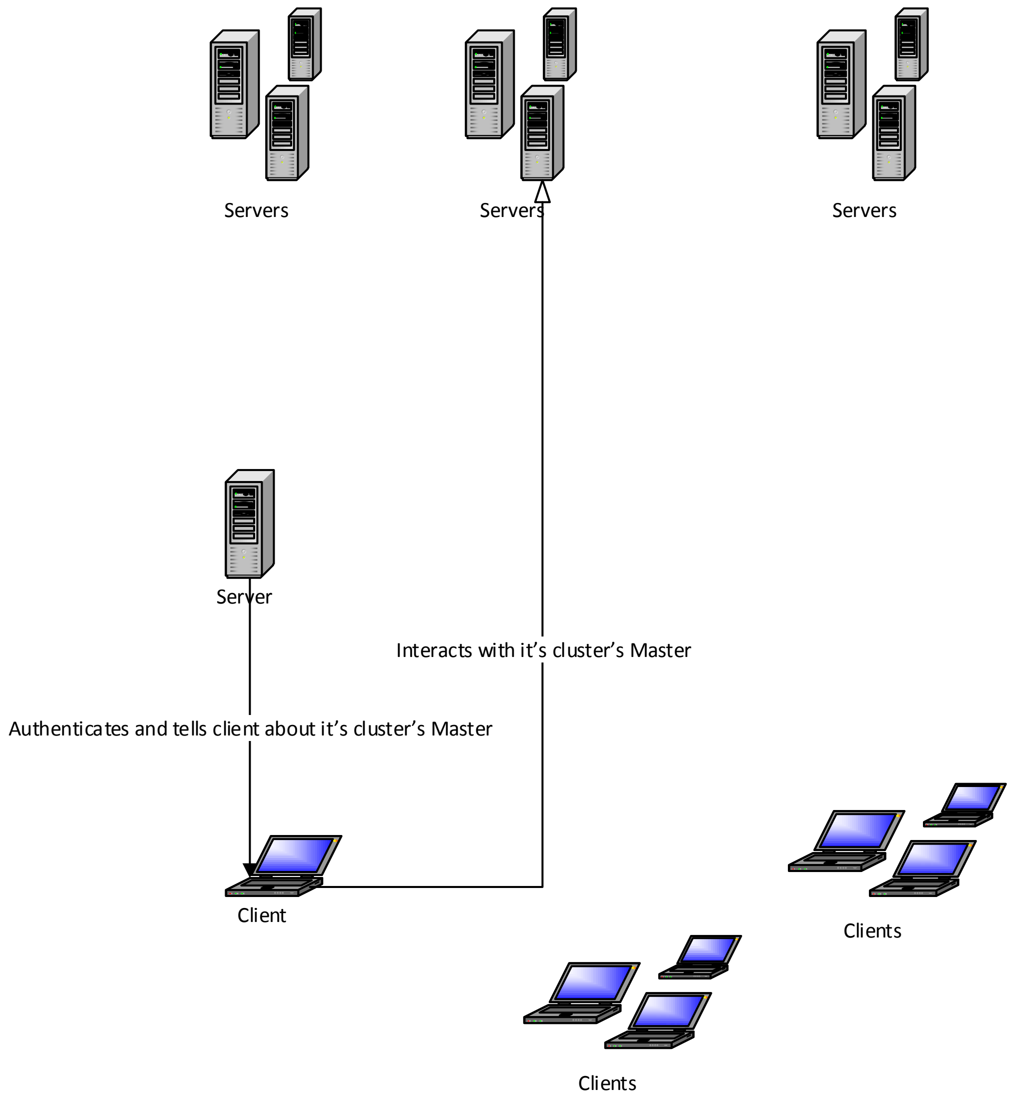
A Report on DBLike

Name	Student Numbers
Harsimran Singh Maan	87446126
Kuntal Joshi	87434122
Prabal Sharma	86932126

Architecture of DBLike


Servers
Servers
Servers
Server
Interacts with it's cluster's Master
Authenticates and tells client about it's cluster's Master
Client
Clients
Clients

The system uses Java RMI to enable the interaction between all systems. The application is designed to be used at any operating system that runs JVM.

This client is authenticated on server side by a server called Broker server that acts as a mediator and lets the clients know of the master server to connect at any given time.

This master server is elected based on the lowest serial number at any given time. The election is called as soon as the master goes down. Every node (except master) monitors the node with immediate lower serial number. When a file is changed on the client, it is replicated in a cluster of servers. A client belongs to a single cluster. A cluster can have multiple clients and there could be multiple clusters. A cluster includes a distributed broker service and set of file servers.

The client utility monitors a folder on the client machine and syncs it with the server. A user can run clients on the different machines and keep the folder in sync on all machines.

Client monitors the folder for modifications and uploads the changes to the master server. These changes are immediately replicated on all machines in the cluster. The interval of client poll to the server can be controlled through configuration.

File hashes are used to check a file for any changes and only modified files are exchanged between the server and the client.

Simultaneous editing of a file on two different client machines is controlled by versioning. A file with the name ~CONFLICT_origianalFileName is created with recent file that's uploaded on the server.
For example, if there are two files with the same name i.e. test.txt then it is uploaded as ~CONFLICT_test.txt in the client directory and client can resolve the conflict as he wishes.

We are implementing a time delay in our application before uploading a file on the server to allow retrieval of file. This ensures optimization in our application by maximizing the data that is exchanged on each roundtrip and minimizing the number of roundtrips for a given file when ot is edited.

Server side Implementation

We have a broker server to give details about the master server to the client. This is implemented by DBoxBroker. It has class as FileServerMonitor which checks whether the selected main server is running or not. If the server fails then it deletes that server which was acting as the main server. It also has an Authenticator class which authenticates the user to access this application. When the DBoxLike is started DBoxBroker starts the connection using Java RMI prior to any user input.

Our DBoxServer has an AliveCheck class which sends the heartbeat to Broker on regular intervals. FileServer class sets the path of the directory and also receive the file from the clients.

Client Side Implementation

We have a class called DirectoryManager which tracks the directory on the users machine which he want to share. Any changes in the directory are monitored by this class. It captures events like entry create, modify and delete. It scans through the path of the directory given by the user and gets all the file's names in that directory and calculates the hashes for those files and saves it as a hash map.
The HashManager is responsible for managing hashes for cases like upload, download, conflict and delete. It reads the hashes for the files and updates them as and when required for reliable file transfer functionality of DBLike.

Rationale for the design choices

DBLike is implementation of distributed system and can have thousands of users at a time using this application and so as the servers to provide services to those clients. Keeping this in mind we came up with all the design choices to create this application.

The Broker which is discussed in architecture of DBLike is assumed to be a well written and distributed server in itself. It is just a medium to authenticate a client and connect it to the master of the cluster to which the client belongs.

For so many clients we need to have proportionate servers. To make this possible, whenever a new machine joins a cluster, that machine/ server starts receiving updates from that cluster and makes that server updated and an ideal candidate to act as a master server in the cluster as and when required.

On every file change event on the master, the file is replicated to every other node in the cluster to which the master belongs. This ensures synchronization of the file. In addition to synchronization optimization was also crucial. For this we made the following design choice. In DBLIke we have implemented hash comparison. This is done locally and the server hashes are save in .dblike file. This .dblike file is then used for hash comparisons with the last known server hashes later on as and when required by the application.

Known bugs

With the extent of testing so far, the application is quite stable and no bugs are known so .

Possible extensions

Our application's next version will have file sharing feature, i.e. if a user knows about a different user using our application, then he can share particular files or directories with that user if he wishes. In addition to this we may utilize our application's platform independent feature in developing a mobile app. Since our application is developed on Java, in DBLike version 2.0 we will develop an interactive GUI for PCs, tablets and smartphones. Further optimization for uploads to carry on the changes in the file rather than the complete file at any given point can be made.