Contents

1. System Introduction

In today's world of distributed systems, gadgets and networks, individuals with access to intranet and internet have need to copy the files from one device to another device (specially from Laptops to Desktops, from Desktops to Servers, etc.). Peer to Peer File Sharing System is a cost effective console based solution that enables the file copy process easier. It provides:

- A distributed / decentralized database where users can register their files (Indexing Server)
- Other users can actually search through this file database and download the files as per their requirement
- A peer based log file where operations done by client can easily be recorded

Some of the features available in the current System are,

- **Easy to use console based interface**
  A console based interface allows user to select through various operations (Register, Lookup, Download, Unregister, etc.). Depending upon the operation selected by user, System prompts for the required inputs (like File Name, File Path, IP Address, etc.) This helps users to opt to work selected files rather than all or a set of files.

- **Highly parameterized system**
  The ".properties" files allows administrator or user to define the various options (like IP Address, Port, Default Sharing directory, Default Download directory, etc.) for keeping the System scalable and parameterized.

- **File Coverage**
  System supports both Text as well as Binary Files. This facilitates users to share their professional as well as personal data (like multimedia files, photos, etc.) easily.

- **TCP Support**
  System uses TCP / IP Protocol to ensure that the data is downloaded with consistency. The reasons for choosing TCP / IP over UDP are listed below.
  - TCP is connection oriented – once a connection is established, data can be sent bidirectional.
  - It is also widely used by other protocols like HTTP, HTTPS, FTP, SMTP, and Telnet which makes this protocol widely popular for Net Traffic.
  - TCP is more reliable since it manages message acknowledgment and retransmissions in case of lost parts. Thus there is absolutely no missing data. UDP does not ensure that

communication has reached receiver since concepts of acknowledgment, time out and retransmission are not present.

- TCP transmissions are sent in a sequence and they are received in the same sequence. In the event of data segments arriving in wrong order, TCP reorders and delivers application. In the case of UDP, sent message sequence may not be maintained when it reaches receiving application. There is absolutely no way of predicting the order in which message will be received.
- It has its own Error checking mechanism ensuring the data order and consistency.

- **Distributed Indexing Server**

  Distributed Indexing Server is used to balance the load. Multiple Indexing Servers help to support a larger File Database with lesser load on each Indexing Servers. It also helps to ensure minimal turnaround time (TAT) ensuring better performance for Register and Search operations.

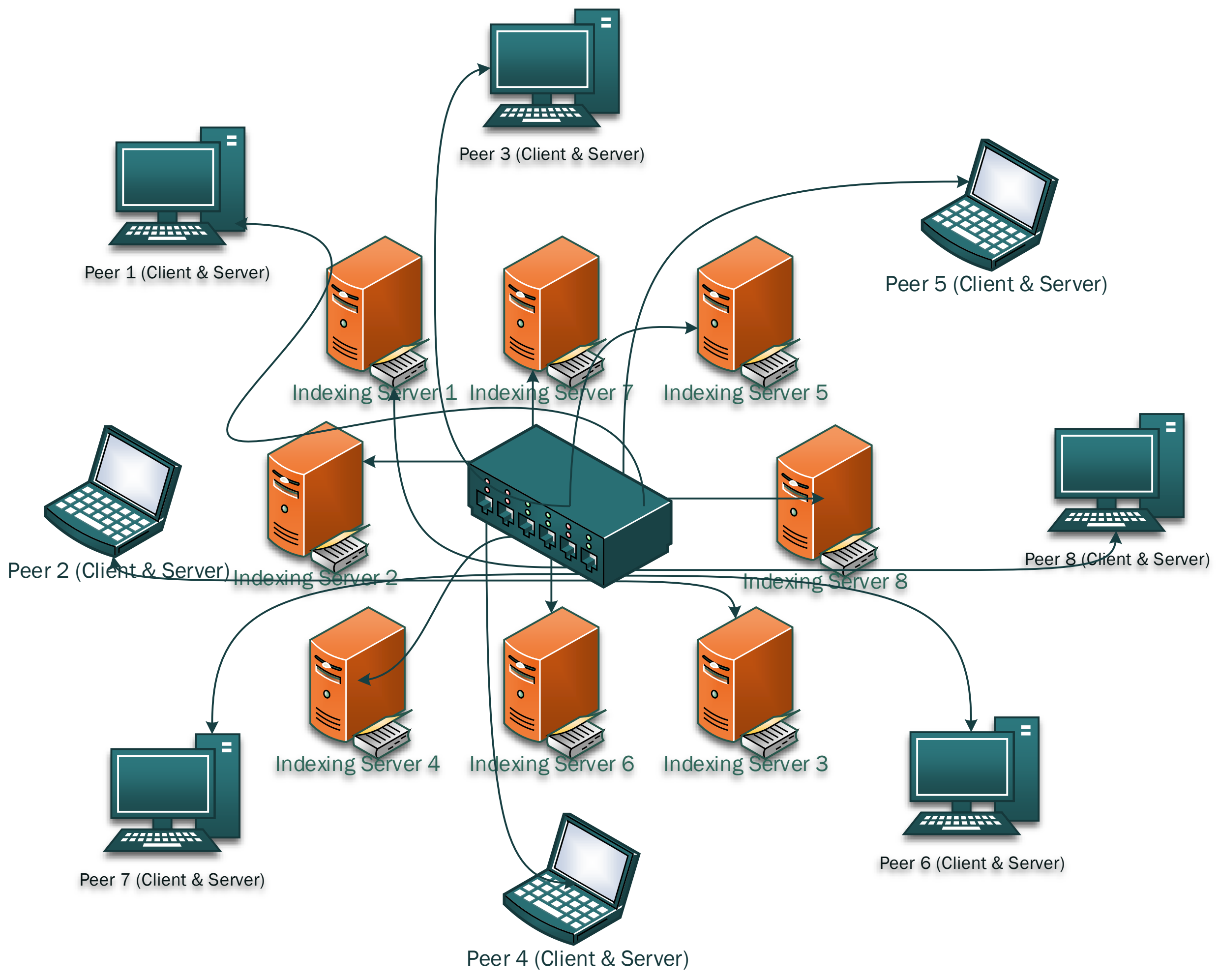
- **Data Resilience**

  Maintaining multiple Indexing Servers also facilitates the replicas of other Indexing Servers' data to achieve Data Resilience. Data Resilience in turn works as fail-safe mechanism ensuring 100% operational time and data availability.

  This decentralized architecture also helps to maintain Data Resilience for P2P Files (actual data to be shared) by keeping a copy of these files on Indexing Servers.

  Number of copies to be maintained for Data Resilience (either for Indexing Servers' or Peer Servers' data) depends upon the Data Resilience factor (intReplicationFactor parameter in dcp2p.properties file). Data Resilience Factor can be parameterized and can have maximum value '***(n-1)***' (**excluding the data from the original Index / Peer Server**) where '***n***' is the number of Indexing Servers used by Peer to Peer System.

2. Architecture



The image above clarifies the architecture for the Peer to Peer File Transfer System with Distributed Indexing Servers.

3. Indexing Server

a. Process

Indexing Server accepts the data from various Peer Client using Socket and Multithreading concepts. As soon as Peer Client connects to Indexing Server, there is a new thread created on Indexing Server and this newly created thread is then taken over by the Indexing Server thread handler in order to fulfill Peer Client's request. The connection between Peer Client and Indexing Server is released once the operation is completed with either success or error.

Indexing Server uses ConceurrentHashMap structure from Java to create an Indexing Server database.

It performs the operations like Register, Search, and Unregister on Indexing Server database based on the inputs received from Peer Client.

In addition to all the above mentioned operations, Indexing Servers also ensures the Data Resilience by replicating the individual Indexing Server database and Files on other Indexing Server. Data Resilience is handled by Peer Client but managed on Indexing Servers by means of the Data Resilience Factor.

b. Tradeoffs made

- Have used ConcurrentHashMap instead of SynchronizedMap. The former one locks a particular bucket or segment to which it is writing whereas the later one locks the complete Map object while writing or reading. But the later one ensures 100% data consistency in case of highly concurrent operations but lacks speed. It is exactly reverse case with ConcurrentHashMap
- Avoided Log Files Indexing and Peer Servers
- In order to ensure performance reduced the number of messages on console
- Avoided detailed Log File writing for Peer Client in order to reduce the IO cost and the File Size

c. Possible Improvements

- Usage of Thread Pool
- Better exception handling

- More focus on Synchronization in order to avoid Data or output inconsistency due to use of ConcurrentHashMap
- Testing on a large scale environment
- Focus on Log File writing in order to keep track of the operations performed on Indexing and Peer Servers

4. Peer Server

a. Process

Peer Server accepts the data from various Peer Client using Socket and Multithreading concepts. As soon as Peer Client connects to Peer Server, there is a new thread created on Indexing Server and this newly created thread is then taken over by the Peer Server thread handler in order to fulfill Peer Client’s request. The connection between Peer Client and Peer Server is released once the operation is completed with either success or error.

Peer Server uses is used to download the required file.

b. Tradeoffs made

- Avoided Log Files Indexing and Peer Servers
- In order to ensure performance reduced the number of messages on console

c. Possible Improvements

- Usage of Thread Pool
- Better exception handling
- More focus on Synchronization in order to avoid Data or output inconsistency due to use of ConcurrentHashMap
- Testing on a large scale environment
- Focus on Log File writing in order to keep track of the operations performed on Indexing and Peer Servers

5. Peer Client

a. Process

Peer Client accepts the data from user and performs various operations accordingly. In addition, it also prompts to user for additional inputs depending upon the operation selected. Peer Client uses Socket to communicate with Indexing Server and Peer Server.

It performs the operations like Register, Search, and Unregister on Indexing Server database based on the inputs received from Peer Client.

b. Tradeoffs made

- User console based input is disabled for automation of Testing
- In order to ensure performance reduced the number of messages on console
- Avoided detailed Log File writing for Peer Client in order to reduce the IO cost and the File Size

c. Possible Improvements

- Possibility to check for multithreading client
- Better exception handling
- Testing on a large scale environment

6. Additional features

Following are some of the additional features Peer to Peer System implemented.

a. Client based log file creation
b. Supports for both text and binary files
c. Console based interface
d. Usage of Properties file to read various parameter
e. Parameterized Data Resilience