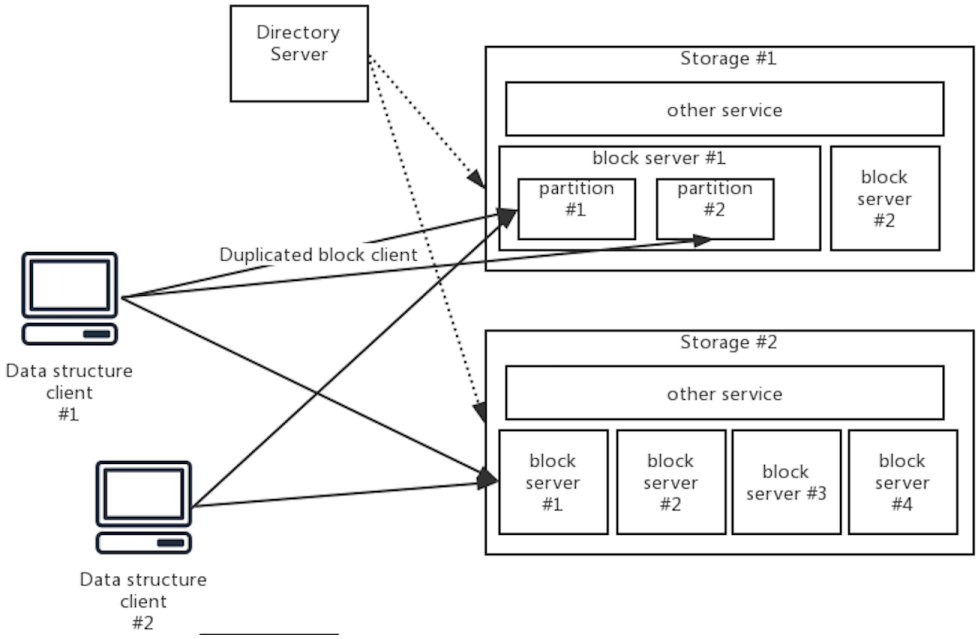
ConnectionPool design in jiffy

Problem to solve:

- Opening and closing new connections has become the bottleneck of jiffy data process performance.
- Connections are reusable, e.g. partition from the same block server(but with different block_id) should be able to share the same connection.

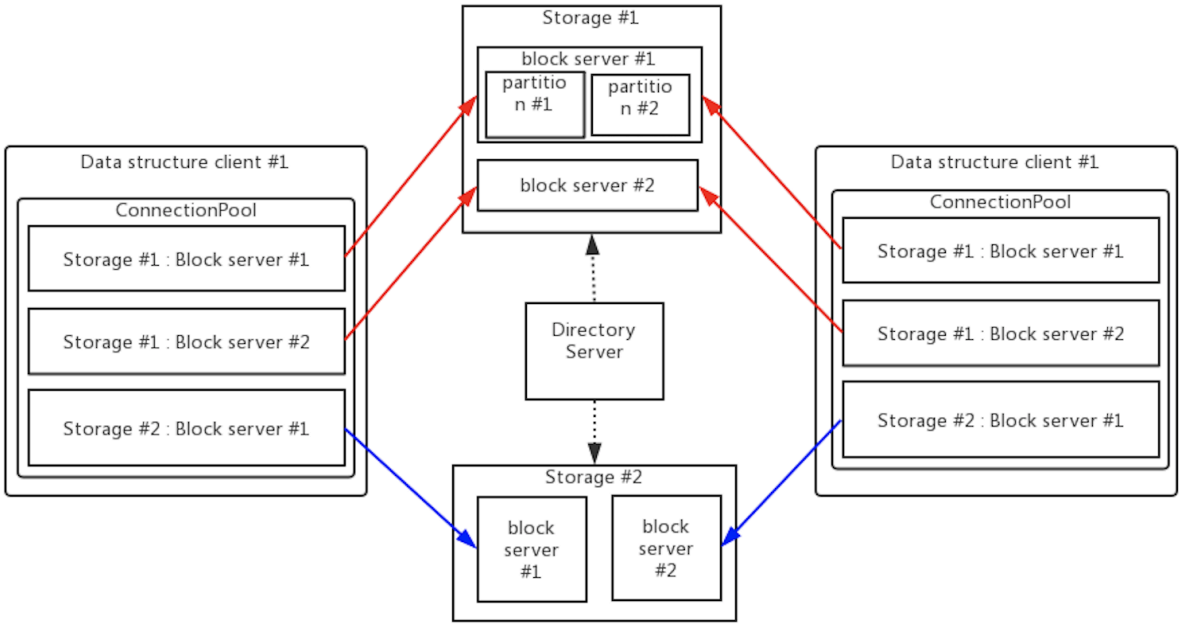


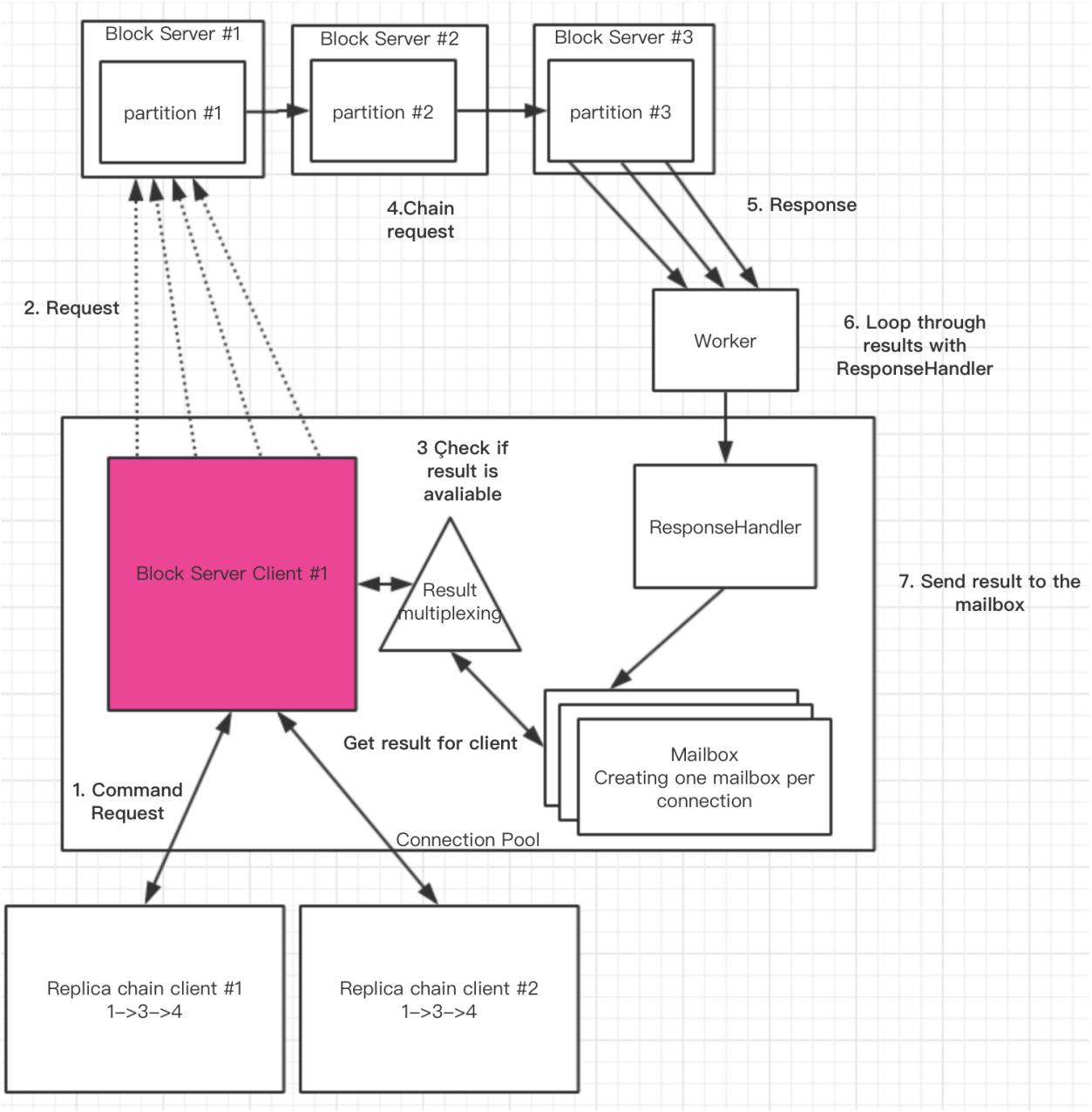
Requirements

- Cache connections from each client to the block servers to avoid adding new connection while refreshing the blocks and redirecting operation to new blocks.
- Correctness: Make sure that the correct client receives the correct response.
- Garbage collection: Tracking of “lost” connections or “idle” connections which were not returned back to the connection pool and claim them back to the connection pool.

Proposal:

- Define a connection_pool class for handling connection with each single block server at the jiffy client, one per client. The Connection Pool should maintain the connections with each block server, which consist of multiple block partitions. A maximum pool size is set, which defines the maximum number of connections from the client to one block server. After the connection number exceeds the pool size, the client has to be blocked and wait for a connection to be ready.
- The connection_pool should be multithreaded among all data structure clients and should be able to handle concurrent request and release of connections using locks.
- The connection_pool should add connections every time it sees a new block_server(determined by the service port). It doesn't need to add connections the the block server at the very beginning since the client usually just needs a small portion of the blocks.
- Whenever a data structure client is constructed, it will issue multiple request_connection(host, service_port) to get the connection instances to the block servers. This happens concurrently and concurrency issues should be handled.
- After a client changes its blocks due to auto scaling, it needs to get the new connections with request_connection(host, service_port) and remove the old not used connections with release_connection(host, service_port).
- Check when a connection fails to send a request/fetch a response, and refresh the connection at that point.
- Response handling: Use one worker thread to iterate through all connections and see if the response arrives.(Using the same logic as data structure listener).The response handler would push the results to the correct mailbox and the client would block until the response arrives at the mailbox.




Block Server #1
partition #1
Block Server #2
partition #2
Block Server #3
partition #3
4.Chain request
5. Response
2. Request
Worker
6. Loop through results with ResponseHandler
3 Çheck if result is avaliable
ResponseHandler
Block Server Client #1
Result multiplexing
7. Send result to the mailbox
Get result for client
Mailbox
Creating one mailbox per connection
1. Command Request
Connection Pool
Replica chain client #1
1–>3–>4
Replica chain client #2
1–>3–>4

To be determined after benchmarking:

For connection multiplexing, three solutions are possible:

1. Creating multiple connections between the same block server and data structure client. Each replica chain client registers one connection and uses it until the connection is closed or the client is destroyed.
   Pros:
   Multiple partitions on the same block server could be accessed by multiple replica chain clients, also the response mapping would be simple since only one replica chain client uses this connection at a given period of time.

   Cons:
   - still lots of connections - 10 servers * 4 block server/ storage server * 4 connection each block server = 160 connections. Start up time may become a new bottle neck for so many connections.
   - Replica chain request might be blocked due to all connection in use.
   - Needs locking when client register connections.
2. Creating one connection between the same block server and data structure client only. All the replica chain client with the same block servers are multiplexed under this scenario, which we need to make sure that the response is in order and responded to the correct client.
   Pros:
   Least possible numbers of connections - 10 servers * 4 block server/ storage server = 40 connections

   Cons:
   Multiplexing logic is a little bit complicated and ordering might hurt the performance.
3. Hybrid.

Current status:

1. Implemented the connection pool without locks, i.e. only supports one data structure client(one client access one unique connection for each replica_chain_client.
2. Pull request of Jiffy updated -> whether add a front() operation for queue? -> problem: if multiple producers and consumers, issuing a dequeue would lead to lost of data. But if dequeue is using to both remove and read, connection will fail if data size is large.

Questions:

1. block_client->get_command_response_reader without handler

https://github.com/charles-typ/jiffy/blob/957d600e4113e13c2c0c9b8e3c32eac8623e7fa2/libjiffy/src/jiffy/storage/client/block_client.cpp#L63

And `processor_->process(protocol, protocol, nullptr)`
https://github.com/charles-typ/jiffy/blob/957d600e4113e13c2c0c9b8e3c32eac8623e7fa2/libjiffy/src/jiffy/storage/notification/notification_worker.cpp#L30

2. Application relevant.

Next step:

Finish the whole implementation by this weekend(if lucky)

Keep up with Hong's work