International Journal of Computer Applications (0975 – 8887)
Volume 19– No.9, April 2011

Multi-Client Multi-Instance Secured Distributed File Server

Arun Singh, Ajay K Sharma, Ashish Kumar
Dr. B.R Ambedkar
NIT Jalandhar, India

ABSTRACT
This paper describes a Distributed File Server, implemented in Java Sockets, based on TCP protocol. The server responds file request of multiple formats like txt, doc, pdf, jpeg, mp3, mp4, flv etc to multiple clients at a time. The requested server first checks the requested file, with it. If file search is not successful then it turns toward other servers connected to it.

The Dedicated Server performs client authentication based on Ipaddress of connecting client and maintains list of files present with other servers connected with it. The clients at it end perform check on user requests, for better server performance. The system proposed is well tested in our laboratory with 0, 2, 4, and 8 clients connecting in distributed environment. Results of heap memory utilization and relative server utilization time in serving varying number of clients is drawn and analyzed.

General Terms
Distributed System and Computer Networks.

Keywords
Java Server, File Server, Java Socket, TCP Server.

1. INTRODUCTION
TCP and UDP are two main networking protocols that can be employed in distributed applications. TCP guarantees that once a connection has been established between two parties, data sent from one side to another will be in proper order without any loss; this is because of acknowledgment and ordering of each data packet. While UDP does not guarantee this, because data is divided into data packets called datagram's, whose delivery is not acknowledged, also at other end ordering of datagram's is not performed [1].

In this paper we proposed and implemented Distributed Server using TCP protocol, called MMSDFS (MultiClient MultiInstance Secured Distributed File Server). The proposed system is used to transfer multi format files from server to client including text, document and pdf files etc, and the data loss in which could not be tolerated. Integrity of data received in TCP protocol is achieved at a cost efficiency and performance.

2. RELATED WORK
Java Sockets, RMI (Remote Method Invocation) of Sun Microsystems or CORBA (Common Object Requests Broker Architecture) of OMG (Object Management Group), are well known used for distributed system development. RMI, one of the core Java APIs since version 1.1., is a way that Java programmers can write object-oriented programs in which objects on different computers can interact with each other. A client can call a remote object in a server, and the server can also be a client of other remote objects. CORBA is well suited for Distributed system working on different platforms with varying specifications and environment [2, 3]. But the implementation of RMI and CORBA are very complex and specialized task. Hence, Java Socket is better choice for easy implementation.

A P2P network is a network of computers or nodes where there is a concept of equality that is anyone could act like a client if required a data and the one who serves act like a server, it is different from traditional client/server architecture where only dedicated system could act like a server and requesting computers would acts like client to it. If one of the client have the data required by other client, in any case it could not serve him, this become a major drawback in traditional client/server architecture. Since all the nodes are equal in a P2P network, there is no requirement of dedicated server, as each node can act like both client/server and free to add or leave the network any time, without disturbing others peers.

The P2P network architecture is mainly grouped into two categories: centralized systems (Napster, BitTorrent etc.) and decentralized systems (Gnutella, FreeNet etc.)[4].

It is found that P2P file sharing accounts for much more traffic than any other application on the Internet, since transport of data take place on Internet hence unsecure without encryption and vulnerable for Intruders attack [5].

3. DESIGN AND IMPLEMENTATION OF MMSDFS
Distributed client/server architecture, with single server connected to other servers listed in its connecting data file, by default it's three. Figure 1 shows the architecture view of MMSDFS. The server composed of functionality of creating multiple instances for each client, as

```
public class ts implements Runnable
{
Socket S;
ts(Socket S)
{
this.S=S;
}
public static void main(String aa[])
{
ServerSocket SS = new ServerSocket(8000);
While(true){
Socket S1 = SS.accept();
new Thread(new ts(S1)).start();
..
}}}.
```

After accepting the client connection [6], the server accesses the client Ipaddress and checks it with the list of authenticated clients with it. If the client Ipaddress present in its list then it allows authenticated client to access the data file otherwise close the connection at the same time without any transfer. The required code is as

```
DataInputStream dis0=new DataInputStream (new
FileInputStream ("auth.txt"));
//S1 Socket Object
while((str0=dis0.readLine())!=null)
{
if(str0.equals((S1.getInetAddress()).toString()))
{
new Thread(new ts(S1)).start();
flag0=true;
break;
}}
if(!flag0)
{
S1.close();
//Close connection for unauthorized user
}
```

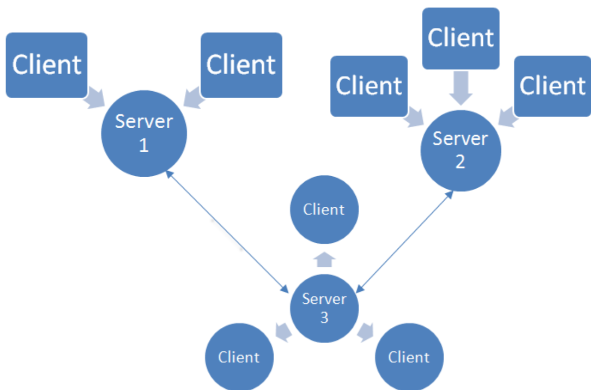

Figure 1. The Architecture view of MMSDFS

The server Administrator is provided the functionality of looking the file present at server repository as

```
DataOutputStream dos0=new DataOutputStream(new
FileOutputStream("d:\\filename.txt"));
File folder=new File("d:\\shared");
/* 'shared' folder content is shared in distributed environment*/
File[] lof=folder.listFiles();
for(int i=0;i<lof.length;i++)
{
if(lof[i].isFile())
{
System.out.println("File : "+lof[i].getName());
dos0.writeBytes((lof[i].getName())+"\r\n");
}
}
dos0.close();
```

Each server along with maintaining list of clients, who can connect with it, maintains list of files present with its adjacent servers. This list of files got updated each time server in the neighborhood got startup. If the file requested by client is not present with the dedicated server then instead of connecting to all other servers in neighborhood, the dedicated server looks for file name in its file list. If file is not present with adjacent servers also then client will be responded with message 'no'. If file is founded with any of the adjacent server, the dedicated server sends file request to it and copy file on it, for future use and send copy of file to client [7, 8]. The waiting distributed servers looks like as in Figure 2. The server connecting to other servers is handled by Java sockets [9] as,

```
DataInputStream dis=new
DataInputStream(S.getInputStream());
DataOutputStream dos = new
DataOutputStream(S.getOutputStream());
boolean flag=false;
name=dis.readUTF();
File folder=new File("d:\\b");
File[] lof=folder.listFiles();
for(int i=0;i<lof.length;i++)
{
if(lof[i].isFile())
{
if(name.equals(lof[i].getName()))
{
filename=name;
dos.writeUTF(filename);
flag=true;
break;
}
}
}
if(flag==false)
{
Socket S1 =new Socket("10.10.54.180",8001);
DataOutputStream dos1=new
DataOutputStream(S1.getOutputStream());
dos1.writeUTF(name);
DataInputStream dis1 = new
DataInputStream(S1.getInputStream());
filename=dis1.readUTF();

if((name.equals(filename)))
{
boolean b=false;
}}
```

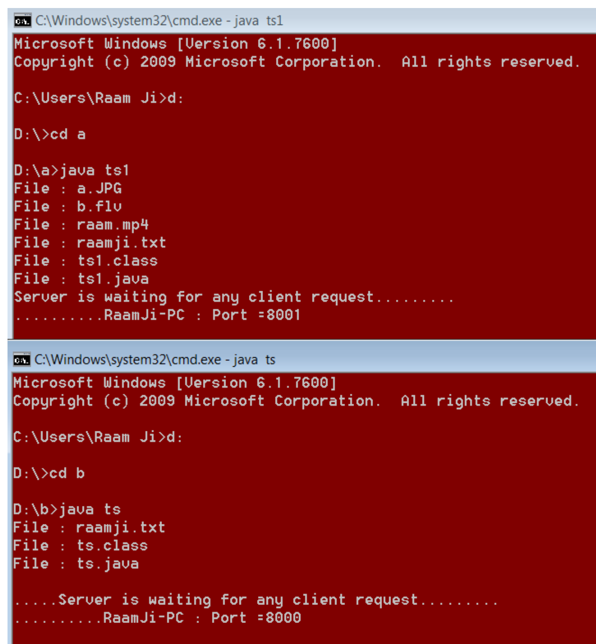

Figure 2. The Distributed Servers in Waiting State

The client is provided with the functionality of authenticating user input at its end, by checking the connecting server Ipaddress at specified port. The client filter is shown in Figure 3. The client is dynamically connected to default server [9], if required Ipaddress and port are not correct or does not exist. At a time client could connect to only one server, but fetch data from multiple servers, as shown in Figure 4. Each intermediatory server will make a copy of requested data with

it, as shown in Figure 5. There is an additional functionality of requesting file efficiently, by categorizing the request on the basis of type of file required by user. This categorization is much more efficient for video files and support multiple video formats, the code for such categorized file transfer is given below

Client code for requesting txt, doc, pdf, jpeg, mp3 etc. is as,

```
BufferedOutputStream bos = new BufferedOutputStream(fos);
try {
num_bytes_read = in.read(buf,0,buf.length);
current=num_bytes_read;
do{
num_bytes_read = in.read(buf, current,(buf.length - current));
if(num_bytes_read>=0)current += num_bytes_read;
}while(num_bytes_read>-1);
bos.write(buf, 0,current);
bos.flush();
bos.close();
}

catch (IOException e)
{
e.printStackTrace();
}
```

Client code for requesting video file such as mp4, flv etc. is as,

```
BufferedOutputStream bos = new BufferedOutputStream(fos);
try {
num_bytes_read=64;
do{
num_bytes_read=in.read(buf,current,num_bytes_read);
bos.write(buf, current, num_bytes_read);
current=current + num_bytes_read;
}while(num_bytes_read!=0);
bos.flush();
bos.close();
}
catch (IOException e) {
e.printStackTrace(); }
```

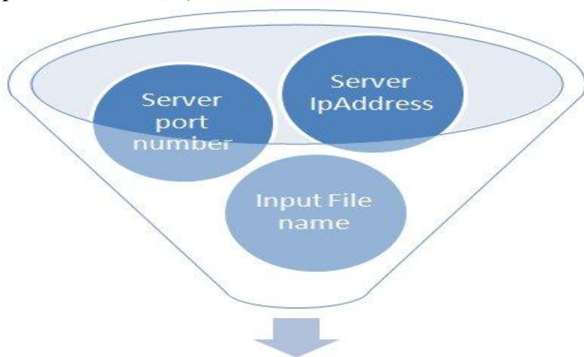

Figure 3. The Client filtering process

```
D:\theesisss java\4mar con code>java tc
------------------------------------------------------------------

 Default Host=localhost & port=8000, to act as an Server

 To change Host or Port press  1
2

 ------ Connectint to Host : localhost  at port number : 8000


  Enter 1 to copy Text File, DOC File , PDF , Image ,or Audio File
  Enter 2 to copy Video File
  Enter 3 to check Status of Server
  Enter 4 to Monitor Itself
2
 ------->Selected Choice is : 2

  Enter name of file required by you, with proper extension
b.flv
```

Figure 4. The Client Instance in Requesting State

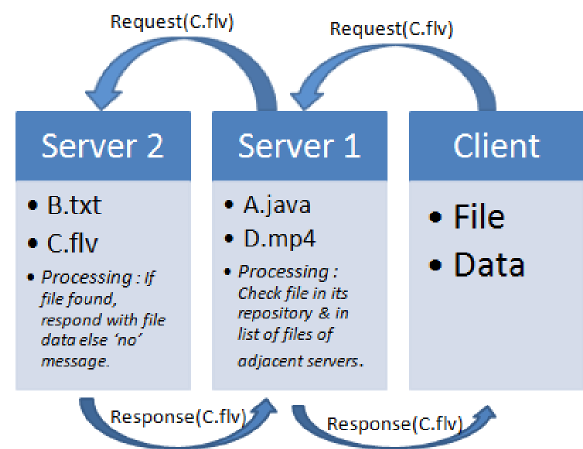

Figure 5. Client Request manipulation in MMSDFS

4. PERFORMANCE EVALUATION

The implemented server is tested on Systems with Intel(R) Pentium Processor(R) D CPU 2.80 GHz 2.79 GHz, 1.25 GB Ram, Windows 7 (32 bit) Operating System.

The distributed servers composed of multiple files in multiple formats meant for sharing over network. Each server composed of dedicated folder which is shared between all servers. Each server is provided with permission of only reading a file from other server. The data files used for testing is mentioned in Table1.

Table 1. The Server Repository

File Extension	File size
Txt	822 Bytes
Java	1.99 KB
Docx	13.0 KB
PDF	52.0 KB
HTM	19.4 KB
MP3	6.80 MB
MP4	10.1 MB
FLV	39.6 MB

Server is tested for 0, 2, 4, and 8 numbers of clients, with multiple requests in mixed format. The Heap memory utilized by CPU running the dedicated server for responding to different number of clients for different period of time is shown in Figure 6, 7, 8 and 9.

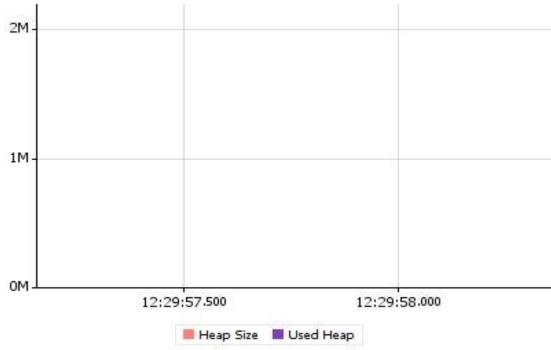

Figure 6. Server Connected to 0 Clients

Figure 7. Server Connected to 2 Clients

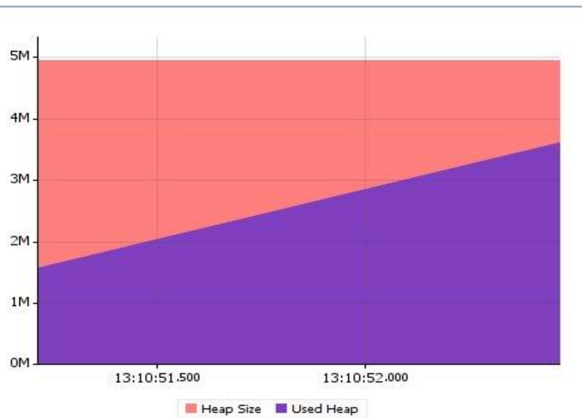

Figure 8. Server Connected to 4 Clients

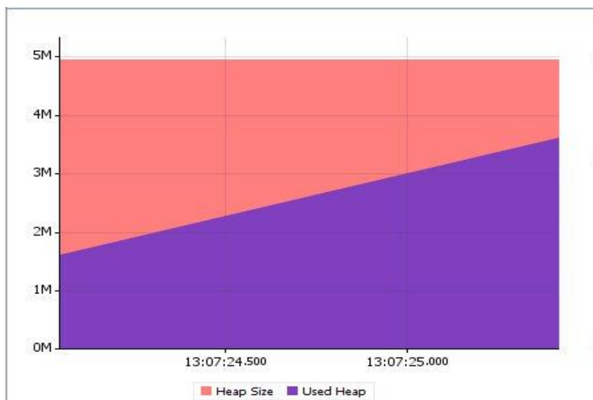

Figure 9. Server Connected to 8 Clients

The figure 6, 7, 8 and 9 shows that for mixed format file transfer, the heap memory utilization does not exceed 4M in any case and after the file transfer the memory consumption becomes almost constant. The figure 10, 11, 12 and 13 specially shows the interaction time of clients with the server for file transfer that is serving time versus time utilized for garbage collection.

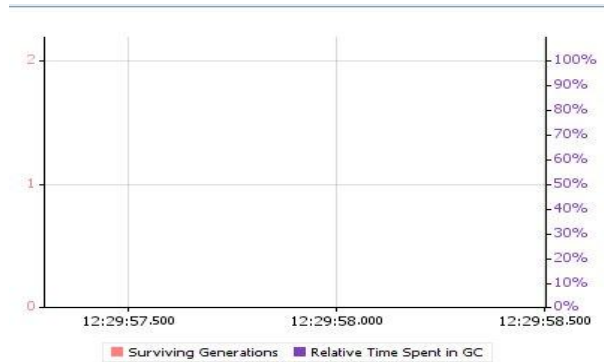

Figure 10. Server Connected to 0 Clients

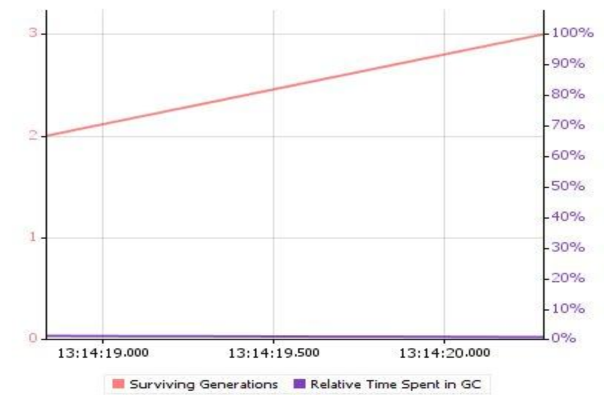

Figure 11. Server Connected to 2 Clients

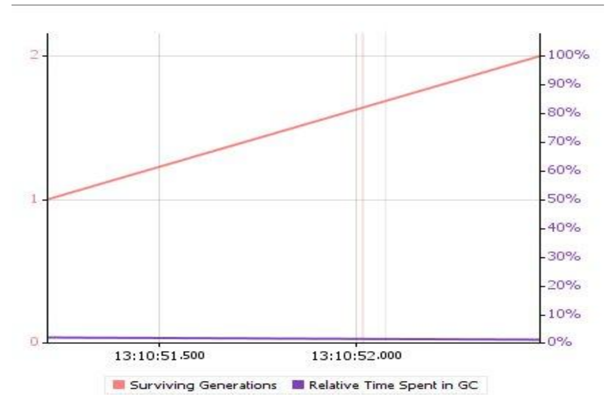

Figure 12. Server Connected to 4 Clients

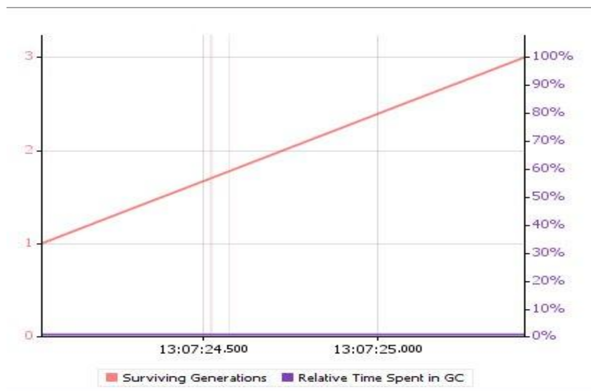

Figure 13. Server Connected to 8 Clients

5. CONCLUSION AND FUTURE WORK

This article puts forward a kind of MultiClient MultiInstance Distributed Java Socket Server, design and implemented for limited number of users. The client/server system proposed is effectively deployed in our laboratory, with three distributed servers connecting with each other only at a time of file transfer. The system is tested for selected multimedia files, so Heap Memory utilization does not go beyond 4M but Server Utilization time decreases with increase in number of clients, due to type of request made by client and response is given by single server or multiple servers. Transfer of data is implemented for authenticated users identified by server administrator prior to connection with the server in personal network so system is well secured but static.

The work is going on to increase server performance by distributing file request filtration at client level and increasing data security during transmission.

6. REFERENCES

[1] Eitan Farchi, Yoel Krasny, Yarden Nir, "Automatic Simulation of Network Problems in UDP-Based Java Programs", proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004 IEEE.

[2] Delin Hou and Huosong Xia,"Design of Distributed Architecture based on Java Remote Method Invocation Technology", pp. 618, 2009 International Conference on Environmental Science and Information Application Technology (IEEE Computer Society 2009).

[3] V.Getov, G. von Laszewski, M. Philippsen, I. Foster," Multi-paradigm Communications in Java for Grid Computing", Communications of the ACM, Vol. 44, No. 10, pp. 118-125, 2001.

[4] Amol Vasudeva, Sandeepan, Nitin Kumar," PASE: P2P Network Based Academic Search and File Sharing Application", First International Conference on Computational Intelligence, Communication Systems and Networks in 2010.

[5] Yao-Nan Lien and Hong-Qi Xu, " A UDP Based Protocol for Distributed P2P File Sharing", Eight International Symposium on Autonomous Decentralized System in 2007.

[6] Ming Xue and Changiun Zhu, "The Socket Programming and Software Design for Communication Based on Client/Server", Pacific-Asia Conference on Circuits, Communications and System in 2009.

[7] What is peer-to-peer? Peer-to-Peer working group http://www.p2pwg.org/

[8] J2SE 1.5.0 API Specification, available at http://java.sun.com.

[9] Java Socket Tutorial, available at http://www.cs.swan.ac.uk/~csneal/InternetComputing/JavaSockets.html.