Drizzle: peer-to-peer file distribution system

High-Level System Description

The p2p system consists of a metadata server that stores the peer's information, files shared, and their chunk locations and hashes in the system. Files are divided into fixed size chunks that enable a peer to simultaneously download multiple parts of a large file from multiple peers. The progress for file download is also shown for each peer.

Peers request information from the metadata server for things like file lists and contact the peers directly to download the actual file. Both, the peers and the server are capable of handling multiple requests at once. The peer may spawn multiple threads to download chunks from separate peers. The server may spawn multiple threads to accept incoming requests from different peers.

Protocol

The protocol starts with peers registering themselves and the initial set of files they wish to share along with the chunk information like hash to the metadata server. Then, they are free to request the current file list in the p2p system and request any one of them. It gets the chunk-level information for the file and sorts them in rarest-first order and parallelly sends download requests. Amongst multiple peers serving a particular chunk, a random peer is selected to ensure there is not a lot of load on any one peer. Once it gets all chunks it assembles the file in its local directory after checking hashes (or reject the file on hash mismatch(es)). The hash is compared to the first calculated hash value stored with the server.

Anytime a new file chunk is downloaded by the peer, it will also invoke the metadata server to register this new available location for that chunk, allowing another peer who requests it to download from here at a later point of time.

Program Structure

The server has its class "server.py" with all the APIs it exposes. A server object is instantiated before starting any peer. The peers are an instance of "client.py" class with functionalities of peers. Clients take arguments of the port on which they want to be able to upload and the file directory to share. Although clients can take in command line arguments, we pass them in constructor arguments for the sake of demo. Then there is the "utils.py" for common utilities used both by peers and server such as data structures and classes for requests and responses. All these are in the "src" directory.

For testing, there is another directory "test". This contains 3 example scenarios for the demo of the system working.

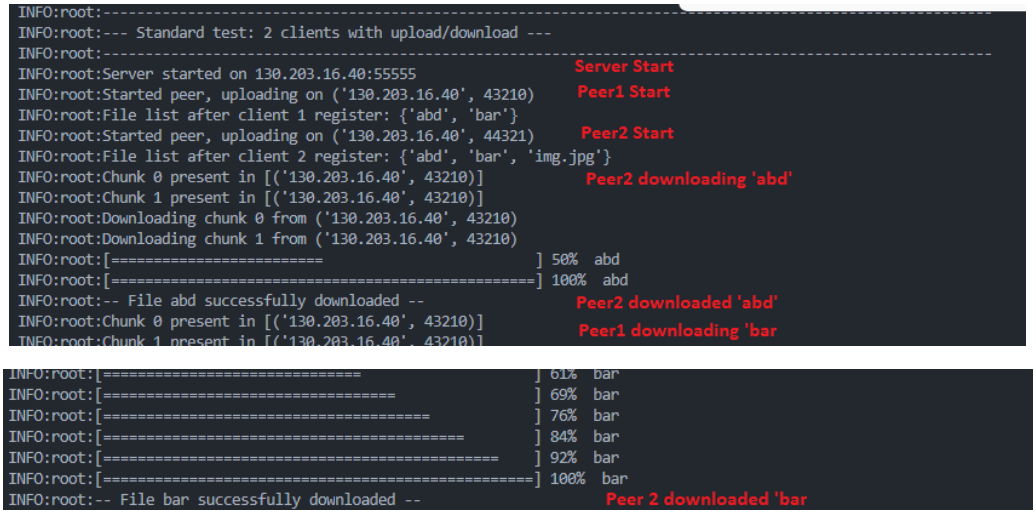
1) The first example is the standard test with 2 peers, 1 server. Peer1 uploads files A, B and Peer2 uploads file C. Then they proceed to download the files they don't have from each other.
2) The second example checks the integrity with 2 peers, 1 server. Peer1 uploads files A with hash and stores at server. Peer2 requests information on A and gets hash value for each chunk. When it tries to download file A, Peer1 has modified A to A'; hence Peer2 rejects this file download.

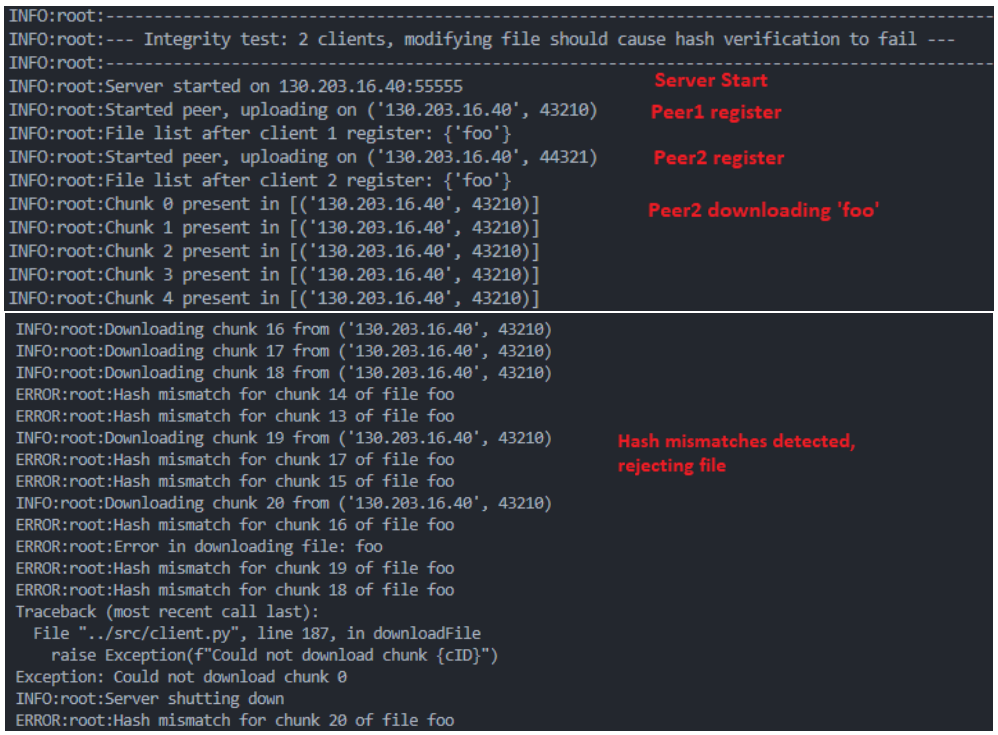
3) The third example shows rarest first chunk download with 3 peers, 1 server. Peer1 uploads file A with all chunks. Peer2 has downloaded and registered only half of the chunks (we simulate this through a parameter) of A. Peer3 gets information on A from the server and sees half the chunks contain only Peer1 serving, while other half are served by both Peer1 and Peer2. So, it prioritizes downloading chunks containing only Peer1.

Working Parts

All basic requirements of the assignment- Multiple connections, parallel download, chunk registering, integrity check is working. All the prescribe APIs are present in the server along with appropriate responses and replies.

Outputs





```
INFO:root:----------------------------------------------------------------------------------------------------
INFO:root:--- Rarest first test: 3 clients, one with all chunks, one with partial chunks, one downloader ---
INFO:root:--- 3rd client should download chunks based on rarity ---
INFO:root:----------------------------------------------------------------------------------------------------
INFO:root:Server started on 130.203.16.40:55555
INFO:root:Started peer, uploading on ('130.203.16.40', 43210)
INFO:root:Started peer, uploading on ('130.203.16.40', 54322)
INFO:root:Chunk 0 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 1 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 2 present in [('130.203.16.40', 43210)]
```

Server started

Peer1, Peer2 up

Peer2 downloading 'foo'

```
INFO:root:[============================================     ] 90%  foo
INFO:root:[=============================================    ] 95%  foo
INFO:root:[============================================== ] 100%  foo
INFO:root:-- File foo successfully downloaded --
INFO:root:Started peer, uploading on ('130.203.16.40', 44322)
INFO:root:Chunk 0 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 1 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 2 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 3 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 4 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 5 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 6 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 7 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 8 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 9 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 10 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 11 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 12 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 13 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 14 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 15 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 16 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 17 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 18 present in [('130.203.16.40', 43210)]
INFO:root:Chunk 19 present in [('130.203.16.40', 43210), ('130.203.16.40', 54322)]
INFO:root:Chunk 20 present in [('130.203.16.40', 43210)]
```

Peer2 downloaded 'foo'

Peer3 downloading 'foo'

Even chunks present only in 1 client

```
INFO:root:Downloading chunk 0 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 2 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 4 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 6 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 8 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 10 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 12 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 14 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 16 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 18 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 20 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 1 from ('130.203.16.40', 54322)
INFO:root:Downloading chunk 3 from ('130.203.16.40', 54322)
INFO:root:Downloading chunk 5 from ('130.203.16.40', 54322)
INFO:root:Downloading chunk 7 from ('130.203.16.40', 54322)
INFO:root:Downloading chunk 9 from ('130.203.16.40', 54322)
INFO:root:Downloading chunk 11 from ('130.203.16.40', 54322)
INFO:root:Downloading chunk 13 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 15 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 17 from ('130.203.16.40', 43210)
INFO:root:Downloading chunk 19 from ('130.203.16.40', 43210)
INFO:root:[==                                              ]  4%  foo
INFO:root:[====                                            ]  9%  foo
INFO:root:[=======                                         ] 14%  foo
INFO:root:[=========                                       ] 19%  foo
INFO:root:[===========                                     ] 23%  foo
INFO:root:[==============                                  ] 28%  foo
INFO:root:[================                                ] 33%  foo
INFO:root:[==================                              ] 38%  foo
INFO:root:[====================                            ] 42%  foo
```

Prioritizes Even chunks since they are rare