COMP90015 Distributed Systems

Semester 1, 2017

Project 1 – EZShare

Resource Sharing Network

Due Date: 29th April, 2017

Group Name: DT

Liam Sanders	Scott Lee	Shuyan Hu (Sven)	Yuanyu Guo (Steve)
886664	793264	874083	814284
Lsanders2	Sangchull	Shuyanh1	Yuanyug
Lsanders2@student.unilemb.edu.au	sangchull@student.unimelb.edu.au	shuyanh1@student.unimelb.edu.au	yuanyug@student.unimelb.edu.au

Introduction

Project 1 - EZShare addresses the challenges and implements a solution for file sharing across multiple servers and to several clients while appearing as a single entity to the clients. This is the fundamental function of any distributed system. The code will be implemented in Java 1.8 for both the server and the client. A Client-Server structure will be used while server to server communication will be transparent to the client. There will be no peer to peer communication implemented at this stage of the project.

The main technical challenges faced will be: Concurrency – ensuring that multiple clients accessing and potentially editing resources at the same time is well managed and safe. Access levels – the system will limit file access to approved clients through the use of unique keys that must be known by the client attempting to access the resource. Cross server communication – seamless communication between servers pertaining to the information currently stored on them, this will ensure that clients can have access to the resources available to them while ensuring client transparency. The implemented solution must successfully allow for the sharing of multiple files across multiple servers to clients. Another major challenge in developing distributed systems is Heterogeneity, this has been deemed out of scope for this project and will not be addresses at this stage.

1. Scalability

There are clear scalability challenges that will be faced when expanding and growing the system. Some will be general scalability challenges that all systems face while some will be project unique challenges. A system is said to be scalable if "*it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity*".

Utilising Vectors to store resources

The approach used in this project for storing resources is through the use of Vectors (see Image 1 in appendix). Although Vectors have many benefits with a system this size, especially when it comes to threading, concurrency and synchronisation, there is a large scalability drawback. Vectors have a maximum size of approximately 2.1 billion elements, meaning that the system will run out of resources relatively quickly when speaking in terms of a distributed system. This will not just impact performance but will essentially limit the system from expanding further, making this the scalability challenge with the biggest potential negative impact on the system. The development team would begin to sort by file type into different resource lists once this became an apparent issue, this would allow us to divide our resources into more manageable portions that our lists can still manage.

Searching

Searching through resources, server lists and clients will eventually, with growth, prove to be detrimental to performance with this distributed system as more and more clients want to search concurrently. The search algorithm currently implemented provides a time complexity of O(n) while utilizing a single loop for all searches, as shown in Image 1. With the increase of resources, the time to return a result will increase exponentially not to mention the queue of requests will prove to be a huge bottleneck and will negatively impact performance. This is a major scalability issue that must be addressed, the development team recommends a Cache be implemented in future releases to tackle this issue.

Hardware

Scalability challenges pertaining to hardware issues such as servers and physical connections are hard to identify with a project of this nature given that virtual machines are being used for testing of the current implementation. That is not to say that there will be no hardware scalability issues, there will be, although they will not be accurately identified with the current release. Hardware challenges are still a major factor at play as one of the biggest areas for performance bottlenecks, especially when components are geographically separated. With distributed systems that span a wide area physically, the system starts to be heavily reliant on third party hardware such as routers, wired & wireless connections and processors that are not managed by the development or ongoing maintenance teams. This presents a unique challenge as there has to be solutions devised that work around the problem instead of directly solving the issue, as it may be inaccessible. This issue can be addressed by investing in hardware that will withstand heavy traffic, third party hardware issues must be worked around.

The final major scalability issue that this report will address is the memory used to store resources. Currently, the servers store all resources in the local in physical memory (RAM), this memory is very limited (average 8GB-16GB) and cannot store many resources. A recommended expandable solution is to use swap space on a hard disk which is easily scalable to store more resources.

2. Concurrency

Throughout the initial development phases, concurrency issues quickly became a challenge that the system must handle. Removing a resource while another client was trying to access it was the first point that the development team experienced a concurrency issue. This was while we were utilizing an ArrayList. An ArrayList cannot be shared between multiple threads, making the use of an Array unsafe to use in a concurrent environment. Once this issue was identified a solution was presented by the development team. It was clear that the use of a Vector, although performance was impacted, was essential given its multithreading capabilities.

The use of Vectors to store resources has been implemented for several reasons, amongst the top reasons was that it allows for safe multithreading. All methods that change the structure of the vector i.e. adding and removing resources, are synchronised. This means that we can provide multithreading functionality safely with a minimal impact on performance at this point of the development stage. This is not to say that with the increase of clients, servers and resources this method may be inviable due to the loss of performance associated with utilizing Vectors over ArrayLists. The development team have not used synchronised methods for every method that may edit the lists as the use of Vectors address this issue.

The code blocks presented in Image 3 & Image 4 show how the development team has implemented the query command. There is a concurrency issue here specifically when there are several servers in the list and response is set to 'true'. The first issue we ran into was when there were several threads that needed to be handled concurrently when queries were sent out to all the servers in the server list and, depending on the response, the same instance of the query class would potentially be executed by another thread. We decided to go with the use of three concurrent, Callable threads and if there are more than three servers in the list then the remaining threads will be put in a queue. This is better than the use of Runnable threads due to the potential of the same instance of the query class being executed by another thread. These Callable threads can throw an exception while they could not if they were Runnable. If there are too many concurrent threads, the following line of code will hold back the main thread until the others have been executed (*match_List*.addAll((List<Resource>) future1.get(5,TimeUnit.***SECONDS***)); *highlighted in Image 3*). The first available thread will send a queued query to the next server in the list. The development team suggest the use of more concurrent threads be used to tackle this issue when the server list grows and more clients are sending queries to more servers.

3. System Transparency

A distributed system should, by nature, appear as a single entity to the client and in turn the user rather than a network of interconnected systems. The client should not be able to see any details of a requested file pertaining to location or the complexity of the entire distributed system. This project has been developed with the intention of full client transparency in all aspects of the system.

Location

One of the most important aspects of system transparency is location transparency. The client cannot access the server list or the resource list on any of the servers, the server that the client has connected to executes these commands and will return a response to the client. This provides full location transparency as the server will never reveal to the client the location of a file it is trying to access. All files will be transferred through the server which the client has already established a connection with, while this is a bottleneck it is more essential to provide this transparency.

Concurrency

As previously identified in this report, there are several concurrency challenges that have been addresses throughout the development of the current system. All concurrency in the system is completely transparent to the client, the client requesting a file cannot see any other clients attempting to concurrently access the same file or server. There may be a slower response once a client has been put in a queue but the queue itself is not accessible by the client or viewable by the user.

Access

A client requesting access to a file may receive an error, negative response or a successful response from the server that it has established a connection with. The respective server where the file is located will never be disclosed to the client nor how this server was accessed. The client does not know which servers have been searched or the resource list of the respective servers. Also, client-server communication will never reveal the access methods used for accessing servers and files.

Replication

Replication transparency is concerned with a file being accessed and copied by several servers and/or clients at once. How many times a file is accessed, copied or moved location will never be disclosed to the client in the current implementation. As long as a copy of the file is still located on one of the connected servers in the distributed system an approved client will have access to the file with no noticeable change. Our system addresses this issue by searching for files using unique keys rather than specific server location. Also, the server executes the query command to search other servers rather than the client being responsible for this.

References

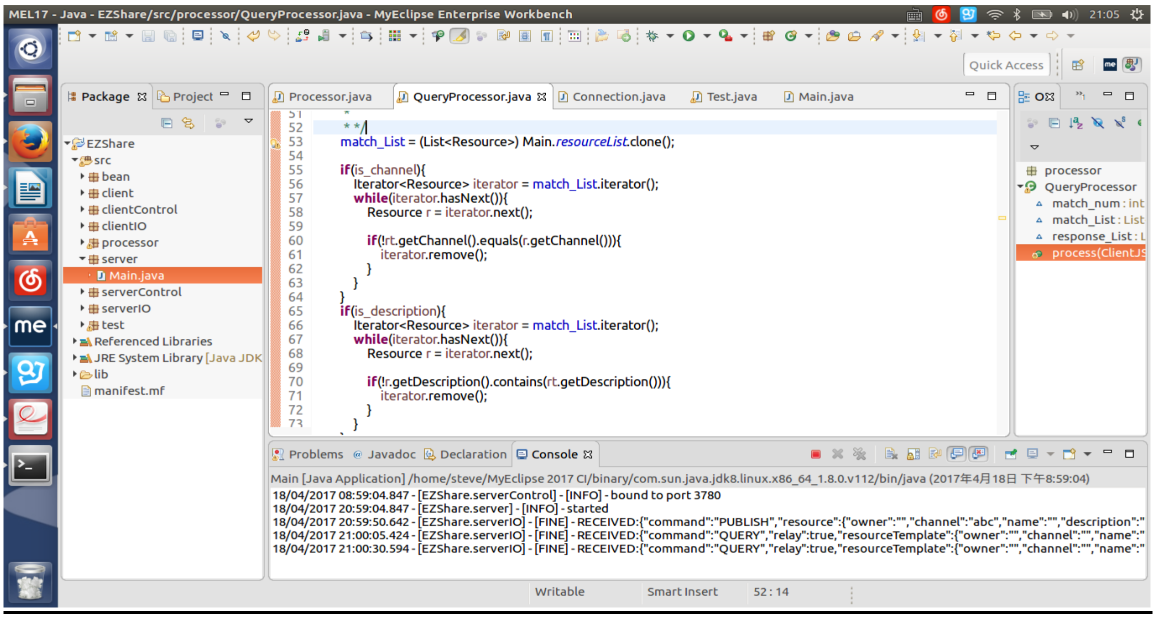
Brent, N, David, E., Mathhew, L. 2004, ‘The Ganglia distributed monitoring system: design, implementation and experience’, *Parallel Computing,* Vol. 30, Issue 7, pp. 817-840, Elsevier.

Foster, I., Kesselman, C., Nick, J. Tuecke, S. 2002, ‘Grid services for distributed system integration’, *Computer (IEEE Journal)*, Vol. 35, issue 6, pp. 37-46, IEEE.

Stroud, R. 1993, ‘Transparency and reflection in distributed systems’, *ACM SIGOPS Operating Systems Review,* Vol. 27, Issue 2, pp. 99-103, ACM, NY, USA.

Appendix

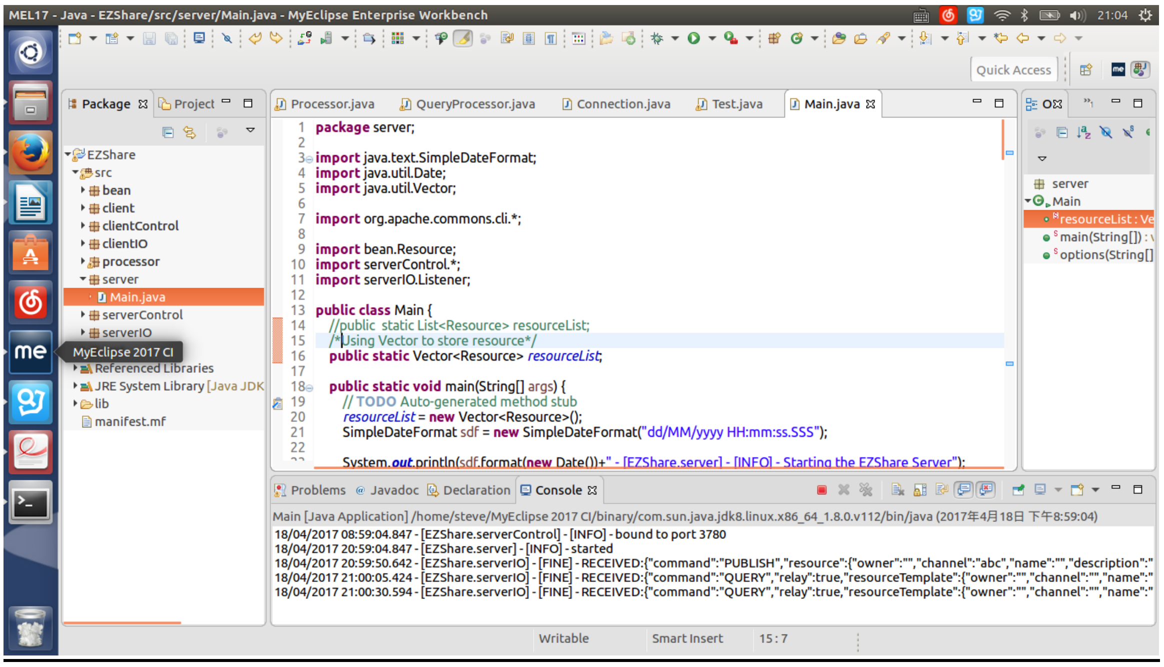
Image 1


MEL17 - Java - EZShare/src/processor/QueryProcessor.java - MyEclipse Enterprise Workbench

```
         * */
        match_List = (List<Resource>) Main.resourceList.clone();

        if(is_channel){
            Iterator<Resource> iterator = match_List.iterator();
            while(iterator.hasNext()){
                Resource r = iterator.next();

                if(!rt.getChannel().equals(r.getChannel())){
                    iterator.remove();
                }
            }
        }
        if(is_description){
            Iterator<Resource> iterator = match_List.iterator();
            while(iterator.hasNext()){
                Resource r = iterator.next();

                if(!r.getDescription().contains(rt.getDescription())){
                    iterator.remove();
                }
            }
```

```
Main [Java Application] /home/steve/MyEclipse 2017 CI/binary/com.sun.java.jdk8.linux.x86_64_1.8.0.v112/bin/java (2017年4月18日 下午8:59:04)
18/04/2017 08:59:04.847 - [EZShare.serverControl] - [INFO] - bound to port 3780
18/04/2017 20:59:04.847 - [EZShare.server] - [INFO] - started
18/04/2017 20:59:50.642 - [EZShare.serverIO] - [FINE] - RECEIVED:{"command":"PUBLISH","resource":{"owner":"","channel":"abc","name":"","description":"
18/04/2017 21:00:05.424 - [EZShare.serverIO] - [FINE] - RECEIVED:{"command":"QUERY","relay":true,"resourceTemplate":{"owner":"","channel":"","name":"
18/04/2017 21:00:30.594 - [EZShare.serverIO] - [FINE] - RECEIVED:{"command":"QUERY","relay":true,"resourceTemplate":{"owner":"","channel":"","name":"
```

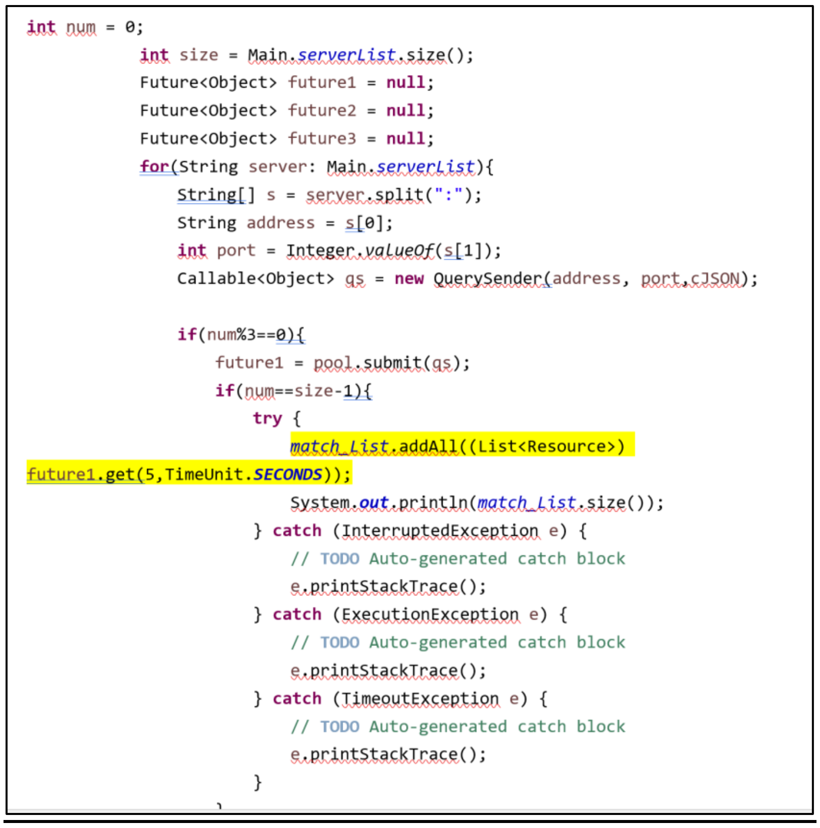
Image 2


MEL17 - Java - EZShare/src/server/Main.java - MyEclipse Enterprise Workbench

```
package server;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Vector;

import org.apache.commons.cli.*;

import bean.Resource;
import serverControl.*;
import serverIO.Listener;

public class Main {
    //public  static List<Resource> resourceList;
    /*Using Vector to store resource*/
    public static Vector<Resource> resourceList;

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        resourceList = new Vector<Resource>();
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss.SSS");

        System.out.println(sdf.format(new Date())+" - [EZShare.server] - [INFO] - Starting the EZShare Server");
```

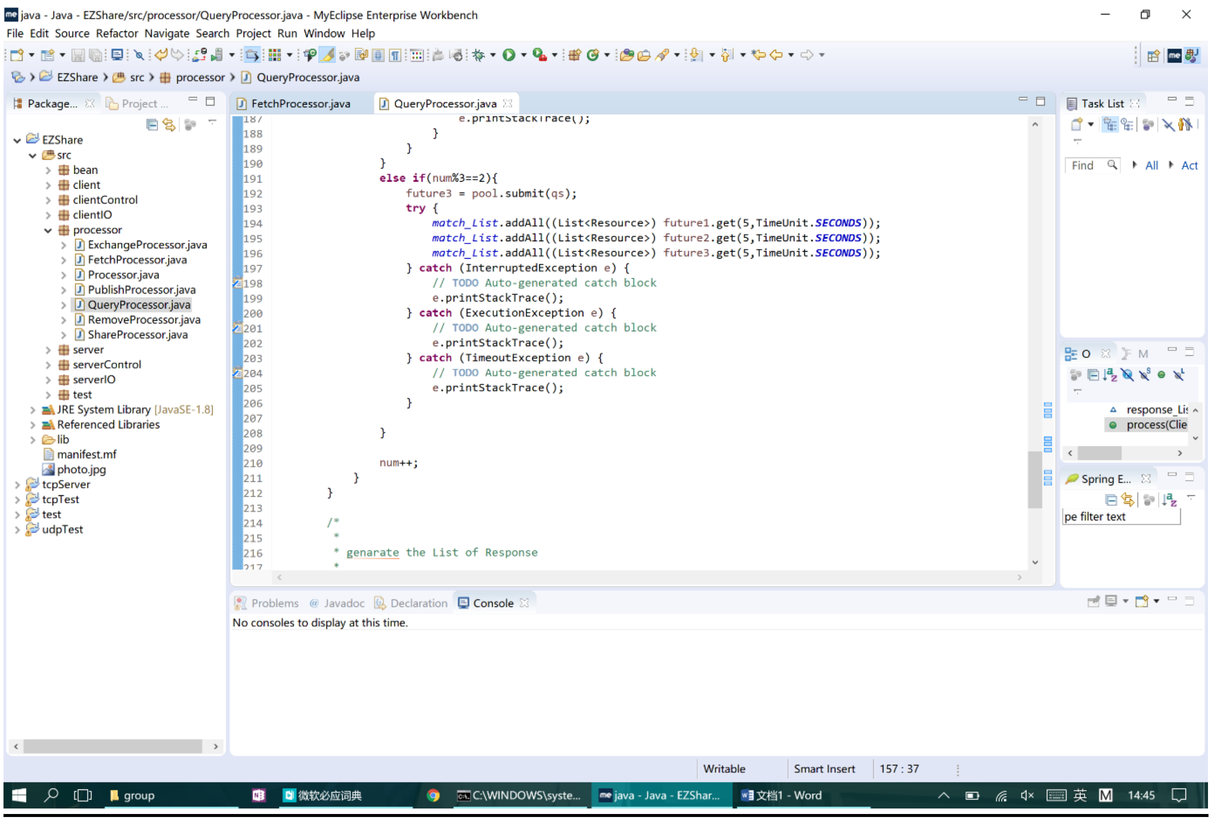
```
Main [Java Application] /home/steve/MyEclipse 2017 CI/binary/com.sun.java.jdk8.linux.x86_64_1.8.0.v112/bin/java (2017年4月18日 下午8:59:04)
18/04/2017 08:59:04.847 - [EZShare.serverControl] - [INFO] - bound to port 3780
18/04/2017 20:59:04.847 - [EZShare.server] - [INFO] - started
18/04/2017 20:59:50.642 - [EZShare.serverIO] - [FINE] - RECEIVED:{"command":"PUBLISH","resource":{"owner":"","channel":"abc","name":"","description":"
18/04/2017 21:00:05.424 - [EZShare.serverIO] - [FINE] - RECEIVED:{"command":"QUERY","relay":true,"resourceTemplate":{"owner":"","channel":"","name":"
18/04/2017 21:00:30.594 - [EZShare.serverIO] - [FINE] - RECEIVED:{"command":"QUERY","relay":true,"resourceTemplate":{"owner":"","channel":"","name":"
```

Image 3



```
int num = 0;
            int size = Main.serverList.size();
            Future<Object> future1 = null;
            Future<Object> future2 = null;
            Future<Object> future3 = null;
            for(String server: Main.serverList){
                String[] s = server.split(":");
                String address = s[0];
                int port = Integer.valueOf(s[1]);
                Callable<Object> qs = new QuerySender(address, port,cJSON);

                if(num%3==0){
                    future1 = pool.submit(qs);
                    if(num==size-1){
                        try {
                            match_List.addAll((List<Resource>)
future1.get(5,TimeUnit.SECONDS));
                            System.out.println(match_List.size());
                        } catch (InterruptedException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        } catch (ExecutionException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        } catch (TimeoutException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
```

Image 4



java - Java - EZShare/src/processor/QueryProcessor.java - MyEclipse Enterprise Workbench

```
                        e.printStackTrace();
                    }
                }
            }
            else if(num%3==2){
                future3 = pool.submit(qs);
                try {
                    match_List.addAll((List<Resource>) future1.get(5,TimeUnit.SECONDS));
                    match_List.addAll((List<Resource>) future2.get(5,TimeUnit.SECONDS));
                    match_List.addAll((List<Resource>) future3.get(5,TimeUnit.SECONDS));
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (ExecutionException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (TimeoutException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }

            num++;
        }
    }

    /*
     *
     * genarate the List of Response
     *
```

No consoles to display at this time.

Team Minutes

Date	Monday, 27th February
Members Attended	All (Liam Sanders, Scott Lee, Yuanyu Guo (Steve), Shuyan Hu (Sven))
Location	WeChat
Notes	Created group, introduced ourselves, made sure all group members' intended outcome for this class were inline.

Date	Monday, 6th March
Members Attended	All
Location	Redmond Barry Building, University of Melbourne
Notes	First in-person meeting.

Date	Wednesday, 22nd March
Members Attended	All
Location	Law Building, University of Melbourne
Notes	High level overview of how we want the project to run. Divided coding in to general groups, server classes, client classes etc.

Date	Thursday, 23th March
Members Attended	All
Location	111, Leicester Street, Carlton
Notes	Set up GitHub repository to share code, made sure everyone's different development environments are working (Eclipse and NetBeans). Split up coding into client side code and server side code. Steve already had a bit of pseudocode written that he explained to other group members. Discussed project challenges. Liam: will write report (everyone will send in the challenges they faced), project plan and minutes. We decided to start with the 17 client side command line arguments and then move on to the server commands.

Date	Saturday, 25th March
Members Attended	Scott Lee, Yuanyu Guo
Location	GSA Building, University of Melbourne
Notes	Briefly discussed structure of client side code and high level functionality. Identified which commands do, and do not require client-server communication (are executed locally).

Date	Monday, 27th Mach
Members Attended	All
Location	111, Leicester Street, Carlton
Notes	Divided the 17 client command line arguments amongst all members. Planned next meeting and set it as due date for these functions to be coded (Wednesday, 29 March). Each member chooses at least 2 client command line argument that DO require client-server communication and 2 that DO NOT require communication (executed locally). Steve: Debug, Secret, Port, Host. Liam: Description, Channel, Query, Remove. Scott: Publish, Exchange, Servers, Owner, Name. Sven: Fetch, Share, Tags, URI. Scott is responsible for one extra command (Servers) because it is needed for the exchange command.

Date	Wednesday, 29th March
Members Attended	All
Location	111, Leicester Street, Carlton

Notes	Combined all client side code that has been developed by all team members. We tested this code and made sure there were no bugs, debugged and fixed all errors. Divide the server side code into tasks for each team member. We did an hour of individual coding in this meeting. Decided on next meeting: Monday, 3rd April.

Date	Monday, 3rd April
Members Attended	All
Location	111, Leicester Street, Carlton
Notes	All group members updated the rest of the group on their progress. Identified some early issues that are arising. Sven changed from NetBeans to MyEclipse so the entire group is using the same development environment. We went through the general structure of the Server code. We identified all group members' next deliverable. Liam: working Remove command. Scott: working Exchange and Publish commands. Sven: Query and Steve: all other server commands. These are due at the next meeting on Wednesday, 5th April.

Date	Wednesday, 5th April
Members Attended	All
Location	111, Leicester Street, Carlton
Notes	Put all code that is completed together. Discussed issues that we are facing, the group debated different approaches to tackle the issues. Documented challenges to be added to report.

Date	Sunday, 16th April
Members Attended	All
Location	ERC, University of Melbourne
Notes	Combining all individual code, completing gaps in the code. Continuing the report, completed first draft.

Date	Thursday 27th April
Members Attended	All
Location	111, Leicester Street, Carlton
Notes	Gathering last parts of code that needed to be changed. Reviewing the report as a group. Testing the system as a whole. Identifying bugs and removing them. Ensuring complete system functionality and that all requirements are satisfied.