EZShare Resource Sharing Network

Team ALAS

**Ao LI**

aol3@student.

unimelb.edu.au**Lilian CHAN**

lilianc2@student.

unimelb.edu.au**Alston CHU**

[a.chu@student](mailto:a.chu@student).

unimelb.edu.au**Sheng WU**

shengw4@student.

unimelb.edu.au

# INTRODUCTION

EZShare is a distributed system for sharing resources, consisting of servers which can communicate with one another, and clients which can communicate with the servers. After resources are published or shared to the server, other users can send a query command to check what is available. The queries can propagate to all servers which are on the list of servers that exchanged IP and port numbers. Resources can also be removed from the server or fetched as a physical file. A ‘secret’ is set on the initiation of the server which is used when sharing files.

The technical challenges for building such a system are correct handling of server-client interaction, predicting operating system dispatch behaviour, managing multi-threading, ensuring data consistency for multiple clients and working around the limited processing power for each server. Through building such a system, some common problem for distributed systems, such as scalability and concurrency, can be explored and better understood.

# 2. SCALABILITY

Scalability refers to the ability of a system to maintain the same quality of performance despite the growing number of users and resources without requiring major redesign. In the EZShare system, a scalability challenge is encountered when there is a growth in the number of clients, so that there is an increase in simultaneous commands submitted to a server.

In the current architecture, a new thread is created to process each command that was received, where the maximum number of threads available is limited by the computing power of the server. Under a heavy load, large amounts of memory would be consumed. Resources are stored in the memory rather than the disk and there may be multiple copies of the file in the memory when several clients request for that resource. The constant context switching between threads would also cause significant losses of CPU time (Erb, 2012).

A revision that can be considered is to limit the maximum number of threads allowed for each server. On the other hand, a drawback would be the lowered amount of consecutive commands that can be received at once that would continue to limit the overall scalability of the system. Client requests are placed in a queue of serial tasks and users would need to wait longer for them to be processed.

The only method that would completely alleviate this problem would be to change the system architecture and add more servers so that tasks can be executed in parallel. For this to be effective, some control mechanisms need to be put in place so that loads can be evenly distributed across all servers. One such method is using a ‘Sticky IP’ (Kumar, 2013). The requests of the client could then be diverted to an available server using a load balancer. However, this would also add to the physical cost of the system.

Significant growth in the amount of resources shared could also affect the system’s performance. When the ‘query’ command is executed, more resources have to be inspected to find matches, decreasing performance speed. Other commands, such as ‘publish’ and ‘remove’, also require processing of existing records to find if there is a match that can be overwritten or removed.

A method in which this problem can be controlled would be to change the storing system of the files. Resources could be put into categories based on their content, similar to how resources are arranged in a library, when they are published by the user.

An increase in the size of the stored files also introduces delays in the ‘fetch’ time experienced by the client, further affecting scalability. To reduce the effects of this problem, mechanisms to divide the data into segments can be put into place.

# 3. CONCURRENCY

In the EZShare system, concurrency is considered to be achieved when all clients receive the same messages from the server and the results obtained are consistent. Additionally, the standard of performance should not decrease as a result of different clients performing tasks simultaneously. This presents as a challenge as operations performed by the server for different clients often interfere with the results of one another.

A concurrency issue demonstrated by the EZShare system is when there are multiple clients that request to fetch resources from a server simultaneously. The files are read from the disk to the memory each time ‘fetch’ is requested, causing a bottleneck in shared memory and decreasing the level of performance experienced by each client.

When multiple clients are editing the same resource, other strange behaviour may occur. The current system allows sharing of resources with the same primary key to overwrite the existing resource. If a client is fetching the resource while another client is attempting to remove or update it, the result would be random and unpredictable.

In order to eliminate these issues, some locking mechanisms or synchronisation blocks can be put on some blocks of code to ensure that certain commands that can interfere with one another cannot be sent at the same time. For example, while a client is requesting a ‘query’, other clients need to be blocked from ‘publish’, ‘share’ and ‘remove’ commands that would change the resource list obtained. Although this will prevent unexpected behaviour or retrieving outdated versions of the resource, it can lower performance by introducing unnecessary waiting times. The priorities between concurrency and availability need to be considered so that the right balance is found.

Inherently, the developing of code for parallel processes to run free of errors is a challenge in itself. Concurrency bugs are extremely hard to detect (Fowler, 2011), and the system could work correctly for a large majority of the time with only random unexpected behaviour.

In order to simplify this problem, many people choose to code the system in a way that completely isolates each process from the others. The drawback would be that this type of system would use a lot of resources. There would be some redundant work as some resources that can be shared by multiple processes would not be utilised.

# 4. OTHER CHALLENGES

Another challenge for the EZShare system is failure handling. It is impossible to create a system that would never fail, as faults may occur in the hardware or software, and sometimes incorrect results are produced. The challenge is for the system to detect the failure, mask it from the users, tolerate some failures and recover from them (Coulouris, 2012).

In the current system, the server has some mechanisms to detect invalid commands and send error messages. The client was also programmed in such a way to assist the server by not sending invalid commands that would cause failure. However, as the server is available in a distributed system, the identity of the clients that make contact cannot be confirmed, and many clients may not behave in such a polite manner. There may even be malicious attackers that seek to cause system failures.

Defensive programming needs to be incorporated into the server to handle such potential errors. This can be done by thoroughly considering anomaly cases and preparing the server to respond to those. Currently the problem is partially addressed by the error message system, where exceptions are detected and feedback is given to the user with some details. The ‘debug’ command line option is also available, that will show the user every message that was sent or received so that they can decide how to recover from errors.

An example of how the system was designed to mask failures is the silent removal of whitespace and the character ‘\0’ at the start and end of input strings, so that the commands can be run as normal.

Further considerations that can be incorporated into the system in the future to improve its performance against failures would be to have some recovery mechanism to ‘roll back’ and restore the state of data to before a server crash. Redundancy can be introduced so that servers frequently exchange information between one another, ensuring that data will remain accessible to clients even after one server has failed. Servers can also be taught to detect fault in their peers, so that when faults are detected in other servers, they can direct clients to other correctly working ones.

By addressing and overcoming all of these mentioned challenges, the advantages of a distributed system can be harnessed, creating a system with higher availability and reliability than a simple single-server system architecture.

# 5. REFERENCES

Coulouris, G. (2012). *Distributed Systems Concepts and Designs.* Boston: Pearson Education Inc.

Erb, B. (2012). *Concurrent Programming for Scalable Web Architectures.* Retrieved from http://berb.github.io/diploma-thesis/original/042\_serverarch.html

Fowler, M. (2011). *Patterns of Enterprise Application Architecture.* Boston: Pearson Education Inc.

Kumar, A. (2013, August 12). *Scalability: Challenges and Solutions for Enterprise Applications.* Retrieved from Software Architecture, Development and Data Science: https://itsabhishekkumar.com/2013/08/12/scalability-challenges-solutions-for-enterprise-applications/