Title Unknown

Annapurna Valluri

1. Introduction

There are a number of situations, one comes across in one's life, in which one has to find the k nearest neighbors of an object, be it a location on a map, an image or even a number. Finding the k nearest neighbors of an object can be done as easily as sorting those very objects, once you have defined the method of calculating the distance between the objects. In some cases, it simply involves calculating the distance using coordinate geometry and in others, it is more complicated than that.

The world wide web is becoming an essential component in every organization. Nowadays, there is not a single company, organization, or TV channel that does not have a web page. An important key to success for a company, university, or even an individual, is keeping up with the latest technology. A lot of software can be downloaded from the web. This saves time and money since the available software is already tested for errors. But wouldn't it be nice if you could simply get your information from a site without having to download the code?

2. Main function of the Project

As you must have already guessed, this project deals with inserting and finding the k nearest neighbors of an object, in this case being a stock. The distance is calculated on the basis of the closing prices of the stocks. A number of databases can be created in which you can insert a number of files with closing stock quotes in them. The distance is calculated using the famous Euclidean distance between 2 vectors. The vectors, in turn, are calculated using the Daubechies-4-wavelet transform.

One of the important features of this project is that it can be run from the web. It was written in JAVA and is run from the web as an applet. There are a number of reasons for choosing JAVA as the programming language. An important reason is that Java is an object oriented programming language. Now comes the question, "Why not use C++?" C++ is not easily portable unlike Java, in which the same code can be run on Windows 95,

Solaris, Unix, and Macintosh. And the main reason being that Java has a number of routines that make it relatively easy to communicate with TCP/IP protocols like HTTP and FTP, which other programming languages lack.

3. Applet Security

Since applets are designed to be loaded from a remote site but executed locally, security becomes an important issue. Security of files restricts the functionality of applets. Applets cannot run any local executable program. Applets can only communicate with the server from which they were downloaded. When applets are run from Netscape, they cannot read or write to the local computer's file system. Although, if you use other browers it may be possible to do so because this is not a part of the Java specification. Applets cannot find out any information from the local computer.

4. Client-Server Architecture

Java is an extremely powerful language in terms of using the object oriented methodology in software engineering. The design approach used in this project was object oriented and therefore, can be expanded to find the k nearest neighbors of any object. As the applet security prohibits reading and writing of files, the design is based on a client server architecture. The current version of applets prevents access of files across the internet. But perhaps, later versions will provide for this facility, thus facilitating the task of accessing files without need for the client-server model. If the project had been an application, the need for a client-server model would not have arisen.

5. Main components

5.1 Client

A user can bring up the applet from a web browser at the URL http://www.y.glue.umd.edu/~annaval. The client comprises the User Interface called the Interface object, the Graph-Maker object and the Connection object.

- The Interface object is responsible for the graphical user interface which is displayed at the specified URL.

- All information requested or wished to be inserted in the *Database* is passed on to the *Database* via the Connection object, which connects to a *Server* on which the *Database* is located. If an error occurs in contacting the Server a ConnectionException is thrown and a message is displayed indicating the occurrence of the error.
- The Graph-Maker object is responsible for displaying on an applet window the x-y plot of the closing prices of the *k* nearest neighbors of a particular stock along with its distance from that stock.

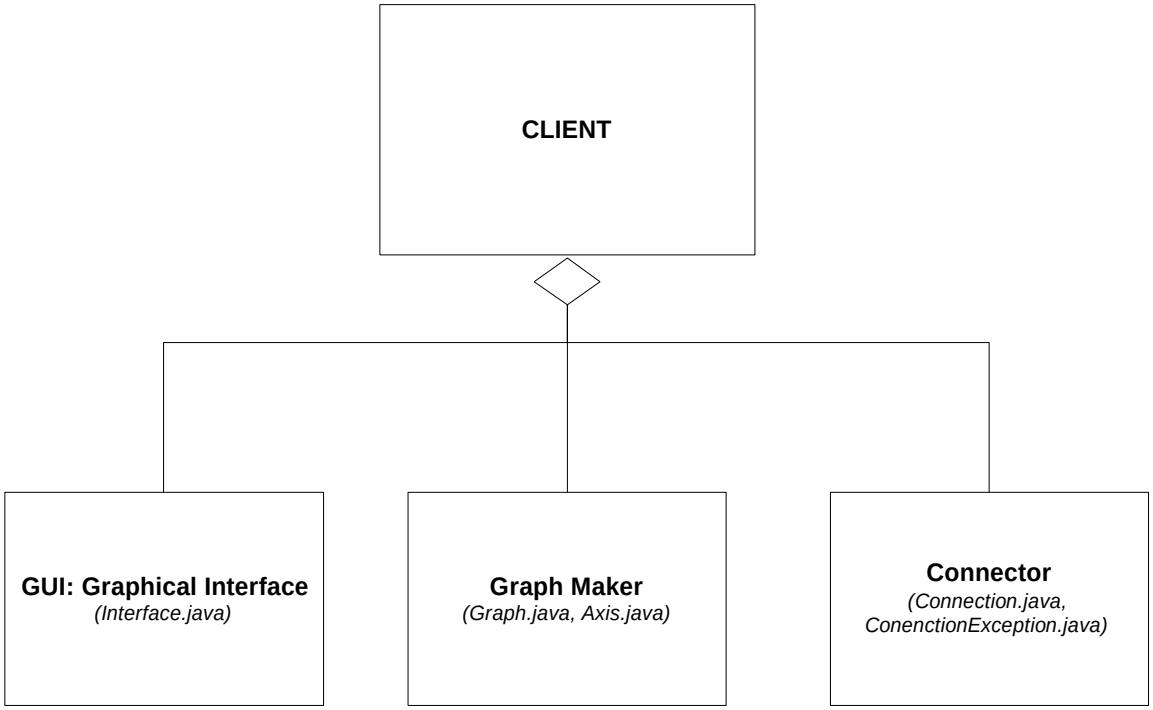

Figure 2.1: The Client component is an aggregation of the above components.

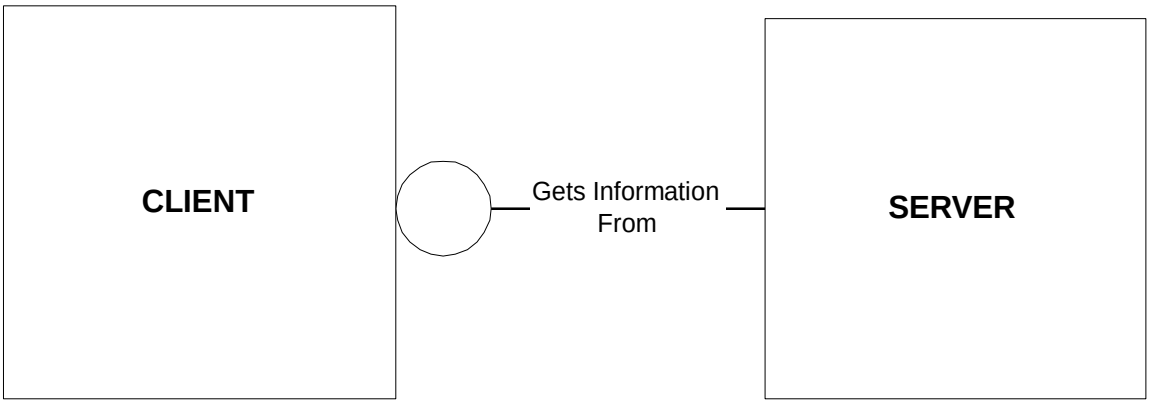

Figure 2.2: Many to one relationship

5.2 Server

The server is a multi-threaded server which allows several clients to connect to it, simultaneously. Every time a client starts up the applet, a new client connection is requested and the *Server* spawns off the responsibility of the new connection to one of its child processes. And whenever, the client is finished with its work the child process is killed but the *Server* continues to run.

Problems arise when two threads have access to the same object and both threads call a method that modifies the same object. This can lead to corrupted objects and loss of data. A problem that crop up in this program is when two users create two files with different stock information in the same sub-database. Normally, this would not cause a problem, but since, the database in this project is, merely, a creation of directories and files in that directory, problems could arise when two users have access to and modify the same file. The next section explains in detail the design of the database in this program.

The objects that the server is comprised of are:

- The MultipleConnection object which is the object which is responsible for creating the different threads so that multiple users can have access to the program simultaneously.
- Generator/Database Object: this object facilitates the insertion and search of stocks in the database. It is responsible for all communication with the database. An additional

responsibility of the Generator object is that it communicates with the VectorCalculator and DistanceCalculator objects.

- The VectorCalculator object calculates the vectors for the individual stocks based on the closing prices. These vectors are later used by the DistanceCalculator to calculate the distance between different stocks. The algorithm for the VectorCalculator is a shell script program written by Dr. Faloutsos. The algorithm is based on the Daubechies-4-wavelet transform. The vector consists of 5 numbers each representing a different value. For example, one number represents the average or mean of all the closing prices for that particular stock.
- The object whose function is to calculate the distance between stocks is the DistanceCalculator. In calculating the distance, it uses the vectors, calculated by the VectorCalculator object, for individual stocks. The current equation used by the DistanceCalculator object is the Euclidean equation.
- During a search for neighbors of a specific stock, the distances between that specific stock and other stocks in that sub-database, are sorted in ascending order by the Sorter object. The Sorter object can be used to sort any length of double numbers.

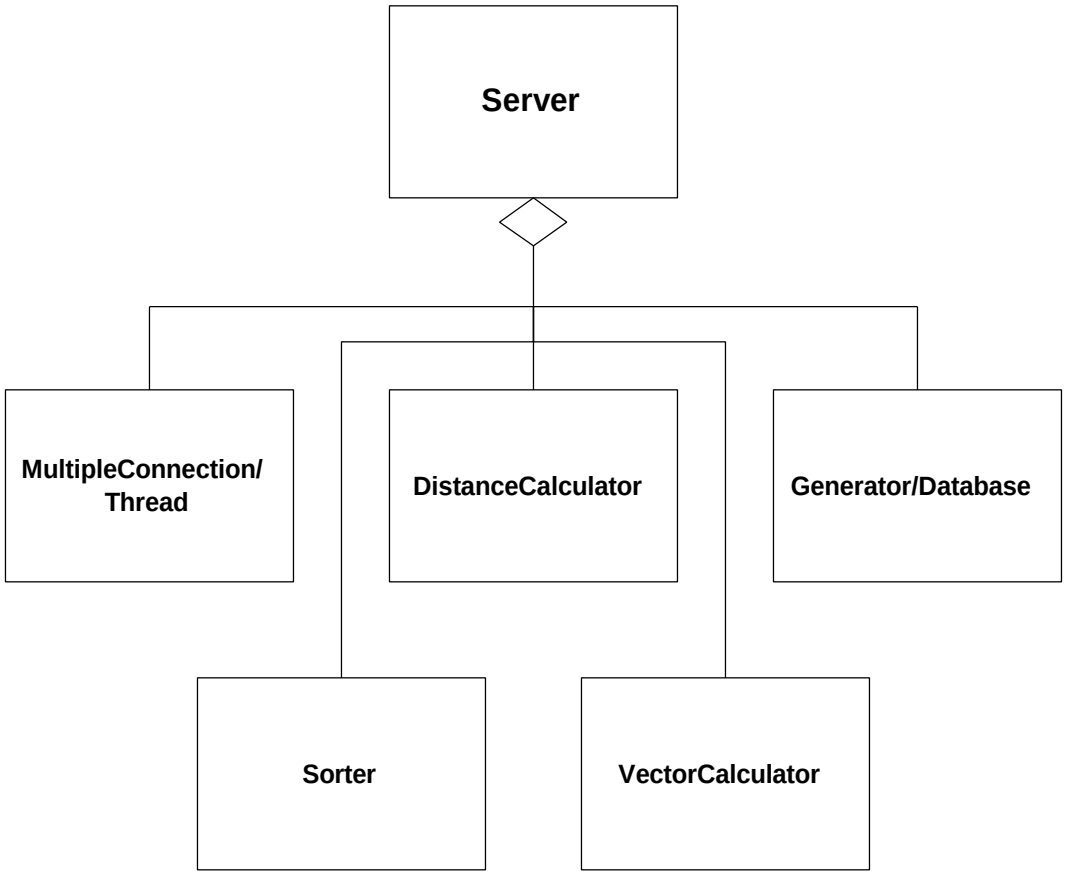

Figure 2.3 Object Model for the Server

6.0 Database

The database is responsible for the storing of files containing the quotes of closing stock prices. The database in this program is only a make-belief one which acts like a database but is not really one. Every time, a new sub-database is created by a user, a new directory is created and the stock quotes are stored in a file. The current version of the program disallows the deleting or appending of closing prices, to existing stocks. And neither does the program allow the deleting of sub-databases. Although, it would be relatively easy to provide these additional features in a later version of this program.

But the fact that there isn't a real database on the server is hidden from user, since this information is not germane for his purposes. Since this program was developed while

keeping in mind the object oriented framework, it would not require excruciating efforts to include a real database to store the closing prices of stocks.

7.0 User's Manual

The most important point to remember before you even start using the applet is to remember to start the applet on the same machine as the one on which you are running the program specific server. Once you know make sure that the applet and the server are running from and on the same machine, you can be sure that your instructions will be carried out properly.

- You can either insert new stocks in an existing sub-database, or create your own sub-database and add a new stock file with closing stock prices. There is nothing much that the program does differently if you are inserting a stock in an existing or a new sub-database. For either of these you have to insert information in three fields.
    1. The first field being the database field. If the sub-database already exists, the program does not create another one. Whereas, if the sub-database does not exist, it creates a new sub-database.
    2. In the second field, the user must type in the name of the stock.
    3. The last field which is an important field, the user has to type in the closing prices for that particular stock. The closing prices must be on separate lines, with one line containing only one stock price. If you make a mistake on any line, you can always return to it, with the mouse.

So to insert a stock whether in an old or a new sub-database, the user must select the "Insert" button. Once the "Insert" button is selected, the three fields mentioned above get activated. When the user is ready to send the information, he/she must click on the "OK" button which is located in the center, towards the bottom, of the applet. The user will see a message on the bottom of the applet, indicating whether there was a success or failure in the insertion. If one or more of the three fields mentioned above were not filled, an error is reported.

- The user can also search for neighbors of a particular stock in a specific sub-database. If the user wishes to conduct a search, the "Search" button has to pressed, so that the

fields related to the “Search” routine, can be activated. The “Search” routine, also has three fields which are the following:

1. The list of sub-databases that the user can choose from. Once a sub-database is selected, the names of all the stocks in that sub-database appear in another list box.
2. The user also has to select the stock in the sub-database on which he/she wishes to conduct a search.
3. The third field requires a number to be entered into it, which will represent the number of neighbors that the user wants to search for, for that particular stock.

One of the nice features of this program is that when a user inserts a new sub-database or stock, it gets added to the list of sub-databases and stocks in the search section of the applet. This has been added so that a user does not have to exit the program every time he/she inserts a new stock or sub-database. If all three fields are not filled, an error is reported. The status of the search is reported to the user on the status message box located at the bottom of the applet. If the search was unsuccessful, an error message is reported. Otherwise, the plots for the all k neighbors are drawn on separate windows. The header of each window reports the name of the stock along with its distance away from the stock on which the search is being conducted. The closing prices for the stocks are plotted on an x-y axis.

- If you wish to delete all the information you have typed, in stead of going to each field and pressing the delete or backspace key, you can click on the “CLEAR” button, located on the bottom of the applet. This will delete all the information from all the fields, except the list boxes containing the names of the sub-database and stocks.
- If you wish to exit the program, all you have to do is exit netscape. The server will kill the child process but the server will continue to run on the machine on which it was started.

8.0 Protocol

When data is being sent back and forth between the client and the server, each one has to know what the data represents. In the world of today, there is a protocol that exits in nearly every situation where communication is needed, whether it is in the diplomatic

circle or in the world of computers. I developed a protocol of my own, to facilitate the transfer of data and information between the client and the server. The user does not need to know how the client and server interact and how the data is deciphered by either one.

- So that the server knows whether the user requested an insertion or a search, the number 1 or 2 is placed first, in the bytes sent from the client to the server. If a 1 is read when the server is reading the bytes, it knows that an insertion is requested, and if the number 2 is read, a search is requested. The other data such as the name of the sub-database, the name of the stock and the closing prices of the stock are also sent to the server and are separated from the number 0 or 1 by ":". They are separated from one another by ",". The number 0 or 1 is placed first, and then the individual fields are placed after it. Each individual closing stock price is separated from other closing prices by semi-colons, ";" . For example, for an insert request the data sent from the client to the server could look like this, "0:mydatabase,IBM,123;124.01;109.98;...and so on. An example of search data is, "1:mydatabase,CISCO,3." The last number representing the number of neighbors requested by the user.
- When the applet is started, the client does a search for the names of the sub-databases on the server end, to make the task of the user as easy as possible. When this information is sent to the server, it looks like the following, "3:". The number apprises the server that a list of sub-databases is needed to requested by the applet to fill the list box attached to the applet.
- When a user chooses a sub-database while doing a search, the client requests the server for the names of all the stocks in that particular sub-database. The client informs the server of its request by placing the number 4 followed by a colon, ":", which in turn is followed by the name of the sub-database. For example, "4:mydatabase".
- After the client is informed by the server of a successful search and with the names of all the neighbors of a particular stock, the client requests the server for the closing prices of a particular stock, in a specific sub-database, so that it can draw the x-y plot for the stock. The server sends the needed request if it reads the number 2 followed by a colon, ":", and the names of the sub-database, and the stock. For example, "2:mydatabase,IBM".

The server sends information back to the client in the form of bytes, which has to be deciphered by the client in order to find out whether the insert or the search was successful or not.

- In an insertion, if the server returns the number 0 followed by a colon, ":" it ensues that there was an error and the insert request was not fulfilled.
- If the server returns the number 1 followed by ":", it represents a successful insertion.
- During a search, if the client receives the number 0 followed by a colon, it translates to mean that there were no other stocks in the sub-database besides the one on which the search was being conducted.
- If the client receives "1:", it represents a successful search, for all k neighbors specified by the user. And the graphs for the k neighbors are drawn on separate windows.
- If the client receives "2:" from the server, it means that not all k neighbors were found because the number of neighbors requested by the user were more than the number of stocks in the sub-database excluding the stock on which the search was being conducted.
- If the client receives "3:" followed by some error message, it means that the number of fields in the vector calculated for one of the stocks did not contain 5 fields. This leads to a problem when calculating the distance between that stock and the stock on which the search is being conducted because the DistanceCalculator object requires the vectors to be of length 5.

9.0 Enhancements to previous work

A similar program was written using the Tcl and Tk tool kit. This program is better than the previous program in terms of the following new features:

1. Ease of portability: The compiled program should work the same with no problems on a Windows 95 based system, or a Unix system. Whereas the previous program could not have worked with very little extra effort on both systems.

2. Web based program: This program can be accessed from the web, whereas the one written in Tcl-Tk could not be viewed from the web. This allows this program to be used by remote users.
3. Object Oriented Methodology: This program was written using an object oriented methodology. As a result, it will be very easy to make improvements or any enhancements to the code, without drastically changing it. For example, a real database can be added using JDBC and most of the code need not change.
4. Multi-users: This program supports multi users simultaneously. All the users need not have a copy of all the code. All they need is one copy, that has the server running on it, and all of them can use it concurrently. The earlier code that was written supported only one user at a time.

10.0 Conclusion and future work

This program is written for the sole purpose of searching for k neighbors of an object, in this case of stocks. Since it was written using an object oriented methodology, it can be expanded to include a search tool for any specifically defined object. An important feature of this program is that it can be run from the web and multiple users have access to it, simultaneously. It was modeled using a client-server model, because of applet security which prohibits reading and writing of locally stored files.

Improvements can be made to this program by adding code that will employ a real database using JDBC, in stead of an illusory database. Some enhancements that can be made are the searching of objects which can be places on a map.

11.0 My Experience

I thoroughly enjoyed working on this project. It was hard at first, since I did not have a clear understanding of what was the project was all about. But as I started working on it, I got a better understanding of the what features I was required to provide in this project.

Another reason for my getting off on a slow start was that I was learning the language as I was progressing in my work. I had done quite a bit of work, when I realized that my approach to the problem was completely wrong. And I had to redo a large portion of the already completed work. Anytime, I ran into any problems with JAVA, it took me some time to figure it out, since I had no friends who could help me because hardly anybody knows JAVA these days.

But on the whole, I really enjoyed working on this, and I can say very confidently that I have learnt a lot from the project. I learnt JAVA, of course. And if I had not been assigned to work on a relatively large project like this one, I don't think I would have learnt the language in the near future. This project, in a sense, forced me to learn JAVA. And I am very grateful to Dr. Faloutsos for assigning me this project. I also got a better understanding of client-server based architecture.

I would like to thank Dr. Faloutsos, once again, for giving me so much freedom in choosing my project and helping me learn so many other auxiliary topics.