REQUIREMENTS   
  
ANALYSIS   
  
DOCUMENT

Haohan (Harry) Xu

Luis Ignacio Perez  
Jitin Jacob  
Sashiraj Dhanrajan

Table of Contents

[1. Introduction 3](#_Toc392546196)

[1.1 Purpose 3](#_Toc392546197)

[1.2 Scope 3](#_Toc392546198)

[***1.3*** Definitions, Acronyms and Abbreviations 3](#_Toc392546199)

[1.4 References 4](#_Toc392546200)

[1.5 Overview 4](#_Toc392546201)

[2. Glossary 4](#_Toc392546202)

[3. Use Cases 4](#_Toc392546203)

[3.1 Actors 4](#_Toc392546204)

[3.2 List of Use Cases 4](#_Toc392546205)

[3.3 Use Case Tables 5](#_Toc392546206)

[4. Design Overview 10](#_Toc392546207)

[4.1 Introduction 10](#_Toc392546208)

[4.2 System Architecture 11](#_Toc392546209)

[4.3 Processes 13](#_Toc392546210)

[4.4 Communication 14](#_Toc392546211)

[4.5 Synchronization 14](#_Toc392546212)

[4.6 Consistency 15](#_Toc392546213)

[5. Non Functional Requirements 15](#_Toc392546214)

[5.1 Usability 15](#_Toc392546215)

[5.2 Security 15](#_Toc392546216)

[5.3 Supportability 16](#_Toc392546217)

[5.4 Performance 16](#_Toc392546218)

[5.5 Implementation 16](#_Toc392546219)

[5.6 Interface 16](#_Toc392546220)

[5.7 Legal 16](#_Toc392546221)

[Appendix: 16](#_Toc392546222)

[1. Project Management 16](#_Toc392546223)

# Introduction

## Purpose

Distributed Systems is a software system in which components on a network system communicates and coordinate their actions using messages. Since its being used in large scale in the current computer industry, we need to understand the implementation of such distributed system by creating an application similar to ‘Dropbox’. In doing so, we will understand the nuances of working with a distributed system and its challenges while implementation. The purpose of this project is to develop a distributed system which is a basically file sharing system through an end user will share a particular folder with a cloud based service. Technically this folder will be hosted on one or more instances of the Windows Azure. Using the cloud service, the same folder can be viewed across many other end users if permission is given.

## Scope

Scope of our project restricts us in using three accounts of Windows Azure, hence will represent three cloud services and can have any number of end users. Each end user will be required to authenticate himself before using the system. Hence we provide the application only to the authenticated users. The features which are available in the system are Uploading, Downloading, Sharing and Modification. The project will be having proper architecture design for implementing the above mentioned features.

## Definitions, Acronyms and Abbreviations

***Client:***

A client is an end system which request a service from a system.

***Server:***

A server is a system which performs the request and process which comes from the client.

***Cloud System:***

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility (like the electricity grid) over a network.

## References

<http://en.wikipedia.org/wiki/Cloud_computing>

<http://en.wikipedia.org/wiki/Distributed_computing>

<http://azure.microsoft.com/en-us/documentation/>

<https://www.dropbox.com/>

## Overview

The Chapter 3 describes the Use Cases present in the project and each Use Case is described in detail. Apart from the Use Cases, we have also discussed in detail the System Design of the application which includes the Architecture etc. A summary of how we decided the application architecture and how we plan to implement is discussed below.

# Glossary

DB- Database

GUI- Graphical User Interface

MSDN- Microsoft Developer Network

# Use Cases

## Actors

The actors participating in the project are given below

* Server1
* Server2
* Server3
* Client(Multiple Clients)

## List of Use Cases

The list of Use Cases identified in the project corresponding to the requirements given are listed below:-

* Login
* Create Account
* Synchronization
* Upload
* Download
* Delete
* Modify
* Share
* Server Side Failure
* Logout

## Use Case Tables

|  |  |  |
| --- | --- | --- |
| *Use case name* | Login | |
| *Participating*  *actors* | Initiated by *Client* | |
| *Flow of events* | 1. The *Client* will use the application by logging into the system. | |
|  |  | 1. The application will prompt the user to enter the UserName of the client. |
|  | 1. The *Client* enters its UserName and press Enter. | |
|  |  | 1. *Server* checks if the UserName is a valid one present in the list of valid users and displays a message for the *Client* incase the UserName is not a valid one. |
|  | 1. The *Client* once enters the valid UserName, the *Server* will display him the files and folders present in his path corresponding to his particular UserName. | |
| *Entry condition* | * *Client* must be a having a Valid UserName. | |
| *Exit conditions* | * *Client* will be logged in and his files and folders are displayed. | |
| *Quality*  *requirements* | * The *Client* will be able to Login into the system very quickly. | |

|  |  |  |
| --- | --- | --- |
| *Use case name* | CreateAccount | |
| *Participating actors* | Initiated by *Client* | |
| *Flow of events* | 1. *Client* access the application by entering his name to login the system | |
|  |  | 1. If username doesn’t exist Server requests the *Client* to create an account by clicking the signup button |
|  | 1. *Client* Clicks the signup button and enters the client details and clicks the Create Account button | |
|  |  | 1. Server requests the *Client* to specify the path which will consider as the root path for the server. |
|  | 1. Client specifies the root path which is to be considered by the server | |
|  |  | 1. Server creates an account for the *Client* |
| *Entry condition* | * *Client should not have an account* | |
| *Exit conditions* | * New account is created for the *Client* | |
| *Quality*  *requirements* | * The Client account will be created in a minimal amount of time by the Server | |

|  |  |  |
| --- | --- | --- |
| *Use case name* | Synchronization: Upload | |
| *Participating*  *actors* | Initiated by *Client* | |
| *Flow of events* | 1. The *Client* will be adding a new file in the path specified by him and clicks on the Synchronization button. | |
|  |  | 1. The *Server* will be listening to the path for that particular user when the *Client* logs in. Any new file present in the path will be uploaded from the *Client* local path to the *Server.* |
| *Entry condition* | * *Client* must be a having a Valid UserName. | |
| *Exit conditions* | * *Client* files will be synchronized with the *Server* including the new files. | |
| *Quality requirements* | * The *Client* will be able to Login into the system very quickly. * The uploading of the new file into the *Server* will happen quickly. | |

|  |  |  |
| --- | --- | --- |
| *Use case name* | Synchronization: Download | |
| *Participating*  *actors* | Initiated by *Client* | |
| *Flow of events* | 1. The *Client* will be logged into the application and present in the path specified by him and clicks on the Synchronization button. | |
|  |  | 1. The *Server* will be listening to the path for that particular user when the *Client* logs in. Any new file present in the *Server* will be downloaded to the *Client* local. |
| *Entry condition* | * *Client* must be a having a Valid UserName. * *Server* must be running and listening. | |
| *Exit conditions* | * *Client* files will be synchronized with the *Server* including the new files. | |
| *Quality requirements* | * The *Client* will be able to Login into the system very quickly. * The downloading of the new file from the *Server* will happen quickly. | |

|  |  |  |
| --- | --- | --- |
| *Use case name* | Synchronization: Delete | |
| *Participating*  *actors* | Initiated by *Client* | |
| *Flow of events* | 1. The *Client* will be logged into the application and present in the path specified by him. The *Client* will delete a file/folder from the path and clicks on the Synchronization button. | |
|  |  | 1. The *Server* will be listening to the path for that particular user when the *Client* logs in. Any file not present in the in the *Client* will be removed from the *Server*. |
| *Entry condition* | * *Client* must be a having a Valid UserName. * *Server* must be running and listening. | |
| *Exit conditions* | * *Client* files will be synchronized with the *Server* including the files which are removed/deleted. | |
| *Quality requirements* | * The *Client* will be able to Login into the system very quickly. * The deletion of the file from the *Server* will happen quickly. | |

|  |  |  |
| --- | --- | --- |
| *Use case name* | Synchronization: Modify | |
| *Participating*  *actors* | Initiated by *Client* | |
| *Flow of events* | 1. The *Client* will be logged into the application and present in the path specified by him. The *Client* will be making any changes in any existing file and saves the changes. Once saved, clicks on the Synchronization button. | |
|  |  | 1. The *Server* will be listening to the path for that particular user when the *Client* logs in. Any file present having a modified timestamp different from the files in the *Server* will be uploaded to it. |
| *Entry condition* | * *Client* must be a having a Valid UserName. * *Server* must be running and listening. | |
| *Exit conditions* | * *Client* files will be synchronized with the *Server* including the modify files. | |
| *Quality requirements* | * The *Client* will be able to Login into the system very quickly. * The uploading of the modified file to the *Server* will happen quickly. | |

|  |  |  |
| --- | --- | --- |
| *Use case name* | Share | |
| *Participating actors* | Initiated by *Client1* | |
| *Flow of events* | 1. *Client1* uploads the file into the application and click the File Share button to share his file with Client2 | |
|  |  | 1. Server receives a request from the Client1 to share the file to Client2 |
|  |  | 1. Server process the request made by client1 and share the file to Client2 |
| *Entry condition* | * *Client 1 Should share a file* | |
| *Exit conditions* | * Server Share the file to Client2 | |
| *Quality requirements* | * Client2 Should receive the shared file without loss in data | |

|  |  |  |  |
| --- | --- | --- | --- |
| *Use case name* | ServerFailure | | |
| *Participating actors* | Initiated by *Server1* | | |
| *Flow of events* | 1. If *Server1* cannot process the request made by the Client, the control passes to the Server2 | | |
|  |  | | 1. Server2 automatically gets initiated and respond to the Client |
|  |  | 1. If Server1 and Server2 is failed then control passes to Server3 | |
|  |  | 1. Server3 automatically gets initiated and will respond to the client | |
| *Entry condition* | * *Server1* Should be failed * *Server1* and Server2should be failed | | |
| *Exit conditions* | * Server2 initiated automatically * Server3 initiated automatically | | |
| *Quality requirements* | * Connection will exist between the Client and server even if Server1 and server2 fails | | |

|  |  |
| --- | --- |
| *Use case name* | Logout |
| *Participating actors* | Initiated by *Client* |
| *Flow of events* | 1. *Client* need to exit the application and clicks the log out button |
|  | 1. Server process the request and client gets   disconnected |
| *Entry condition* | * *Client* Should be logged in |
| *Exit conditions* | * The application is stopped and disconnected |
| *Quality requirements* | * Client successfully logged out of his application |

# Design Overview

## Introduction

The application operates as a distributed system that provides storage and file sharing service to a number of different clients. The system provides transparency by hiding the internal behavior of the application and the use of multiple server machines.

The client application running on user’s machine establishes connection with a particular service server (SS). The application is capable of redirecting user requests to one of the backup servers (BS) in case of failure of a particular service server.

File storage is distributed among four service servers, each one of them storing a different subset of files. The system provides two backup servers where each one contains a copy all the files distributed among the service servers. By providing this architecture, the system is capable of tolerating the failure of two simultaneous servers.

In order to keep track of file locations and appropriately redirect requests referring to them, each server in the system must maintain a routing table acting as a file index.



Figure 1: High Level System Architecture

The following sections provide a brief explanation of design features of the application. The general system architecture and other important design features such as processes and multithreading, communication mechanisms, synchronization and data consistency are covered in the following sections.

## System Architecture

The system is based in the client-server model and divides its components using a three tier architectural style, having an interface subsystem, application logic subsystem and a storage subsystem. The logic subsystem is present in both the client and server machine. These components are organized on the client and server side as follows:



Figure 2: UML Components Diagram of the System

The system architecture uses a decentralized architecture in which we observe a horizontal distribution of the multi-tiered client-server architecture spread among different machines with the same logical components. Each one of these machines is in charge of managing the data corresponding to one particular user of the system.

Our system consists of six server machines in total. The first five servers are assigned to store the information of particular users whereas the sixth server machine serves as a backup unit and stores a copy of all the data in the system. The system’s architecture considering one possible client is shown below:



Figure 3: UML Deployment Diagram

## Processes

The communication among servers and between server and client will use asynchronous sockets, so both the client daemon and server side application will be implemented in multithreading.

When the client connects to the server, multiple threads will be used and each thread creates a TCP socket. One thread will take responsibility for receiving and sending messages from or to server, and other threads will be used for data transmitting from the client to servers.

In the normal server side application, multiple threads will be created for communication with clients and other servers include buck up server.

Theoretically, a normal serve will have at least one thread that creates socket connection between the servers and buck up server for data back up. Two threads will be created when a client connects to the server.

On the buck up server, at least five threads are created for connections to other four normal servers and the other back up server. The threads for data transfer between client and back up server will be created when some other servers failed.

## Communication

The communication method between client and servers, and among servers will be based on TCP sockets.

When client connects to a server, a socket will be created and first receiving authorization message from server side. After authorization, such socket will synchronize data on the client side according to the data on server side. If the normal server is failed, and client side daemon detected such failure, the client daemon will try to connect to one of the buck up server to retrieve data.

There are also sockets among servers. One server will have socket for data transferring between itself to backup servers, and other sockets for message passing to other servers.

## Synchronization

Client – Server Synchronization:

The synchronization between the client and server (service) can be achieved by using Router Table. In the beginning the client will use the config file to identify which server is more responsive and the client will connect to that particular server. This will help the client to connect only one particular server.

Once connected the client will do the operations such as upload, download and sharing of files. Since we are provided with three service servers, the client will place the file into the client which it is connected to it. Once placed the details are properly inserted into the Router Table attached to the server.

The Router Table consists of four columns namely UserName, Filename, Version and Server Name. All the servers will have their own Router Tables and all the Router Tables will be synchronized with each other. Thus every Router Table will have the same information across all the servers. This will allow the client to know which file is present in which server during the request.

In addition to three service servers, we will also have two Backup servers which will be continuously monitoring the three servers and taking backup of all the files present in three servers.

If situation arises where a file is present in a particular server and that server is not working. The client will be redirected to Primary Backup Server and get the file. In case, if the primary Backup server also fails, the client request will be transferred to the Secondary Backup server.

During synchronization, each particular client will also receive all the files pertaining to itself from the all three service servers.

Server – Server Synchronization:

All three service servers are capable of processing simultaneous requests from multiple clients. Each server will have files corresponding to the requests completed for particular clients. This means that every server will not have every file. Instead they all will have their own Router Table which will provide the details of which file or folder is present in which server. These Router Table are synchronized regularly.

Also the synchronization between the Service Servers and the Backup Servers happen regularly such that all the data in three servers are noted by both the Backup servers. In case, if the service server goes down, it will take all the data from the Backup server when it comes back online.

Every server when started, will have to synchronize itself and the router tables from the Backup server such that all the latest information of files are present.

## Consistency

Different data records (files) are stored in separate service servers and both backup servers maintain a copy of every the records in all service servers. Thus, data consistency must be maintained among one individual service server (that contains that particular file) and the two backup servers.

Data replication provides a mechanism to ensure its availability in the event of a service server or backup server failure. In such a case, the request for the file is redirected to the available backup server or to the service server. The provided architecture can tolerate failure of two simultaneous server failures. The next figure shows how the system handles failure of one service server and one backup server.

When a server is added to the system, it will ask other servers for the routing table and provide service to client.

When a server is reset from failure, it will check the newest versions of files those stored on such server, and synchronize them with other servers.



Figure 4: Server Failure Case

In order to prevent data corruption and ensure that at every time files have the latest modifications incorporated, the application assigns a version number to every file stored in the system.

Version number for each file is updated every time a modification is made and committed (synchronized) to the server. The version number is maintained in the routing table of each server and the information regarding file versions is propagated when version modifications are made.

The version number allows identifying the file that must be preserved in the event of encountering two identical filenames with inconsistent (different) content.

In particular, this situation can be observed in the event of two different clients modifying a shared file. Both will acquire the file having the same version number and will attempt to update its content and send it back to the server. In this case where both files have the same version, only the last upload client request will prevail. This scenario is explained in the following figure:



Figure 5: Version Conflict Resolution

# Non Functional Requirements

## Usability

The Drop box application should provide simple interface for users to perform their functionalities with minimal or no guidance. The controls and corresponding functionality is mapped to the users. Each user can upload, download and share the files with other users in an efficient way.

## Security

The Drop box application should accept the unique credentials from each user. User authentication must be offered by the system based on which the corresponding functionality of the user can be achieved successfully.

## Supportability

The Drop box application can be deployed in 32 bit or 64 bit version of Microsoft Windows operating system. The Server program is deployed in Windows Azure and all the data is stored in Windows Azure.

## Performance

The average update time for the application should be less than 10 seconds

## Implementation

The application should run on any Operating Systems. The system should have active Internet connection to perform the required functionality.

## Interface

The Drop box application must be developed in such a way that it supports the extensibility in the functionalities or modifications without developing the application from scratch

## Legal

The Drop box application should be legal property to the Drop box management team.

# Appendix:

## Project Management