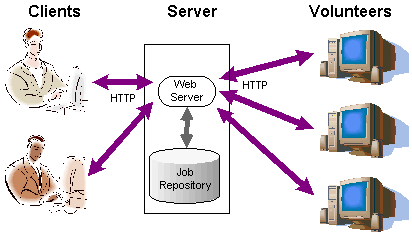
Assignment 3

Protocol Design for Distributed Applications

Introduction

This report will describe our teams attempt to develop an implementation of a volunteer computing platform. Volunteer computing is a type of distributed computing in which computer owners (‘Volunteers’) donate their computing resources (such as processing power and storage) to one or more "projects".



The implementation allowed a client to send to a server a name that they wished found amongst a dataset of names. The server would then divide the dataset amongst the worker nodes, which were used to represent the volunteers, and allowed them to process the data and search for the name. If the dataset contained its name then its index would be returned to the client. This type of process is known as data mining, in which data is extracted from a data set and convert it into a structure that is understandable to a user for further use.

High-level Description of The Solution

Server

The Server class acts as a coordinator for the worker nodes, ensuring that they are always processing a part of the file and recording their progress.

Whenever the Server receives a packet, its OnRecipet() method is called. If a SendName packet is received from the client, it obtains the name to be searched from the packet, and sets a Boolean variable to true, which then allows the method OrganiseFile() to begin.

**public** **synchronized** **void** onReceipt(DatagramPacket packet)

{

PacketContent received = PacketContent.*fromDatagramPacket*(packet);

**if**(received == SendName)

{

currentSearch = received.information;

startWork = **true**;

}

The OrganiseFile() method then goes through each name in the file to be searched and loads them into a two-dimensional array of Strings, so the contents of the file are divided into 20 sections.

**public** **void** organiseFile()

{

String fname= "names.txt";

String line = " ";

**long** counter;

File file;

FileInputStream fin;

BufferedReader in;

**try** {

file= **new** File(fname);

fin= **new** FileInputStream(file);

in= **new** BufferedReader(**new** InputStreamReader(fin));

FileContents = **new** String[20][*DIVISION*];

counter= 0;

**int** array = 0;

Stats = **new** **int**[5][2];

**while**(line != **null** && array < 20)

{

**int** index = 0;

counter=0;

**while**(counter < *DIVISION* && line != **null**)

{

line= in.readLine();

FileContents[array][index] =line;

index++;

counter++;

}

array++;

}

}

**catch**(Exception e) {e.printStackTrace();}

}

If the Server receives a Register packet from a Worker Node, it calls the isMissing() method, which\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. If the Server has not gone through every section of the file and has not stopped searching, it then calls the NextSection() method, which returns the next section of the code to be searched, and then the SendWork() method that sends the section to the Worker Node.

**else** **if**(received.type == Reigister)

{

**int** node = ((Register)received).nodeNumber();

isMissing(packet, node);

**if**(NextSection < 20 && startWork)

{

getNextSection();

timeTaken += ((Register)recieved).time();

sendWork(packet.getSocketAddress(), FileContents[NextSection]);

heartbeats[node].timerTask();

}

}

A Timer is then set so that if no Heartbeat(an update from the node) is received within the time limit of Two timers, the Server assumes the node is not processing any data and sends it another section. If it does receive a Heartbeat, then it stores the number of indexes that particular node has processed.

**else** **if**(recieved.type == *HEARTBEAT*)

{

**int** node = ((Heartbeat)recieved).number();

**int** index = ((Heartbeat)recieved).index();

Stats[node][0] = Stats[node][0] + index;

Stats[node][1] = ((Heartbeat)recieved).sectionsprocessed;

heartbeats[node].timerTask();

}

If the Server receives a ResultPacket, it sends a StopWork Packet to each of the server nodes so that they stop searching for the name. The Server then sends a ResultPacket to the client, which dispays the index of the name in the file if it was found or otherwise it states the name was not found.

**else** **if**(recieved.type == *RESULTPACKET*)

{

**int** node = ((ResultPacket)recieved).getID();

endWork(packet, node);

}

}

Heartbeat

The Heartbeat class is an extension of the PacketContent class, and is sent from the Worker Nodes to the Server to represent acknowledgements that they are still working and to update the Server on their progress. Each Heartbeat contains information on the Worker Node that sent it.

It contains the methods Heartbeat, which is used to define a new Heartbeat packet, toObjectOutputStream, which is used when converting a Heartbeat to a DatagramPacket, active() which returns whether or not the Worker Node is processing data, index() which returns the current index the Worker Node was on when it sent the Heartbeat, sections() which returns the number of sections of the file the Worker Node has processed and number() which returns the number of the Worker Node.

**public** **class** Heartbeat **extends** PacketContent

{

**boolean** active;

**int** currentindex;

**int** sectionsprocessed;

**int** NodeNumber;

Heartbeat(**boolean** active, **int** currentindex, **int** sectionprocessed,

**int** NodeNumber)

{

type= *HEARTBEAT*;

**this**.active = active;

**this**.currentindex = currentindex;

**this**.sectionsprocessed = sectionprocessed;

**this**.NodeNumber = NodeNumber;

}

**public** **boolean** active()

{

**return** active;

}

**public** **int** index()

{

**return** currentindex;

}

**public** **int** sections()

{

**return** sectionsprocessed;

}

**public** **int** number()

{

**return** NodeNumber;

}

5. Reflection & Conclusions: This section should present your thoughts on the outcome of your assignment ie. what worked for you, what didn't, and what could be improve