**COMP90015 Distributive Systems**

Semester 1, 2017

**Project 1 – EZShare**

**Resource Sharing Network**

Due Date: 29th April, 2017

**DT**

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

**Introduction**

Project 1 - EZShare addresses the challenges and implements a solution for file sharing across multiple serves and to several clients while appearing as a single entity to the clients. This is the fundamental function of any distributive 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: Security & Accessibility - ensuring that all communication is secure as well as limiting file access to approved clients. 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. Concurrency – ensuring that multiple users accessing resources at the same time is well managed and safe. The implemented solution successfully allows for the sharing of multiple files across multiple server to clients.

**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*”.

*The use of Vectors to store resources*

The approach used in this project for storing resources is through the use of Vectors. Although Vectors have many benefits with a system this size, especially when it comes to threading and concurrency, 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 distributive system.

*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 distributive systems that span a wide area physically, the system starts to be heavily reliant on third party hardware such as routers, wires 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.

*Searching*

Searching through resources, server lists and clients will eventually, with growth, prove to be detrimental to performance with this distributive system. The search algorithm currently implemented provides a time complexity of O(n) while utilizing a single loop for all searches. 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.

*Bottlenecks*

There are several areas where our system will experience a loss in performance with the increase of client systems. \*\*\*A resource must pass through servers to reach the client\*\*\*

**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 proved to impact performance, 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.

Another factor that must be addresses is the approach used to share resources and server lists. Currently, the active servers will, at a predetermined interval, send out the most current server list to another random server. This will eventually result in all servers having an up to date list of all other servers accessible to them. This, while benefits the system given its simplicity, is not a flawless method by any means. This approach means that two clients, with the same access levels to the same resource, may not both be able to access the resource. \*For example, if Client 1 was connected to Server 1 which has full server list, they may access a resource on Server 3, although if Client 2 was connected to Server 2 which does not have a current list, Server 3 will not be accessible by Client 2\*. This means that two current users will not see the same resource list. Although this will be completely transparent to the user, it is not a good practice for a distributive system.

**Other distributed systems challenges**

Performance – arrayList is way faster than Vector

With the current implementation of the system, to satisfy the requirements there have been sacrifices made to performance. Although utilizing an ArrayList to store resources would provide better performance, the development team has gone with the decision to use a Vector. At this stage, the use of Vectors provides us with more benefits than drawbacks. Multithreading was essential for this release as it solves several concurrency issues.

*Transparency*

The system has been developed with the intention off full client transparency. EZShare has location, failure, access and concurrency transparency to the client.

Failure Tolerance – should have to improve change

*Heterogeneity*

The data is not saved anywhere, if the server goes down, we lose the resource.