COMP90015 Distributed System
2019 Semester 1 Project 1

Group Name: **Not Safe For Workshop**

Member Name	**Login Name**	**University Email**
Yujun Zhang	yujuzhang	yujuzhang@student.unimelb.edu.au
Tong Su	tsu2	tsu2@student.unimelb.edu.au
Zhenyu Wang	zhenyuw6	zhenyuw6@student.unimelb.edu.au
Yu Bai	ybbai1	ybbai1@student.unimelb.edu.au

Introduction

What the project was about

- This project was to implement a file system for Bitbox peers to share and modify files in a given directory. This was monitored by File System Manager and an unstructured P2P network was formed for peers to share, and monitor changes in the directory. Peers observed and got notified when changes happened in the share directory such as directory creation or deletion, file creation, deletion and modification.

What were the technical challenges that you faced building the system

- One of the technical challenges that we faced in developing this system is to handle the case when different peers send or try to modify the same file at the same time with different content. This would cause data corruption.
- Another technique issue is the conflict between threads as we use many different threads to deal with different tasks and also dynamic threads which could automatically create and destroy threads.

What outcomes did you achieve

- We implemented this Bitbox file system for Bitbox peers to create, delete and modify files and directories in the shared directory which will automatically notify all connected peers and do the same instructions to their files.

Fault detection/tolerance/recovery

Identify sources of faults that can have the greatest impact on the system, i.e. to disrupt it from working as intended

- **Failure of hardware or internet** - This kind of fault may occur when two peers are sharing or transferring files bytes and one of the computers shuts down or some hardware issues caused the disconnection between these two peers so that the transmission disrupt.

- **Data corruption** - Data corruption may occur if different peers are sending the file bytes to the same peer at the same time, and this will cause the system receiving and writing down the file bytes to the wrong file because of the transmission order and speed.

- **Command conflict** - if two peers happened to send handshake request to each other at the same time, this is a kind of concurrency faults and in order to establish the connection, peers need to send the handshake request and the received peer has to send back the handshake response, it is possible that they both send the handshake request before they receive the other's request. And conflict would occur if they both deal with the request and try to send the response to each other.

Describe how the system attempts to detect/tolerate/recover from these faults, if at all Suggest revisions to the protocol that may overcome these problems

- For the **failure of hardware or internet**, the system could detect this kind of error if the bytes request or response were sent but the did not receive the response for a long time, or it is not able to send to the connected peer. And the system will need to close the connection and delete all incomplete file that has been read and write. The system will attempt to tolerate this fault by reported the error saying that the connection needs to be reset. Also, the system will attempt to recover the connection when the disconnected peer is back and will send the handshake request again the other peer. The system will automatically clear hash values for a period of time in order to deal with the disconnection failure so that the disconnected peer could reconnect by resending the handshake request.

- For **Data corruption**, this could be detected by checksums and the system use multi-thread to use one certain thread with one file bytes transmission so that no file bytes will be corrupted. This means FILE_BYTES_REQUEST and FIL_BYTES_RESPONSE protocol for each pair of peers that want to transfer files would be processed in a separate thread.

- For **command conflict**, In HANDSHAKE_REQUEST and HANDSHAKE_RESPONSE protocols, the system will also use a different thread for receiving and sending, to avoid command conflict. And in this case, if two peers still sending connection request together, this fault will be tolerated and reported to each peer.

Scalability

Identify aspects of the system that present problems for scalability

- **Cost of physical resources Problem**: In this system, the number of peers that can connect with one user at the same time should be controlled. If this number overload, the host might overburden, that is, the host does not have enough resource to provide support to maintain the transformation.

- **Controlling the performance loss Problem**: In Peer to Peer System, there might be many files in transfer at the same time, the congestion of the transformation might happen. If there are a large number of files, transfer in order might cause a lot of time, especially if some file stuck and needs retransmission.

- **Resources should not run out Problem**: The buffer for transfer files should be set rationally. At first, we decided to set buffers for file sending and file receiving. But this might cause the problem that a very huge file might run out all the buffer resources and cause some subsequent problems, like transfer aborted, file byte receives incomplete.

- **Avoiding Performance bottlenecks Problem**: If the architecture of the system is centralized, the system might have performance bottlenecks because the central server needs to support all the clients' activity. And the central server will be hard to maintain.

Suggest revisions to the protocol that may overcome these problems

- **Revision for the cost of physical resources Problem**: To overcome this problem, our system limits the number of visiting users for a host, which the maximum number is 10. But the number of other peers that this host can access has no limitation. Besides, each peer should have the same capacity in the number of connections.
- **Revision for controlling the performance loss Problem**: Currently, the number of files transfer at the same time has no limitation. The file byte size is limit by the size of each block, which can be set manually.
  To get the higher speed of transmission, our system decides to no transfer file byte one by one in order. The method is each connection will create a thread, and the thread can create many threads to handle the transmission of multiple files. One file responds for one thread. All the file bytes' transmission is independent without sequencing. After the file transfer successfully, the thread used for transferring this file will be closed immediately. Hence, in our system, the transmission bandwidth can be used in utmost. The time for all files delivery successfully will faster than transfer in order.

- **Revision for resources should not run out Problem**: We decided not to set buffer for cache file byte on both sides. File receiving and file delivery will process directly on the hosts' machine. For the comments, this system arranges a queue buffer to store comments so that each comment can be executed one by one to avoid conflict.

- **Revision for avoiding Performance bottlenecks Problem**: The Peer to Peer system we built is using a decentralization architecture, each process in the system plays a similar role, that is, play both client and server at the same time. The system does not need any centralized server. Any Peer host breakdown will not affect other peer's activity.

Security

Discuss the security issues surrounding the reading and writing of files/directories to a file system in the kind of P2P system

- **Security Issues**: The most significant security issue in Peer to Peer system is the file send and file receive should not processing without permission. If an unauthorized peer has the accessibility to read and write files, it might cause a series problem. For example, some confidentiality problems such as information leakage or attack by computer virus; some integrity problems like the local file being maliciously modified; some availability problems like interference access.

How does the current system, including considering the File system Manager implementation and the protocol specification, address these security problems

- **Address security problems**: In our current Peer to Peer system, each connection needs to build the HANDSHAKE_REQUEST first, only when the peer in the other side accepted the request and the number of connections is not over the maximum connections limitation, the handshake will be accepted. Before starting to send file bytes, the FILE_CREATE_RESPONSE must be sent first. When the create response successfully accepted, then the system can start to send FILE_BYTES_REQUEST and

FILE_BYTES_RESPONSE. Considering in our system, each file transfer will respond to an arranged thread, if the same file request sends repeatedly in this connection when the file is transferring, this request will be ignored. For the situation when one side is suddenly losing connection, the other side will get the information about socket denied visiting. (The information will show "connection reset"). Every a period of time, the system in online Peer will check all the connections recording in a Hash table, removes all the dead connections. Those offlines Peers need to restart handshake to set up a new connection. For the incomplete receiving file, the host will detect it and delete it from the local machine.

What more could be done to address any outstanding security problems in this regard

- For example, some file the Peer wants to share with other Peers, but do not want the file to be modified by others. Besides, a filter file function can be built to avoid receiving files include harmful information for the system.

Other Distributed System Challenges

Choose a third distributed system challenge that related to the system and explains how it relates, how the system currently addresses the challenge (if it does at all), and how you might change the system to improve it with respect to the challenge.

- Another distributed system challenge is **concurrency**. This relates to this system as it forms an unstructured P2P network which means each peer could connect with many peers under the connection limitations and they all can access and modify directories and files. So it needs to handle concurrency.

- As all connected peers could access and modify files in the share directory, if multiple peers modify the same file at the same time, this could cause error and conflict. The current system uses multi-thread to handle different sockets and separate all connections and their messages and indications. This could use a different thread to deal with the modification instructions. As at the basic writing level, only one of the threads could call the writing method, it is actually accessing and processing sequentially to avoid conflict. In this case, the modification time matters.

- Also, the current system did not have much access control and all peers that show in the configuration. Properties file could connect and modify anything they want in another peer's local directory. This approach is not secure enough as other peers could do something bad to the current peer's local machine. Also, if one peer accidentally deleted something important, all connected peers' share directory would lose the same file.

- If the maximum number of connections grows, the system has to handle a large number of connections and messages. So, concurrency is very important to improve the performance of the system. In order to make it better for all peers to share files and resources, all resources under this system have to be designed to be safe enough to handle concurrency. First, the system could improve the access control to make sure each instruction from other peers is safe and secure enough to process to avoid conflicts and improve security for the system environment. Also, to avoid concurrent processes by making access sequential. Another way is to use timestamp for each message or instructions. For example, if two peers both modify the same file, from the timestamp current peer could compare the time and only need to process the later one to improve the efficiency of making it sequential.