destination address, packet type and message. This is a specially created class for this project to handle the needs of distributed computation. The source/destination address field contains sender/receiver of a frame. The packet type is used to differentiate between connection request packets, computation request packets, log request packets and results of a computation packet. The message field typically contains data to be computed, IP addresses of other mpiClients or the result of a computation request.

- **Main.java:**

  This is the main class that initiates the sandboxGUI.java class. After initiating it, the sandBoxGUI.java class starts the receiver thread, waits a second for proper initialization of program variables then makes the sandbox GUI visible.

7.3.2SandboxGUI.java

This is the key class in the sandbox program. This class handles the GUI, networking, connection requests, and computation requests and gets results of computation. The GUI is created using Java Swing and the components in the GUI use the code here to manipulate how they are presented or what to do when certain buttons are click. The GUI is very user friendly and hides addition windows behind buttons. When buttons are clicked, the respective windows are displayed. The computation window has the option of requesting a computation amongst the parallel mpiClients or performing the calculation locally sequentially.

The GUI provides the interface that controls the whole project Once started, the GUI initiates two threads, sender () and receiver () which handle transmitting and receiving packets respectively. The start page for the GUI provides the options to connect to the mpiClients, start an algorithm and read Sandbox/MpiClient logs. A separate java swing frame handles the algorithm sending to MpiClients and receiving results. Following are the variables used in the sandbox for the functions to act accordingly. When a server receives L it sends a log request, S has all the IP address of the servers connected to sandbox. When sandbox receives a packets with the letter C it knows that client has received the packets.

L = Log request, S = All other server IP addresses from sandbox, C = Client has received server locationsprivate String firstBit = "";

```
private static final int PORTNUM = 12345;
 private String[] connectedServers = new String[4];
 private String[] respondedServers = new String[4];
```

To make a inter-processor communication, port number 12345 is used for connection from Sandbox to other servers. First few lines are the array declaration to store the number of servers connected and responded. First, one is connectedServers that have the ip address of the servers connected, after the connection has established. Second, one is the respondedServers to store the IP addresses of the servers that have responded to location packet sent after the connection has established.

```
private long startTime; // time parallel algorithm has been running
// get a datagram socket
DatagramSocket socket = null;
// to write to the receiver log file
BufferedWriter bw = null;
// the sender thread to handle sequential algorithm as well
sender mySender = new sender();
```

Second important function in the sandbox is the transmit function which is responsible for sending the packets between the sandbox and other servers to and from. It takes a frame, a destination IP address and a description of what kind of packet this is, specified by a bit it opens a byte stream and writes the required information to the byte stream byte stream is then packaged as a frame and sent to the destination IP address. Transmit function use bytestream variable array to write a output frame into the buffer. The variable bytestream have the size 102040 bytes which is used to write the operating system's output stream by the transmit function itself. Before it writes on the buffer it flushes out the buffer to make it clean and ready to be written with the packet details.

```
public void transmit(myFrame frame, String destIP, String firstBit) throws NetworkException {
try {

        // bytearraystream and outputstream handle writing frame info to buffer
        ByteArrayOutputStream byteStream = new   ByteArrayOutputStream(10240);
        DataOutputStream os = new DataOutputStream(byteStream);
        os.flush(); //flush/clear the outputstream before we write to it
        os.writeBytes(firstBit); // first byte tells us about the packet.
        os.write(frame.preamble); // just used for error checking later
        os.write(frame.sourceAddress); // this sandbox's ip address
        os.writeShort(frame.packetType);// packet type used to differentiate between algorithm requests, connection requests, etc
        os.writeBytes(frame.message); // the actual message to be send
        byte[] buf = byteStream.toByteArray();//retrieves byte array and store in buffer
        InetAddress address = InetAddress.getByName(destIP);     // address we want to send packet to
        DatagramPacket packet = new DatagramPacket(
                        buf, buf.length, address, PORTNUM);    // set the datagram packet with the port number
        socket.send(packet);    // send the packet
        os.close(); // close the output stream
    } catch (IOException ex) {Logger.getLogger(sandboxGUI.class.getName()).log(Level.SEVERE, null, ex);}}
```

7.3.3 SandboxGUI Receive Function

SandboxGUI has a receive function which is a very elaborate function that retrieves packets sent by the MpiClient. Packets could be connection confirmations, results of algorithm calculations or log files. It creates a packet to receive and send to other MPI servers. All communications are

saved in the log file with the date and time. These logging are done by bw variable. In the lines below there is bw.foo () function this is used for writing, reading and logging. This bw is used for all kind of inter-processor receive or send request. This SandboxGUI's receive function does the calculations and start the communications when user initiate an algorithm. The byte count variable is to get the length of the packet. Data input stream is created to store the incoming message. Then from the input stream, the information is extracted in the variable "a". The variable "a" contains the source address, frame type, and the results. These things are arranged in an order. In the packet the first byte is very important because the first byte tells the receiver about the kind of message it has received. If the first byte is one of the letters "R", "L", "S" it does the following things: If the first byte is "R" then it tells the receiver that the message contains results from other Mpiclients. If the first byte is "L" then there is Log request from the sandbox. If the first byte is "S" it means that the sandbox is sending a request to make a connection. The time is calculated in a way that whenever servers receive the message and they read the first byte the timer gets started. Servers send their calculated time to the Mpiclient1. MPiclient1 calculate the total time taken by each server and sends the resultant time to the sandbox.

```
public void receive() throws NetworkException {   //System.out.println("i received");
      try { byte[] buf = new byte[10240];
          DatagramPacket packet = new DatagramPacket(buf, buf.length);    // packet to receive the data
     socket.receive(packet); // receive packet from socket and  write information about packet to communications log
          Date d = new Date();
          DateFormat df1 = DateFormat.getDateTimeInstance(DateFormat.MEDIUM, DateFormat.MEDIUM);
```

```
String s1 = df1.format(d);

bw.newLine();

bw.write(s1);

bw.newLine();

bw.flush();

System.out.println("Packet Received. Read communications log for more information");

// display information of who sent the packet

bw.write("Packet received from " +

    packet.getAddress().getHostName() + ":" +

    packet.getPort());

bw.newLine();

bw.flush();

int byteCount = packet.getLength();

// the lenght of the packet. used to know how to retrieve the data

// bytearraystream and inputstream to retrieve date from the packet

ByteArrayInputStream byteStream = new  ByteArrayInputStream(buf);

DataInputStream is = new DataInputStream(byteStream);  // stream for reading information

// Extract all possible information from packet

byte a = is.readByte();

String frameType = new String(new byte[] {a});  // read the first byte to know what kind of packet we are dealing with

if (frameType.equals("R")) { // result of some algorithm calculation

    long elaspedTime = System.currentTimeMillis() - startTime; // get the time taken to perform the calculation

    totalTimeLabel.setText(formatSeconds((int) (elaspedTime / 1000F))); // write it in the label in the GUI
```

```
}
// extract preamble. Just a holder to tell us in future if this is a packet by this program
byte[] framePreamble = new byte[8];
for (int i = 0; i < 8; i++) {
    framePreamble[i] = is.readByte();}
   // extract source address
String frameSourceAddress = "";
for (int i = 0; i < 6; i++) {
    a = is.readByte();
    String b = Integer.toHexString(a).toUpperCase();
    frameSourceAddress = frameSourceAddress + ":" + b;}
frameSourceAddress = frameSourceAddress.substring(1);
// extract packet type
short framePacketType = is.readShort();
String framePacketString = Integer.toHexString(framePacketType);
bw.write("Frame Source Address: " + frameSourceAddress);
bw.newLine();
bw.write("Frame Packet Type: 0x" + framePacketString);
bw.newLine();
bw.flush();
// extract frame message
String myFrameMessage = "";
for (int i = 0; i < (byteCount - 17); i++) {
    a = is.readByte();
    String b = new String(new byte[] {a});
```

```
            myFrameMessage = myFrameMessage + b;
        }
        if (frameType.equals("L")) { // log packet from an mpiclient
            frameMessage.setText(myFrameMessage);
    // display the retrieved log in the text area in the GUI
            bw.write("Received system log from an MpiClient");
            bw.newLine();
            bw.flush();
        } else {
            bw.write("Ethernet Frame Message: " + myFrameMessage);
            bw.newLine();
            bw.flush();
        }
        if (frameType.equals("C"))
// response gotten from mpiClient for connection. update response table
    {
            for (int i = 0; i < connectedServers.length; i++)
            {
                if (connectedServers[i].equals(packet.getAddress().getHostAddress())) // update
table to show sandbox now connected to this
                {
                    connectionSwitch(i);
                    respondedServers[i] = "responded";
                    bw.write("Now connected to this mpiClient");
                    bw.newLine();
```

```
                bw.flush();
                break;
            }
        }
    }
    is.close(); // close the stream

} catch (IOException ex) {
    Logger.getLogger(sandboxGUI.class.getName()).log(Level.SEVERE, null, ex);}}}
```

7.3.4 Receive Function of Mpiclients

This function is similar to the sandbox receive function it has some other functionalities responsible for the algorithmic calculation. It has function that receive the packets with different integer in the variable a. If it receives the letter "A" as a first byte, it tells the client that there is request for a particular type of algorithm. In the first few lines of the code there is an if statement which is true if first byte is letter "A" and if it's true it check the frame message. In the frame message if the packet string equal to 101 or 201 it calls the Sorting algorithms. It dissembles the message to get the input array. The call for the sorting algorithms is further classified, if the packet string is equal to 101, the Parallel merge algorithms is invoked. Once a frame is received, the mpiclient dissembles the message to know what kind of frame it is. if it is 101, this is a direct message from the sandbox to start a merge sort. If it is 102, it is a bubble sort. The message from such packets is numbers to be sorted. The message is a string so it has to be convert to a string array and the individual elements sorted.

```
if (framePacketString.equals("101") || framePacketString.equals("201")) // 101 = mergesort, 201
= bubblesort
            {
               // reset currentStore
               currentStore = "";
```

CurrentStore is a string I used to store the sorted arrays from the mergesort. Each time an mpiclient has sorted numbers, it stores it as a string here. When an mpiclient receives a sorted array from another mpiclient, it combines the two into the current store after sorting it.

```
               // dissemble message
               StringTokenizer st = new StringTokenizer(frameMessage);
               ArrayList<String> myList = new ArrayList<String>();
               while (st.hasMoreTokens())
               {
                  myList.add(st.nextToken());
```

This is where the numbers to be sorted in the array are extracted from the string array.

```
               }
               int[] myArray  = new int[myList.size()];
```

The integer array is initialized here and in the next for loop, the numbers are put into the array.

```
               for (int i = 0; i < myList.size(); i++) {
                  myArray[i] = Integer.parseInt(myList.get(i));
                  //myArray[i] = ((Integer) tempArray[i]).intValue();
               }

               if (framePacketString.equals("101")) {
                  mergesortTime = System.nanoTime();
```

Once the mpiclient has extracted the message, it starts the timer. each algorithm has its own timer. bubblesort has bubblesortTime. Below the mpiclient then starts the actual sorting calling function myAlgorithm.mergeSort();

```
                  System.out.println("Got mergesort calculation request. Starting...");
                  myAlgorithm.mergeSort_srt(myArray, 0, myArray.length - 1);
                  System.out.println("Completed mergesort calculation request");
               } else {
```

```
            bubblesortTime = System.nanoTime();
            System.out.println("Got bubblesort calculation request. Starting...");
            myAlgorithm.bubbleSort(myArray);
            System.out.println("Completed bubblesort calculation request");
        }

        iterationNum = 1;
```

iterationNumber is a way of me know what level the mpiClient has gone in mergesorting. it is 1 because this is the first iteration after getting the message from the sandbox. Anytime this mpiclient gets another array from an mpiclient to sort and combine, it increases the iteration number. The iteration number helps in the calculation to know who to send a message to, whether another mpiclient or to the sandbox

```
    for (int i = 0; i < myArray.length; i++)

        {

            currentStore = currentStore + Integer.toString(myArray[i]) + " ";
```

currentStore will now contain the results of the previous store and the now sorted array

```
        }

        if (myPosition % 2 == 0)
```

if this is an even numbered server and if this mpiCLient is say number 2 or 4 or 8, it sends the result to a neighbouring mpiclient

```
        {
            myFrame frame = new myFrame();
            frame.message(currentStore);
            frame.setUpIP();
            if (framePacketString.equals("101")) {
                frame.packetType((short)0x0102);
            } else {
                frame.packetType((short)0x0202);
            }
            try {
              send the frame ,transmit the frame, then log the messages.

                transmit(frame, connectedServers[myPosition - 2], "A");    // send the frame
                bw.newLine();
                d = new Date();
                df1 = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
```

```
DateFormat.MEDIUM);
                    s1 = df1.format(d);
                    if (framePacketString.equals("101")) {
                       bw.write(s1+ ": Mergesort done. Result to adjacent server: " + currentStore);
                       System.out.println("Mergesort done. Result to adjacent server: " +
currentStore);
                    } else {
                       bw.write(s1 + ": Bubblesort done. Result to adjacent server: " +
currentStore);
                       System.out.println("Bubblesort done. Result to adjacent server: " +
currentStore);
                    }
                    bw.newLine();
                    bw.flush();
                 } catch (NetworkException ex) {
                    Logger.getLogger(myClientNIC.class.getName()).log(Level.SEVERE, null,
ex);
                 }
              }
```

8. How Parallel Algorithms Work in Sandbox?

8.1 Parallel Merge sort

Merge sort is an efficient divide-and-conquer sorting algorithm [3]. The average complexity of merge sort is O(n log n), the same as quick sort and heap sort. In addition, best-case complexity of merge sort is only O(n), because if the array is already sorted, the merge operation perform only O(n) comparisons; this is better than best case complexity of both quick sort and heap sort. The worst case complexity of merge sort is O(n log n) , which is the same as heap sort and better than quick sort. However, classical merge sort uses an additional

memory of n elements for its merge operation (the same as quick sort), while heap sort is an in-place method with no additional memory requirements.

The average, best, worst asymptotic complexity of merge sort is at least as good as the corresponding average, best, worst asymptotic complexity of heap sort and quick sort; despite of this, merge sort is often considered to be, slower than the other two in practical implementations. On the positive side, merge sort is a stable sort method, in contrast to quick sort and heap sort, which fail to maintain the relative order of equal objects.

Parallel Sorting Algorithm

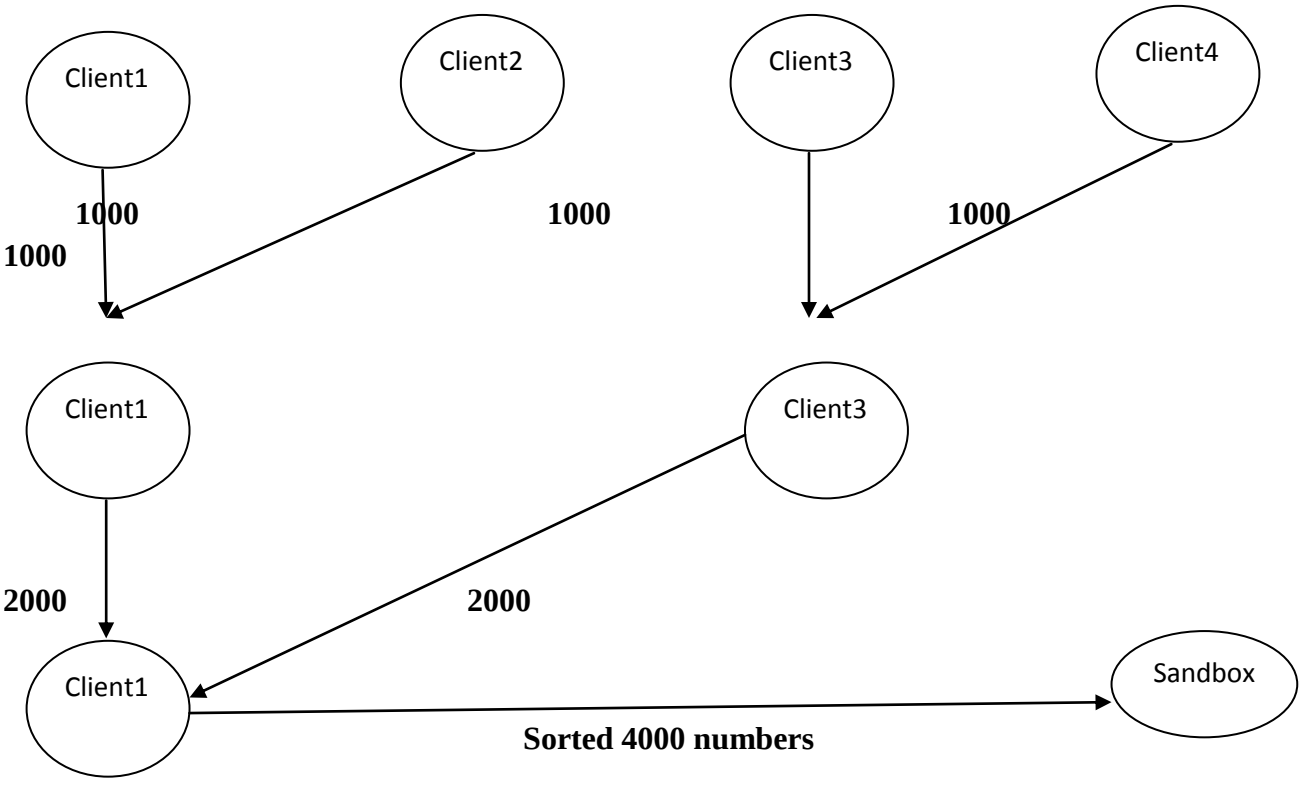


Figure 9 Parallel Merge Sort

The figure explains how the parallel merge sort works it start from the sandbox when a user initiates an algorithm. The algorithm call from the sandbox have an input which is being divided in the servers according to the rule that I have setup, to make merge sort support distributed processing. This rule is a combination of three functions. These functions decides the how servers should communicate during the calculation and which server should sends the local result from the current store, which server should get the local result from other servers? It can be explained by the figure above. Suppose there are four clients and one master server and an input array have 4000 numbers to sort. Then this array can be partitioned among the server by the sandbox. Once the clients have their share of input they call the relevant algorithm to sort. The sorted array is then saved in the current store of the client. Now all the clients have a sorted array of number receive function checks its own position and then it sends the sorted array to the adjacent client. Adjacent client merge the two arrays one its own and second the array it receives from the other clients. This continues until client one have full size array like the input array passed to sandbox. Once client one has a full array it sends the resultant array to the sandbox with a specific message type “R” which tells sandbox that calculation is completed and it has received the result from the client one.

```
class algorithms {
   public static void Parallel_mergeSort_srt(int array[], int lo, int n) {
      int low = lo;
      int high = n;
      if (low >= high)
      {
```

```
        return;
    }

    int middle = (low + high) / 2;
    mergeSort_srt(array, low, middle);
    mergeSort_srt(array, middle + 1, high);
    int end_low = middle;
    int start_high = middle + 1;
    while ((lo <= end_low) && (start_high <= high))
    {
        if (array[low] < array[start_high])
        {    low++; }
      else {  int Temp = array[start_high];    for (int k = start_high- 1; k >= low; k--)
            {   array[k + 1] = array[k];    }
            array[low] = Temp;
            low++;
            end_low++;
            start_high++;
        }
    }
}
```

8.2 Parallel Bubble sort

Parallel bubble sort do follow the same rule mentioned above like the array is divided and passed to the clients. Client then call the sorting algorithm for sub array sent to them from the sandbox. Then after first iteration, sorted array is sent to adjacent client. Two arrays are merged and again bubble sort algorithms is called to sorted the merged array, this continues until client one does not have sorted array of the same size of the input array of the sandbox.

```
public void bubbleSort(int array[]) {
    boolean swap = true;
    int j = 0;
    int temp;

    while(swap)
    {
       swap = false;
       j++;
       for (int i = 0; i < array.length - j; i++)
       {
          if (array[i] > array[i + 1])
          {
             temp = array[i];
             array[i] = array[i + 1];
             array[i + 1] = temp;
```

```
                swap = true;
            }
        }
    }
}
```

8.3 Parallel Monte Carlo Pi

```
public double monteCarloPi(int accuracy) {
    Random generator = new Random();
    int count = 0;
    for (int i = 0; i < accuracy; i ++)
    {
        double x = generator.nextDouble();
        double y = generator.nextDouble();
        if (x*x + y*y <= 1.0) {
            count++;
        }
    }

    return 4.0 * count / accuracy;
```

```
    }
```

8.4 Parallel Standard Pi

```
public double standardPi(int accuracy) {
        double area = 0.;
        double h = (1. / accuracy);

        for (int i = 1; i <= accuracy; i ++)
        {
            double a = (h * i) - (h / 2.);
            double b = Math.sqrt(1 - a * a);
            area += h * b;
        }

        return 4 * area;
    }
```

9. Scalability

9.1 Adding Instances to Sandbox

To make the sandbox run the algorithms with different number of processor we have to launch all new Ec2 instance form the scratch. Creating a new instance would take half an hour; we have to press the launch button in the Amazon Ec2 console to create the instance. Users can select any windows server they want during the configuration of machine but it is advisable to use the micro instances with 4 G B ram and 40 GB hard drive because this will keep the cost of research low. Instances can be created with same key used for the old instances or new key can be created. Amazon SQS server assigns these keys and credentials that we can download with an option available at console. Once the instance is created, we have to wait for fifteen minutes, after this the newly created instance will show up in the list of available instance. Now those instances can be started by right click and can be connected by the local machine using remote desktop connection or putty. In the new instance users have to open up the inbound and outbound connection at control panel. These inbound and outbound connections tell the OS that machine can accepts and sends the packets in a network via ports and sockets. Once the inbound and outbound connections are setup we can install the Java SDK latest version, Netbeans 7 and Firefox or any web browser.

9.2 Adding MpiClient Classes to New Instances

MpiClients classes can be added in the new instances after the installation of the Netbeans. In the Netbeans Editor we have to create project to add the MpiClient classes, this can have any name the only thing that makes it to connect to the other server and the Sandbox is the Java socket programs using same port number used in other clients and Sandbox. The port number, which is

used in the Sandbox and other clients, should be used in the new client to accept and send the packet. To accept the packet first thing in the instance is the socket program that has functions to open data input and output stream and have the similar features as other instances. New classes should be made in the instance then we can just copy the code of the classes of other MpiClients in the new instance classes.

10. Results

Results can be viewed in the log tables of sandbox that gives the comprehensive results. Sandbox has two options to view the logs first one is the main log table of sandbox which has the logs from sandbox and log details of other servers. The second option is to view the log tables of a specific client. Figure 7 is the main log of sandbox, which have all the results. Figure 8 is the log view of client in this figure there is a radio button with the help of which users can choose the specific client to view its result.

Figure 7 Main log of Sandbox

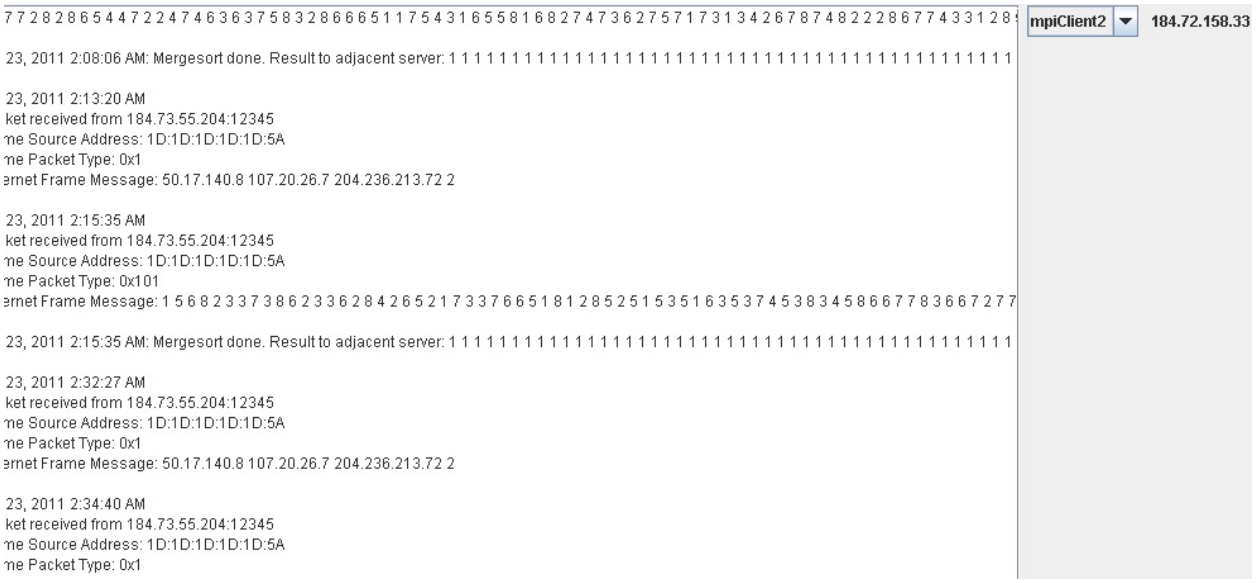


Figure 8 Log View Mpiclients

10.1 Merge sort results

	Merge Sort Results Four Instances				
Input	Time in Milliseconds	Response time	Input	Overheads	Complexity
Parallel Merge Sort	-16395747895216	T	8000	#t	O(n log n) + overhead
Sequential Merge Sort	0.93	T	8000	t	O(n log n) + overhead
Parallel Merge Sort	-16926144110757	T	8000	#t	O(n log n) + overhead
Sequential Merge Sort	109	T	8000	t	O(n log n) + overhead
Parallel Merge Sort	-17216649397114	T	8000	#t	O(n) + overhead
Sequential	0.47	T	8000	t	O(n) +

Merge Sort					overhead

Log results from the sandbox Merge Sort

This is the log result of Parallel merge sort when a user initiates a parallel merge sort algorithms, the first happens is the inter-processor connection between the processors. Once the connection is established user can send the algorithm request form the sandbox to all the Mpiclients along with an input.

Results can be explained in two steps

Steps (1) Algorithm calculation request.

Below is the log results of the 8000 random numbers with Parallel sort.The arrows between sandbox and MpiClients indicate that sandbox has sent an Algorithm calculation request.

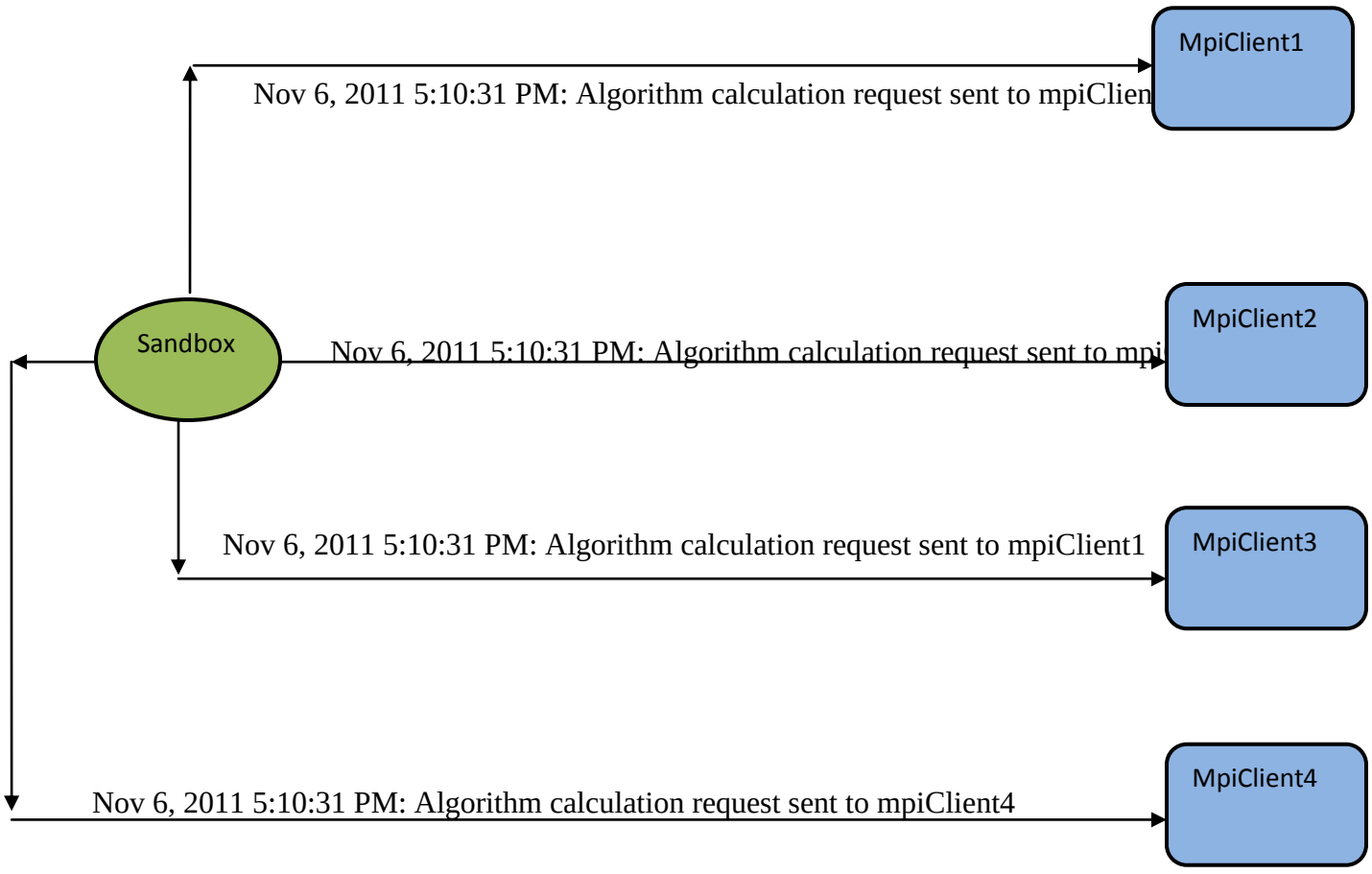
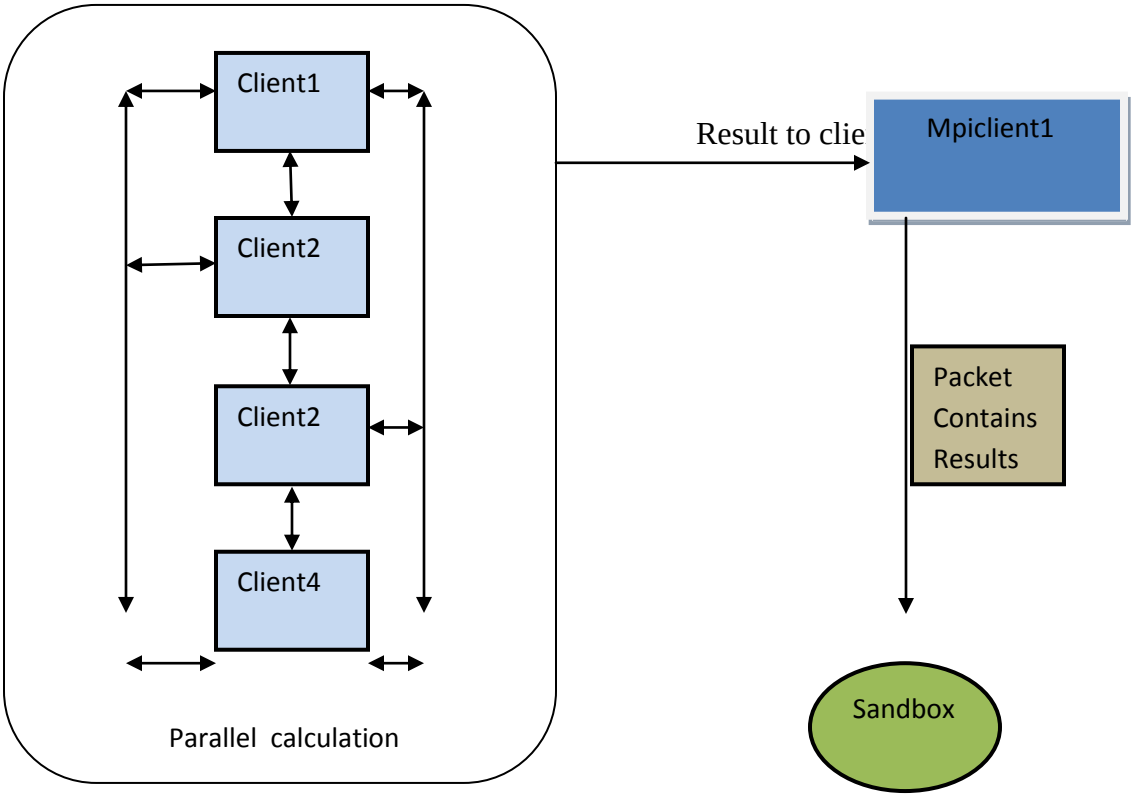


Figure: Algorithm request from sandbox

Step (2) Packets received from Mpi client 1



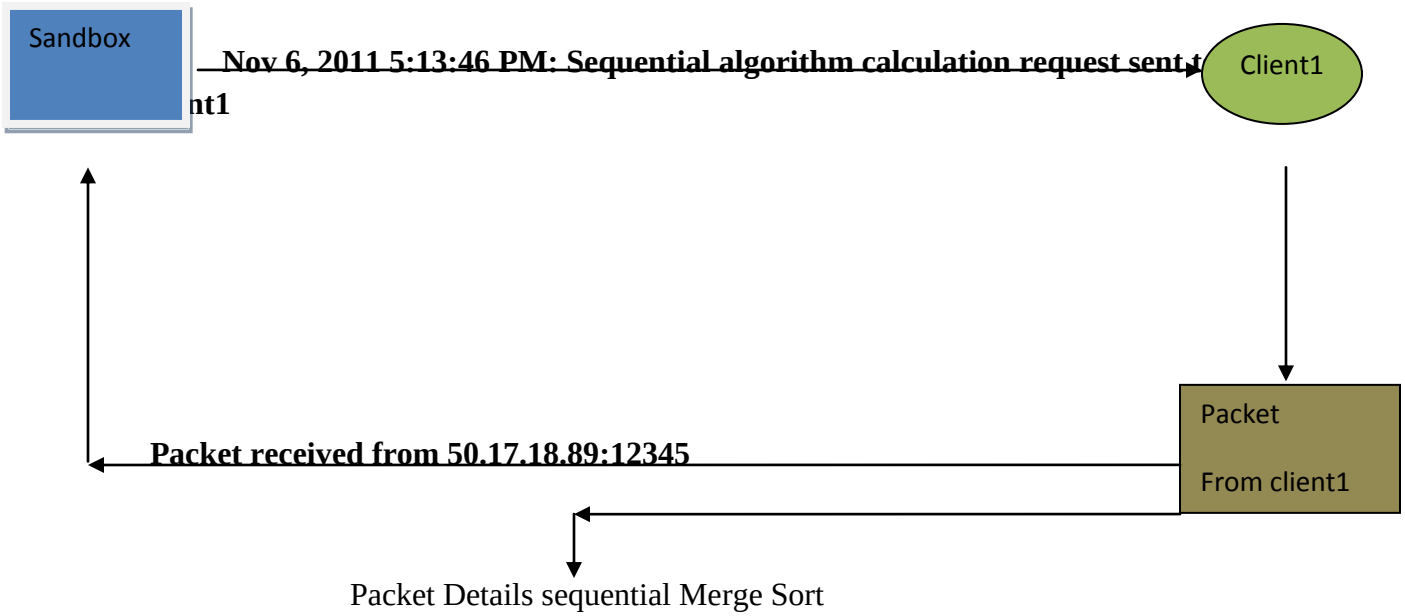
Sandbox sends an algorithms request to all the Mpiclients, and all Mpiclients complete their calculation and send the result back to the mpiclent1. Mpiclient1 assembles results of other clients and send the result to sandbox.

The IP address with the packet containing results is the IP address appended with port number in the last belongs to Mpiclient1. This port number is defined in all the servers and is used for the back and forth communication between them.

UDP Packet details Parallel Merge Sort	
Algorithm Results	Ethernet Frame Message: 1 1 1 1 1 1 1 1s
Time to complete the algorithm	16395747895216 milliseconds
Packet Type	Frame Packet Type: 0x103
IP address of client sending final packet	Packet received from 50.17.18.89:12345
Time of Receiving the packet	Nov 6, 2011 5:10:31 PM
Mac Address of Client sending final packet	Frame Source Address: 1D:1D:1D:1D:1D:4A

Figure Packet detail parallel Merge Sort

The sixth row in the figure above is the Mac address of Mpiclient1 and following row is the type of frame message. Every algorithm has three digit integer, which tells users, and servers what kind of algorithm request is received. Here the Packet type is 103 it means that this calculation is for parallel Merge Sort. Further in the log results there is result in the form of Ethernet frame message. I have used full array of results because of its length it only has the first few sorted numbers. Then there is a result for the time taken by Parallel merge sort. The time is in milliseconds. This Merge sort was done for 8000 numbers and still the time calculation is - 16395747895216 milliseconds, which is less time. This time also have overheads which n times of communication between the processor it means that the time sandbox is telling us is more than actual time taken by the algorithm



Ethernet Frame Message: 1 1 1 1 1 1 1 1 1 1

Nov 6, 2011 5:13:46 PM

Frame Source Address: 1D:1D:1D:1D:1D:4A

Frame Packet Type: 0x103

Ethernet Frame Message: Time for Sequential Mergesort: 93 milliseconds. 0.0seconds

10.2 Bubble Sort

Log results from Sandbox

This is the log result of bubble merge sort when a user initiates a parallel bubble sort algorithms, the first happens is the inter-processor connection between the processors. Once the connection is established user can send the algorithm request form the sandbox to all the Mpiclients along

with an input. Below are the log results of the 8000 random numbers with Parallel and sequential bubble sort. Below in the figure arrows from the sandbox are indicating that the algorithms calculation request is sent to all the

Log Results Can be explained in two steps

Steps (1) Algorithm calculation request.

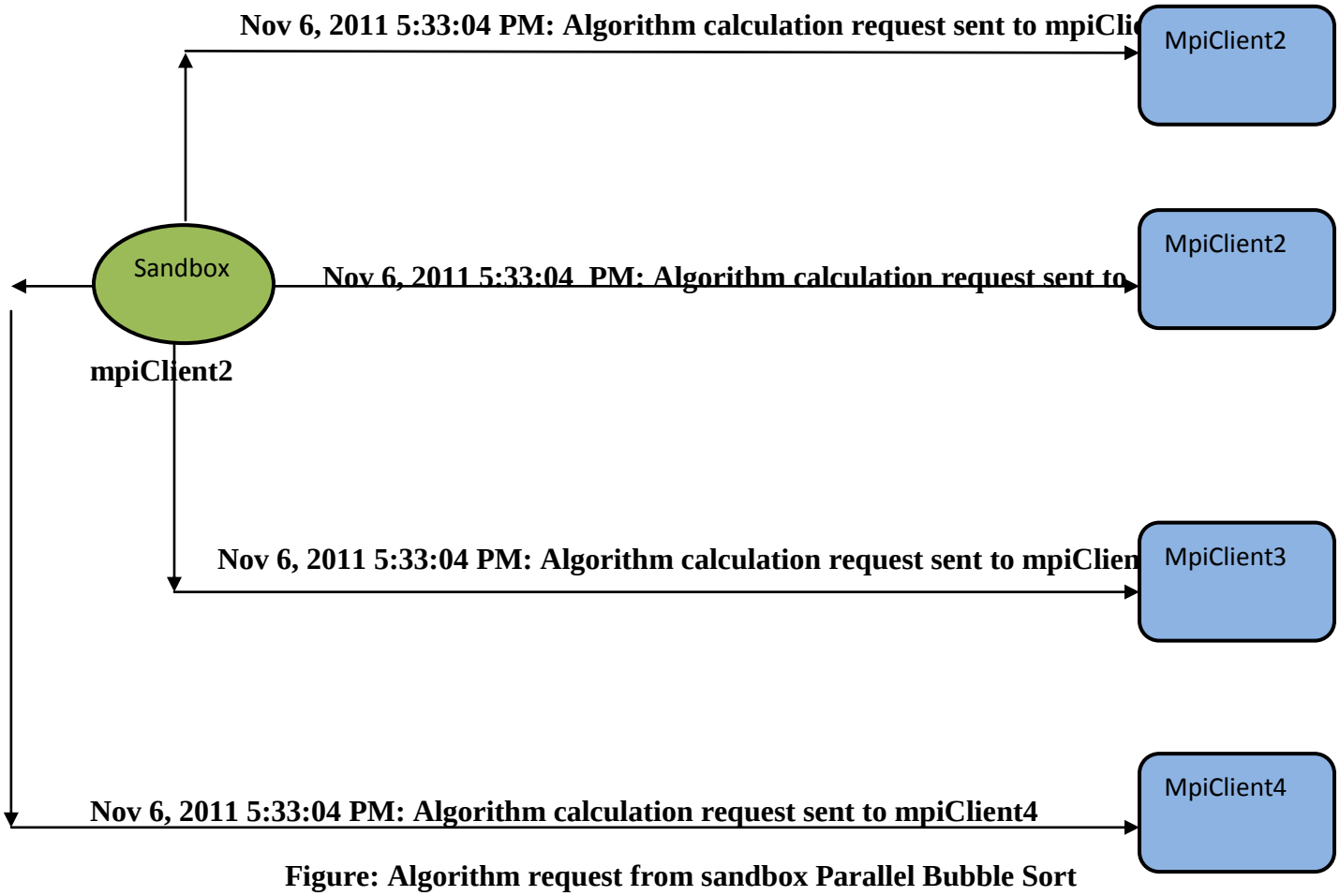
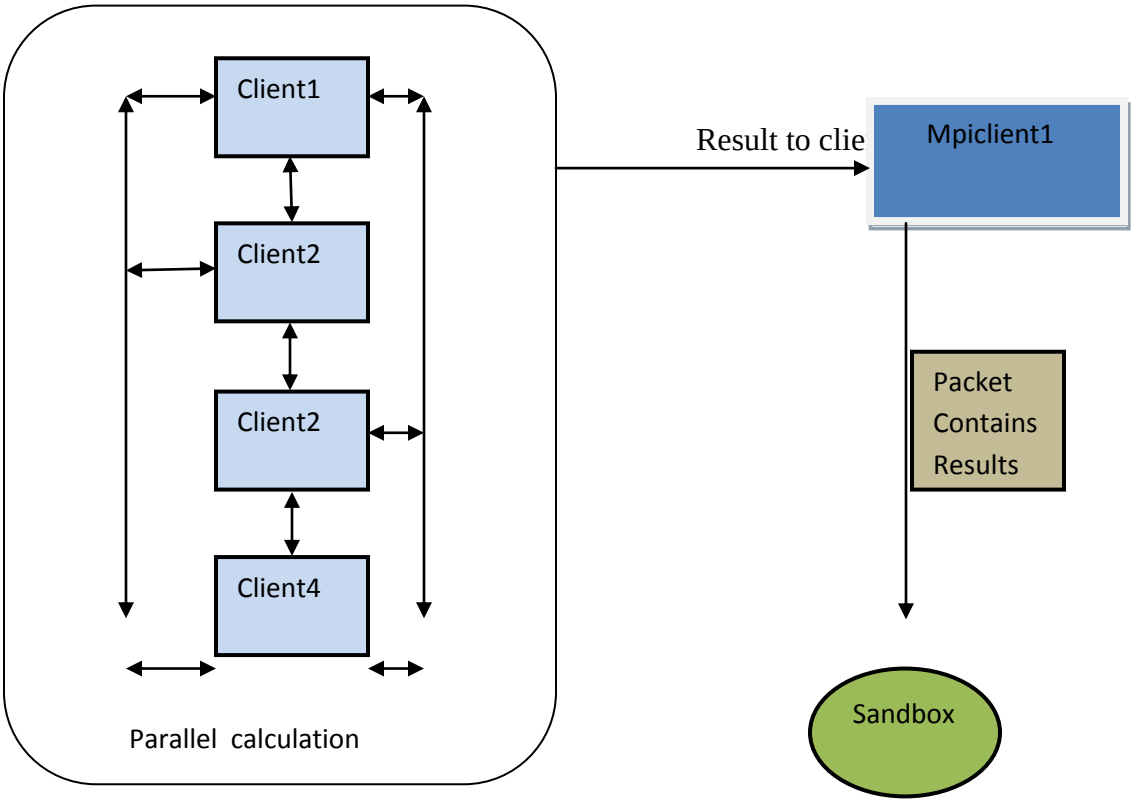


Figure: Algorithm request from sandbox Parallel Bubble Sort

Sandbox sends an algorithms request to all the Mpiclients, and all Mpiclients complete their calculation and send the result back to the mpiclent1.

Step (2) Packets received from Mpi client 1



Mpiclient1 assembles results of other clients and that result is sent to sandbox. and the following next row is. Further in the log results there is result in the form of Ethernet frame message. I have used full array of results because of its length it only has the first few sorted numbers.

Packet received from 50.17.18.89:12345 this is IP address of the MPIclient1 which is appended with port number.

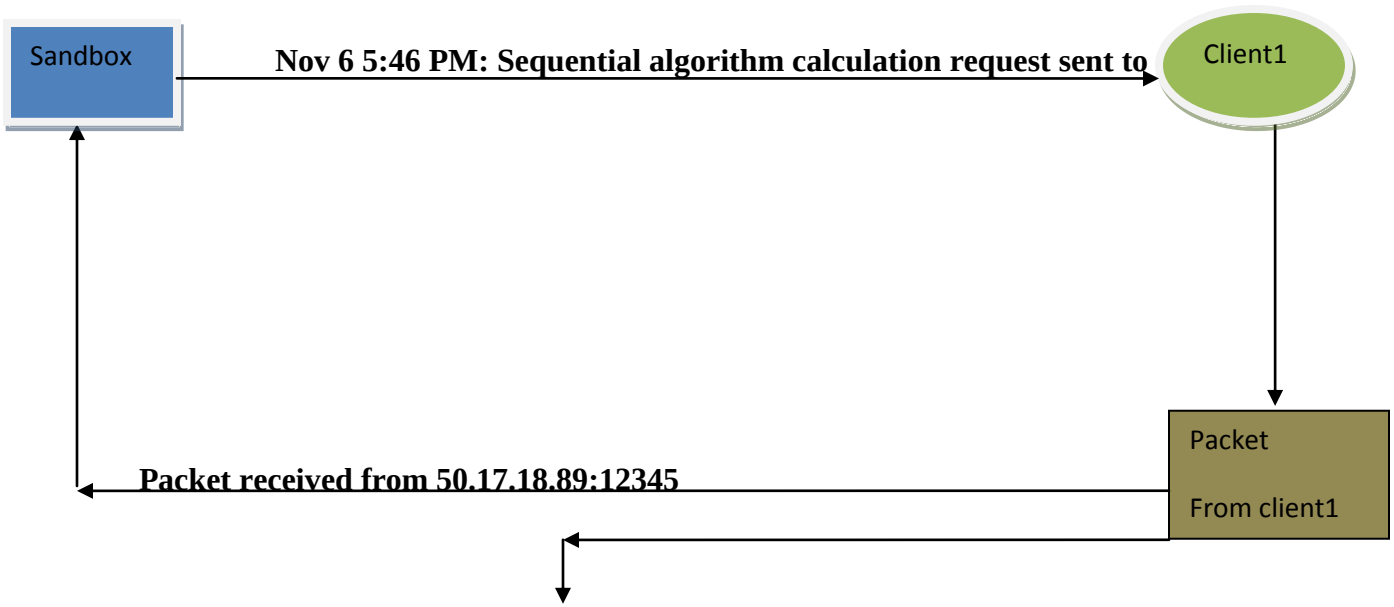
Frame Source Address: 1D:1D:1D:1D:1D:4A is the Mac address of Mpiclient1

Frame Packet Type: 0x203

Every algorithm has three digit integer, which tells users, and servers what kind of algorithm request is received. Here the message type is 203 it means that this calculation is for parallel bubble Sort. Ethernet Frame Message: 1 1 1 1 1 1 1 1 1 result

Ethernet Frame Message: Time for Bubblesort: -17672137396121 milliseconds. -16.0seconds

This is a result for the time taken by Parallel merge sort. The time is in milliseconds. This Merge sort was done for 8000 numbers and still the time calculation is -17672137396121 milliseconds, which is a less time. This time also have overheads which n times of communication between the processor it means that the time sandbox is telling us is more than actual time taken by the algorithm.



Packet Details sequential Merge Sort

Packet received from 50.17.18.89:12345

Frame Source Address: 1D:1D:1D:1D:1D:4A

Frame Packet Type: 0x203

Ethernet Frame Message: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Ethernet Frame Message: Time for Sequential Bubblesort: 94 milliseconds. 0.0seconds

	Bubble Sort Results Four Instances				
Input	Time in Milliseconds	Response time	Input (Random Number)	Overheads	Complexity
Parallel Bubble Sort	-17393905834122	t	8000	#t	Ologn + #t
Sequential Bubble Sort	140	t	8000	t	Logn
Parallel Bubble Sort	-17535222474580	t	8000	#t	Ologn + #t
Sequential Bubble Sort	93	t	8000	t	Logn
Parallel Bubble Sort	-17672137396121	t	8000	#t	Ologn + #t
Sequential Bubble Sort	94	t	8000	t	Logn

10.3 Monte Carlo Pi

Log results from Monte Carlo Pi

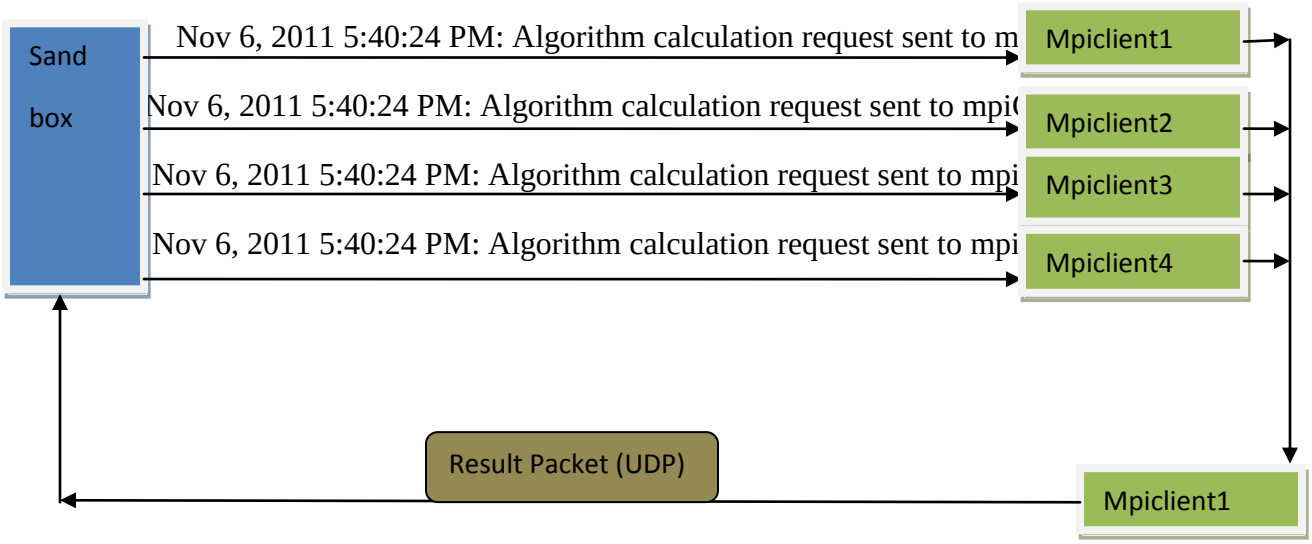
This is the log result of Parallel Monte Carlo Pi. When a user initiates a parallel, Monte Carlo Pi like other algorithms the first things happens is the establishment of inter-processor connection between the processors. Once the connection is established user can send the algorithm request form the sandbox to all the Mpiclients along with an input. Below is the log results of the 500000, 300000, 60000000 points with Parallel and sequential Monte Carlo Pi sort.

Nov 6, 2011 5:40:24 PM: Algorithm calculation request sent to mpiClient1

Nov 6, 2011 5:40:24 PM: Algorithm calculation request sent to mpiClient2

Nov 6, 2011 5:40:24 PM: Algorithm calculation request sent to mpiClient3

Nov 6, 2011 5:40:24 PM: Algorithm calculation request sent to mpiClient4



Above the four lines and the figure explains that the calculation request from sandbox to other clients is sent.

Nov 6, 2011 5:40:24 PM

Packet received from 50.17.18.89:12345

These lines are to tell the user that Mpiclient1 has sent the result back to the sandbox.

The IP address (50.17.18.89:12345) is the IP address of Mpiclient1.

Frame Source Address: 1D: 1D: 1D: 1D: 1D: 4A

Frame Packet Type: 0x303

The line above in the log result is the Mac address of Mpiclient1 and following row is the type of frame message. Every algorithm has three digit integer, which tells users, and servers what kind

of algorithm request is received. Here the message type is 303 it means that this calculation is for parallel Monte Carlo Pi.

Ethernet Frame Message: 3.1414165

Further, in the log results there is result in the form of Ethernet frame message that the pi that is 3.1418019999999998

Ethernet Frame Message: Time for Monte Carlo PI: 531 milliseconds. 0.0seconds

Then there is a result for the time taken by Parallel Monte Carlo Pi. The time is in milliseconds. This Monte Carlo Pi was done for 60000000 numbers and still the time calculation is 531 milliseconds. This time also have overheads which n times of communication between the processor it means that the time sandbox is telling us is more than actual time taken by the algorithm

Below is the last part of the log results is the result of sequential Monte Carlo Pi it has same explanation as the Parallel Monte Carlo Pi, the only difference is that Mpiclient1 does the calculation. Other Clients are not invoked, input is sent to Mpiclient1 and it sends the sequential result to the sandbox.

Nov 6, 2011 5:40:32 PM

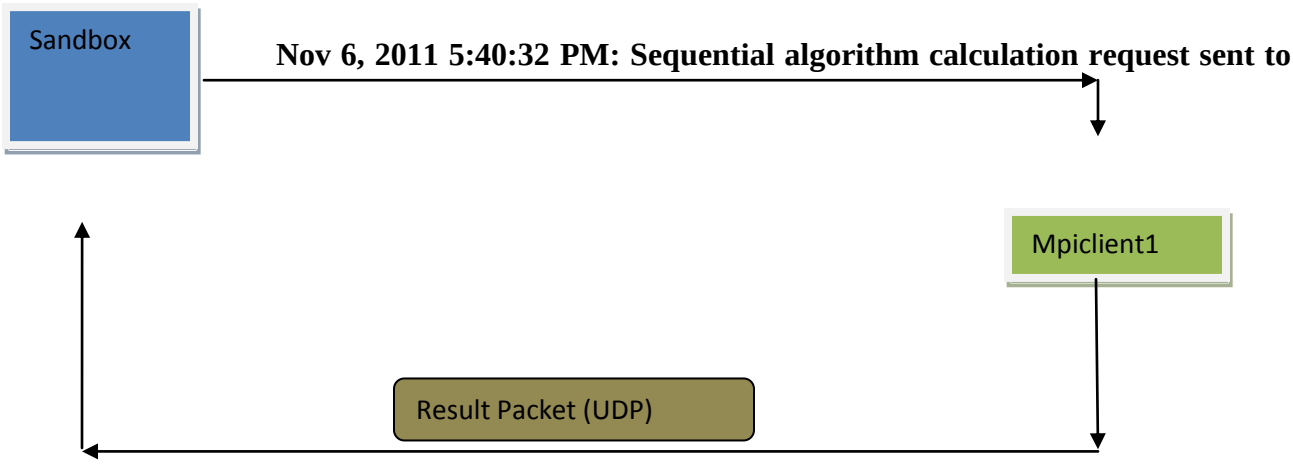
Packet received from 50.17.18.89:12345

Frame Source Address: 1D:1D:1D:1D:1D:4A

Frame Packet Type: 0x303

Ethernet Frame Message: 3.142532

Ethernet Frame Message: Time for Sequential Monte Carlo PI: 468 milliseconds. 0.0seconds



Monte Carlo Pi Results with Four Processor						
Calculation Type	Input (Points)	Time in Milliseconds	Pi	Overheads	Response time	Complexity
Parallel CS	6000000	156	3.1414165	#t	#t	#t +C
Sequential C S	6000000	468	3.1459125	t	T	t+c
Parallel CS	60000000	1326	3.1418019999999998	#t	#t	#t +C
Sequential C S	60000000	4789	3.141476133333333	t	t	t+c
Parallel CS	5000000	140	3.1422832	#t	#t	#t +C
Sequential C S	5000000	390	3.1415456	t	t	t+c
Parallel CA	6000000	531	3.1421493333333332	#t	#t	#t +C
Sequential C A	6000000	486	3.1416773333333334	t	t	t+c
Parallel	5000000	437	3.1409752	#t	#t	#t +C

CA						
Sequential C A	5000000	390	3.1413544	t	t	t+c
Parallel C A	60000000	51664	3.141528700000000 3	#t	#t	#t +C
Sequential C A	60000000	4742	3.14101133333333	t	t	t+c

In the figure # stands number of interaction between the instances. T is the response time form the server, this is the time taken by the clients to respond to Sandbox or other clients

In the figure CA stands for Calculation for Accuracy and CS stands for Calculation for Speed

10.4 Standard Pi

Log results from Standard Pi

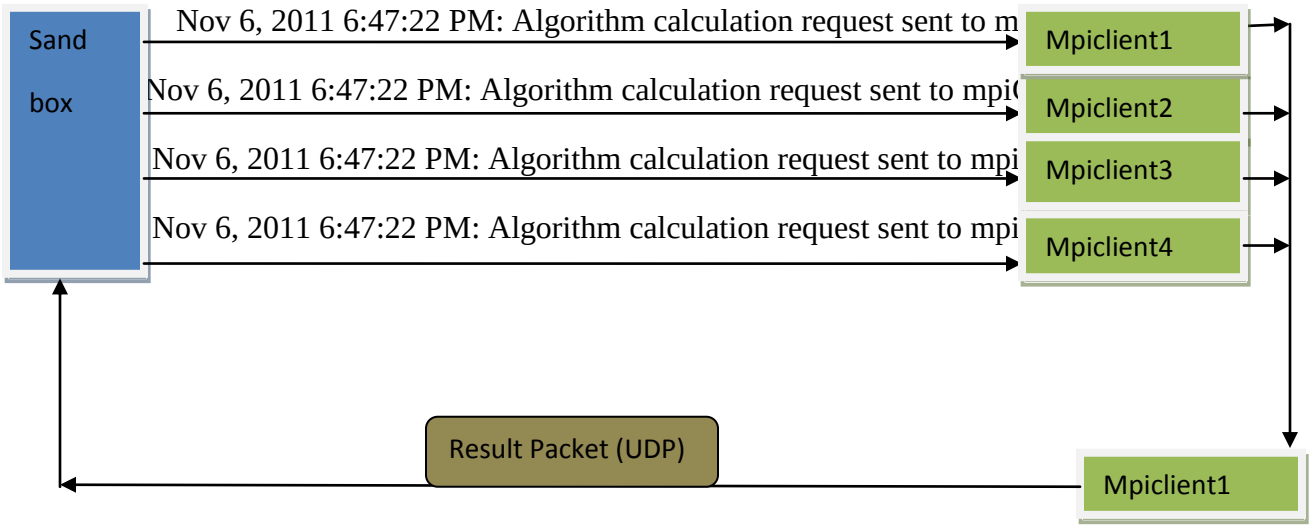
This is the log result of Parallel Monte Carlo Pi. When a user initiates a parallel, Standard Pi like other algorithms the first things happens is the establishment of inter-processor connection between the processors. Once the connection is established user can send the algorithm request form the sandbox to all the Mpiclients along with an input. Below is the log results of the 500000, 300000, 60000000 points with Parallel and sequential Standard Pi.

Nov6, 2011 6:47:22 PM: Algorithm calculation request sent to mpiClient1

Nov 6, 2011 6:47:22 PM: Algorithm calculation request sent to mpiClient2

Nov 6, 2011 6:47:22 PM: Algorithm calculation request sent to mpiClient3

Nov 6, 2011 6:47:22 PM: Algorithm calculation request sent to mpiClient4



Nov 6, 2011 6:47:22 PM

Packet received from 50.17.18.89:12345

The IP address (50.17.18.89:12345) is the IP address of Mpiclient1.

Frame Source Address: 1D:1D:1D:1D:1D:4A

Frame Packet Type: 0x403

The sixth row in the log result is the Mac address of Mpiclient1 and following next row explains about the type of frame message. Every algorithm has three digit integer, which tells users, and servers what kind of algorithm request is received. Here the message type is 403 it means that this calculation is for parallel Standard Pi.

Ethernet Frame Message: 3.1415926535975345

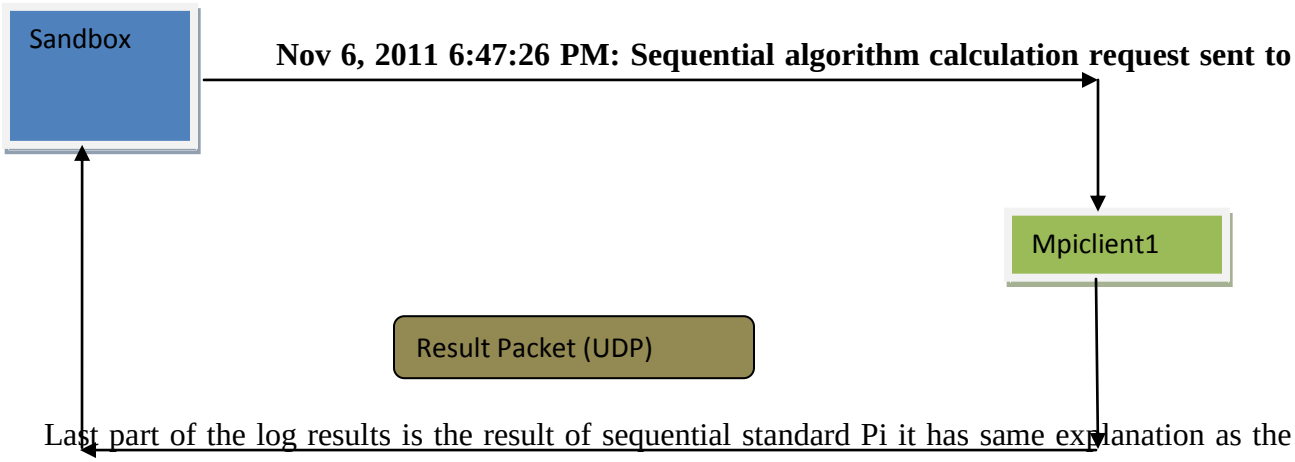
Further, in the log results there is result in the form of Ethernet frame message that the pi that is 3.14159265360606387

Nov 6, 2011 6:47:22 PM

Frame Packet Type: 0x403

Ethernet Frame Message: Time for Standard PI: 390 milliseconds. 0.0seconds

Then there is a result for the time taken by Parallel standard Pi. The time is in milliseconds. This Monte Carlo Pi was done for 30000000 points and the time to complete the calculation is 390 milliseconds. This time also have overheads which n times of communication between the processor it means that the time sandbox is telling us is more than actual time taken by the algorithm.



Last part of the log results is the result of sequential standard Pi it has same explanation as the Parallel Standard Pi, the only difference is that Mpiclient1 does the calculation. Other Clients are not invoked, input is sent to Mpiclient1 and it sends the sequential result to the sandbox.

Standard Pi Results with Four Processor						
Calculation Type	Input (Points)	Time in Milliseconds	Pi	Overheads	Response time	Complexity
Parallel CS	3000000	31	3.1415926541200103	#t	#t	#t +C
Sequential CS	3000000	62	3.14592653656143	t	t	t+c
Parallel CS	30000000	234	3.14159265360606387	#t	#t	#t +C

Sequential C S	3000000	624	3.141592653592723 7	t	t	t+c
Parallel CS	500000	16	3.144159265456378 3	#t	#t	#t +C
Sequential C S	500000	15	3.141592654563783	t	t	t+c
Parallel CA	3000000	93	3.141592653656143	#t	#t	#t +C
Sequential C A	3000000	623	3.141532653656435	t	t	t+c
Parallel CA	3000000	749	3.141592653592783 7	#t	#t	#t +C
Sequential C A	500000	16	3.141592655456387 2	t	t	t+c
Parallel C A	500000	47	3.141592655456387 21	#t	#t	#t +C
				t	t	t+c

In the figure CA stands for Calculation for Accuracy and CS stands for Calculation for Speed

11. Graphical Representation of Results

This section explains the results of the sandbox algorithm with the help of graph. Graphs are plotted for time and rounds of input for each algorithm. These graphs also explain about the overheads of the algorithms. Overheads occurs when the instances setup packets, creates input and output data streams to receives and send the packets

11.1 Merge Sort algorithm with four processors

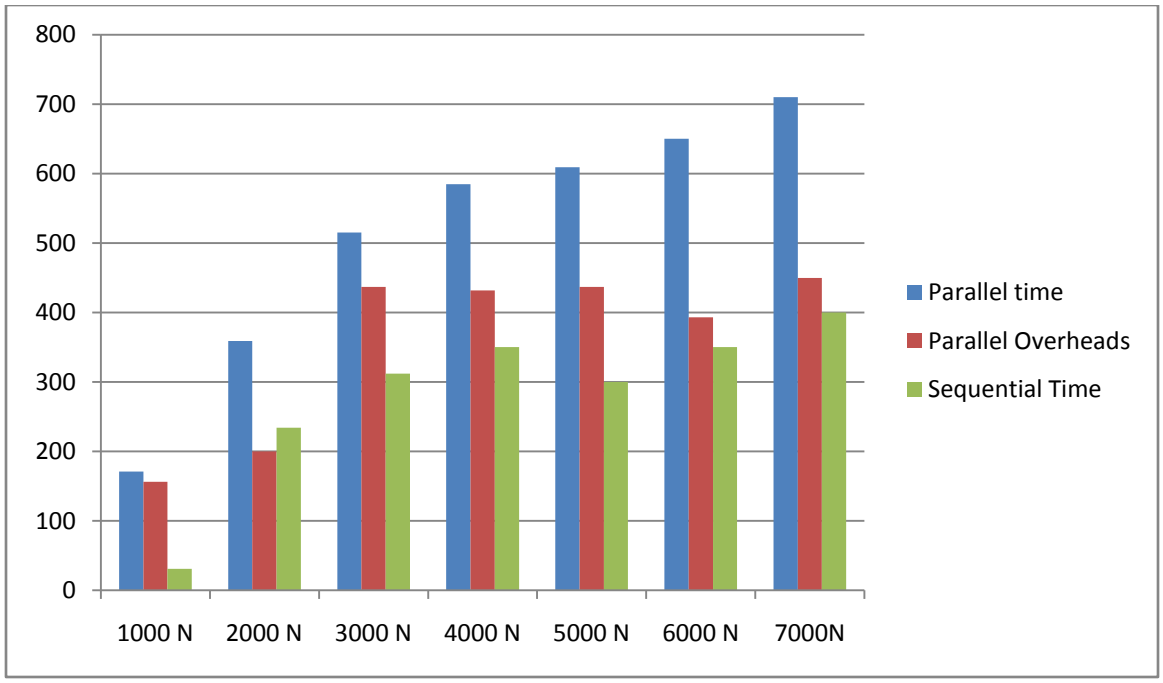
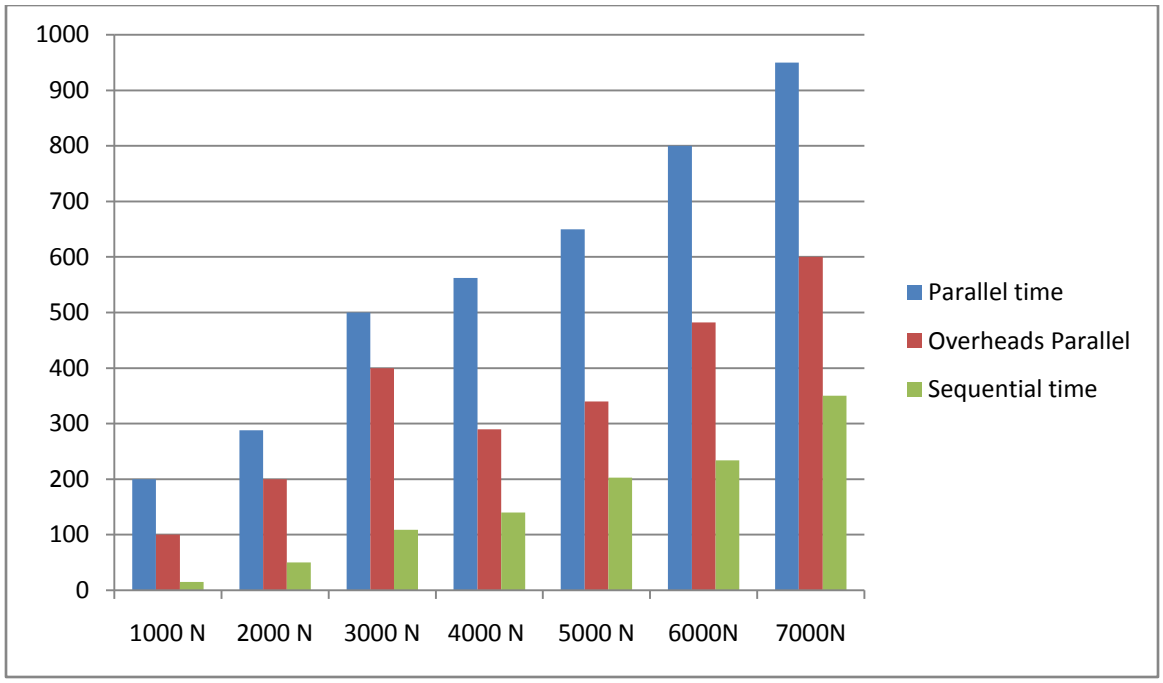


Figure 10 : Parallel and Sequential Merge Sort Graph

Figure 10 explains the sequential and parallel merge sort algorithm. Time for each rounds of input is in milliseconds and is given in the left most column of the graph. First three columns have 1000 random number for both sequential and parallel merge sort. Red color column has a time for the overhead occurs for parallel execution of the merge sort. Next vertical columns have high random numbers as an input to merge sort algorithm.

11.2 Bubble Sort Algorithm with four Processors

Figure 11 for the bubble is plotted the way graph was plotted for the merge sort for time and rounds of input for both sequential and parallel bubble sort. The graph explains about the overheads of the algorithms. Overheads occurs when the instances setup packets, creates input and output data streams to receives and send the packets



Figures 11 : Parallel and Sequential Bubble Sort

Figure 11 explains the sequential and parallel bubble sort algorithm. Time for each rounds of input is in milliseconds and is given in the left most column of the graph. First three columns have 1000 random number for both sequential and parallel bubble sort. Red color column has a time for the overhead that occurs for parallel execution of the bubble sort. Next vertical columns have high random numbers as an input to bubble sort algorithm.

11.3 Monte-Carlo Pi

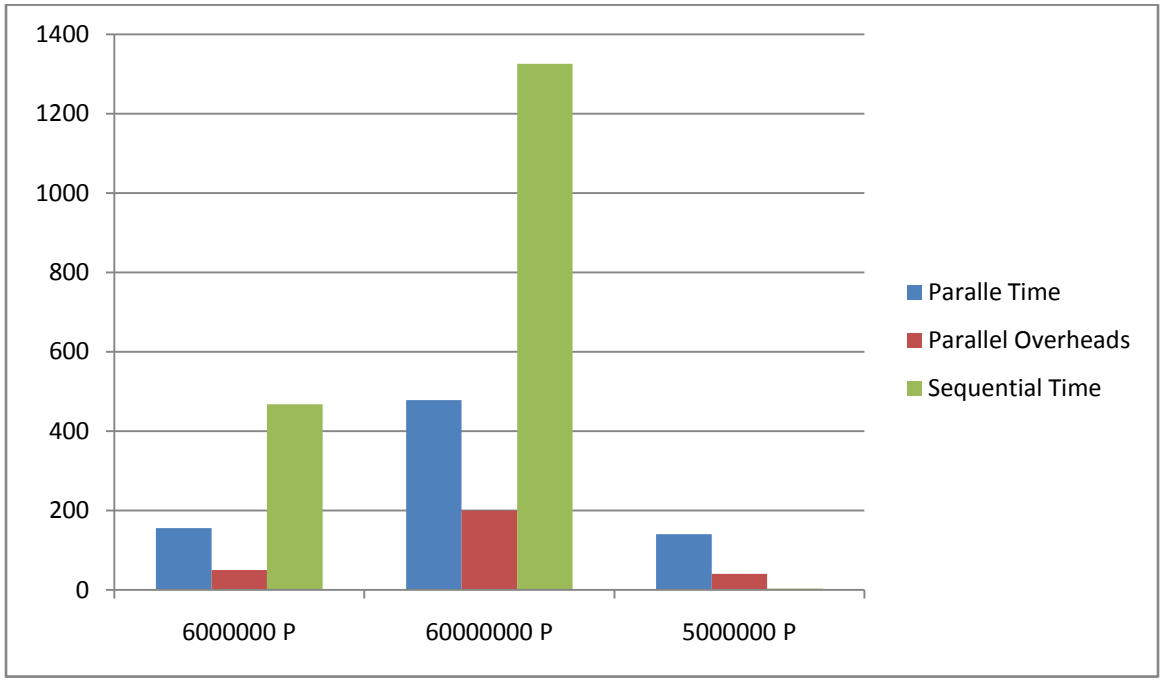


Figure 12 : Monte-Carlo Graph for Speed

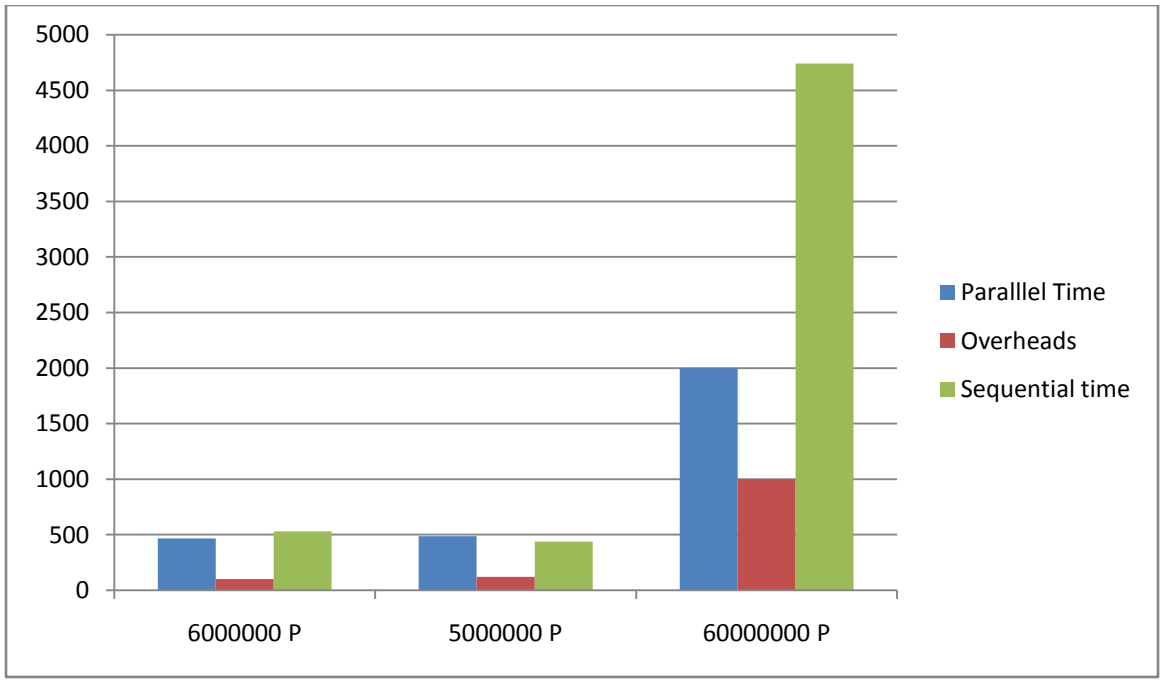


Figure 13 : Monte-Carlo Graph for Accuracy

11.4 Monte-Carlo speed

Figure 12 explains the Monte-Carlo algorithm for speed. Here the time is given in the left most column in Milliseconds. First three bars described the time taken by the algorithm for parallel, total overheads in the parallel execution of the Monte-Carlo algorithm, and last one describe the time for the sequential Monte-Carlo algorithm. The numbers in the bottom of the graph are the number of points in the circle. These numbers are passed as an input which is also the numbers of points for the circle.

11.5 Monte-Carlo Accuracy

Figure 13 explains the Monte-Carlo algorithm for accuracy. Accuracy means that the data type and the number of iteration are different then the speed version, it has a longer pi. Here the time is given in the left most column in Milliseconds. First three bars described the time taken by the algorithm for parallel, total overheads in the parallel execution of the Monte-Carlo algorithm, and last one describe the time for the sequential Monte-Carlo algorithm for accuracy. The numbers in the bottom of the graph are the number of points in the circle. These numbers are passed as an input, which is also the numbers of points for the circle.

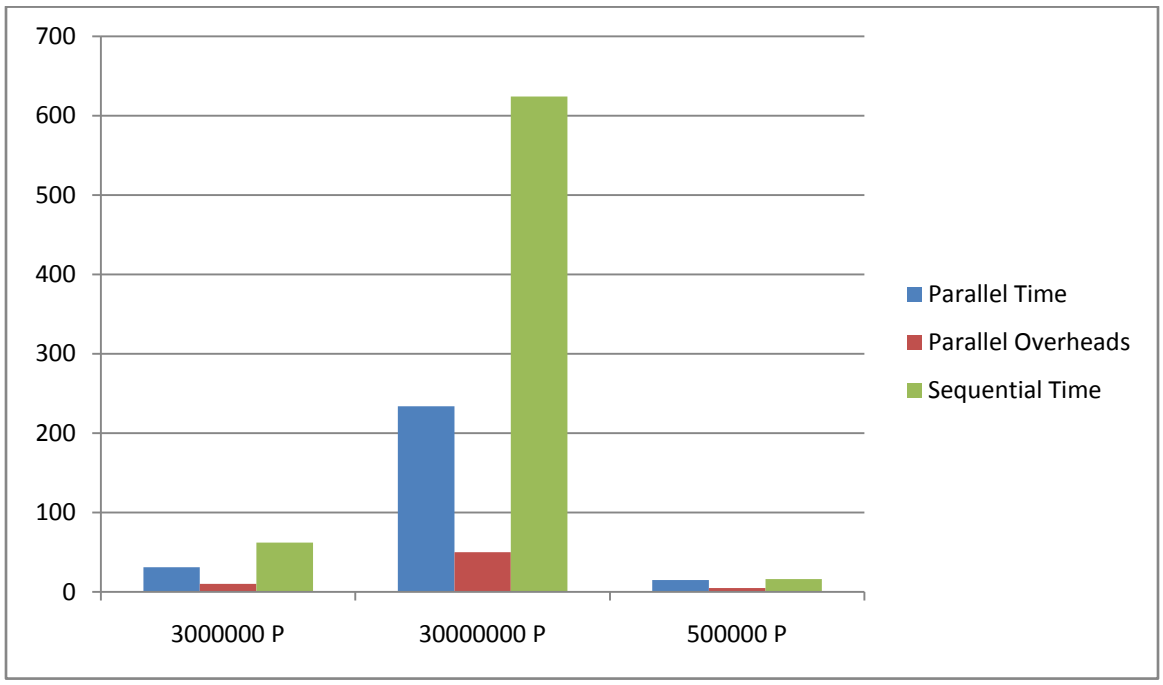


Figure 14: Graph for the Standard Pi for speed

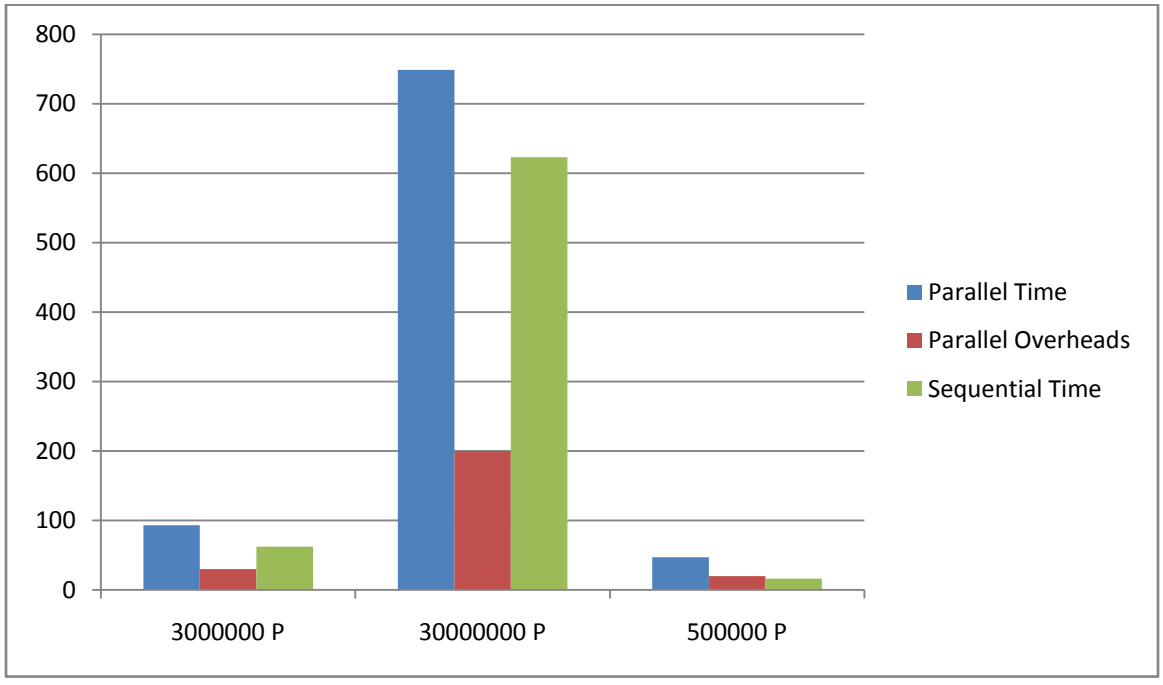


Figure 15 : Graph for the Standard Pi for Accuracy

11.6 Standard Pi Speed

Figure **14** explains the Standard Pi algorithm for speed. Here the time is given in the left most columns in Milliseconds. First three bars described the time taken by the algorithm for parallel overheads in the parallel execution of the Monte-Carlo algorithm, and last one describe the time for the sequential Standard pi algorithm. The numbers in the bottom of the graph are the number of the points in the circle. These numbers are passed as an input which is also the numbers of points for the circle.

11.7 Standard Pi Accuracy

Figure 15 explains the Standard pi algorithm for accuracy. Accuracy means that the data type and the number of iteration are different then the speed version, it has a longer pi. Here the time is given in the left most columns in Milliseconds. First three bars described the time taken by the algorithm for parallel, total overheads in the parallel execution of the Monte-Carlo algorithm, and last one describe the time for the sequential Standard pi algorithm for accuracy. The numbers in the bottom of the graph are the number of points in the circle. These numbers are passed as an input, which is also the numbers of points for the circle.

12. Conclusion

The aim of this project is to provide a Sandbox for the users to get hands on experience with parallel processing. Parallel processing is simply running the algorithms concurrently on several processors, which reduces the computational load of the processor and make the algorithm work effectively for high computation.

Problems with such kind of tool:

- Hardware requirement in order to run parallel programs we need number of processors in the same machine.
- Debugger proper debugger lets the users debug parallel programs.
- Cost effective parallel processing involves expensive requirements.
- Comprehensive analysis of the results.
- Interactive Graphical user interface.

Sandbox address these issues by providing the following functionalities

- Sandbox have interactive Graphical user interface, which is used run the algorithms the algorithms with rounds of input from the algorithm option section of the sandbox.
- No additional hardware is required for making such a tool and to setup the multi-processor environment. Sandbox uses the Amazon instances to make a distributed environment.
- Sandbox has a log of results which is very user friendly and interactive. There is a separate log from all the clients to track the result of algorithms. Each client logs can be requested form the sandbox for every single iteration of the parallel programs.

- If an algorithm breaks at certain iteration or at some client. Sandbox logs and exceptional handling helps the users to analyze why the algorithm break.
- Sandbox can be used with less expenses as the Amazon provides the option to start and stop the instances.

Sandbox is written using Java socket programming and Java swing. It requires only Java Software environment it could be any java IDE. Sandbox is designed with ease of use mind, allowing any user to focus on the core functionality of the sandbox without getting lost in how to use the tool. Sandbox is tested in the machine have single processor machine without using the cloud services.

Future study would be to allow the sandbox to increase the data transfer capacity for the algorithms like image processing. Enhancing the GUI and adding the module that draws the results in graph instead of creating logs in text file.

13. References

1. W.Groop, E. Lusk and A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing
Interface, 2nd edition, MIT Press, Cambridge, MA, 2002
2. N. Karonis, B.Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing
Interface". Journal of Parallel Distributed Computing, 2003
3. MPICH-G2: http://www.hpclab.niu.edu/mpi, 2003, Date 09/28/2010
4. MPI Standard: http://www-unix.mcs.anl.gov/mpi/3. Date 08/01/2010
5. J. Attallah, R. Cole, and M. T. Goodrich. Cascading divide and conquer: A technique for designing parallel algorithms. SIAMJournalOn Computing, 18(3):499–532, 1989.
6. A. Aggarwal, A. K. Chandra, and M. Snir. On communication latency in pram computations. Annual ACM Symposium on Parallel Algorithms and Architectures, pages 11–22, 1989.
7. J. L. Bentley. Multi-dimensional divide-and-conquer. Communications of the ACM, 23(4):214–229, 1980.
6. J. L. Bentley. Tools for experiments on algorithms. In R. F. Rashid, editor, CMU Computer Science: A 25th AnniversaryCommemorative, chapter 5.ACMPress, 1991.
8. P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.
9. G. Blelloch. Scans as primitive parallel operators. IEEE Transactions On Computers, 38(11):1526–1538, 1989.
10. G. Blelloch. Vector Models for Data-Parallel Computing. MIT Press, 1990.
11. G. E. Blelloch. Nesl: A nested data-parallel language (version 2.6). Technical Report CMU-CS-93-129, CMU, 1993.
12. G. Blelloch, C. Leiserson, B. Maggs, G. Plaxton, S. Smith, and M. Zagha. A comparison of sorting algorithms for the connection machine cm-2. Annual ACM Symposium on Parallel Algorithms and Architectures, 1991.
13. S. Chatterjee. Compiling Data-parallel Programs for Efficient Execution on Shared-Memory Multiprocessors. PhD thesis, School of Computer Science, CMU, 1991.
14. Java Message Passing Interface

15. Aerospace and Electronics Conference, 1997. NAECON 1997., Proceedings of the IEEE

1997 National

16. Towards Efficient Shared Memory Communications in MPJ Express IEEE 2004

17. http://docstore.mik.ua/orelly/java-ent/dist/ch06_01.htm, Date 10/15/2011