Distributed Replicated Files Protocol Specification

Sining Ma
sma87@stanford.edu

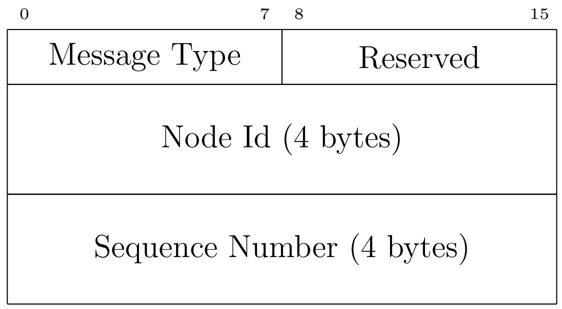
May 26, 2014

1 Protocol Specification

This section describes all message packets syntax, semantics. All messages bytes are in network byte order.

1.1 Common Header

All packets start with the same header structure.



Message Type The type of Message which is defined in this design.

Reserved This field is reserved and should always be zero.

Node Id This field represents the unique id for a client or server. Will talk more in Client and Server Id generation section about how this id is generated.

Sequence Number Each outgoing packet contains a monotonically increasing sequence number. This number wraps back to zero when it overflows.

1.2 Init (Message Type 0xC0)

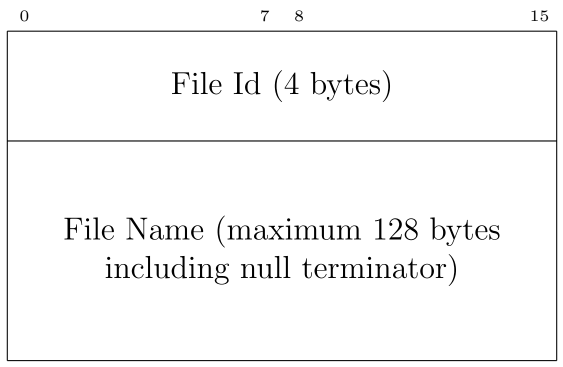
The client sends Init message when InitReplFs() is called to detect the servers.

1.3 InitAck (Message Type 0xC1)

Servers send InitAck messages when the server receives Init message to inform the server's existence. Put the server's id in node id field.

1.4 OpenFile (Message Type 0xC2)

The client sends OpenFile message to servers when OpenFile() is called to open a new file on local client file system. Servers open the file based on mount directory and filename.

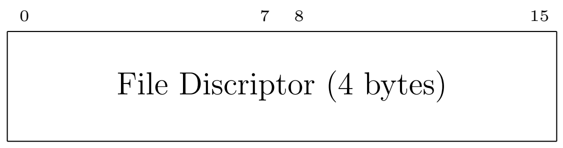


File Id File Id generated at client side. This Id is a monotonically increasing number.

File Name File name is OpenFile() argument.

1.5 OpenFileAck (Message Type 0xC3)

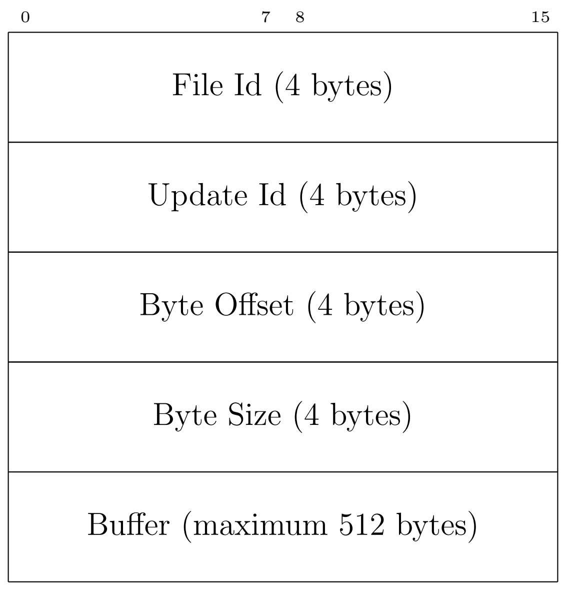
Servers send OpenFileAck messages to acknowledge OpenFile message. Return 0 if no error. If the server is unavailable or error happens, -1 returns.



File Descriptor File descriptor that is used at server side. Returns -1 if error happens, and 0 if no error.

1.6 WriteBlock (Message Type 0xC4)

The client sends WriteBlock message when WriteBlock() is called to stage a contiguous chunk of data. No Ack message will be sent from servers. Before one commit, the client can do multiple WriteBlock call to update the file. Update Id is used to mark update sequence id before commit call.



File Id File Id for the opened file at client side.

Update Id In one commit, current update id is a monotonically increasing number from 0. Reset back to 0 after one commit.

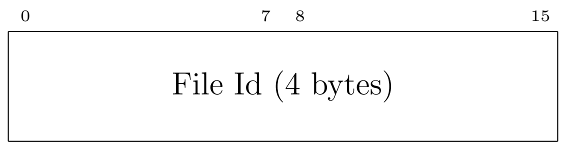
Byte Offset Byte offset the offset within the file to write to.

Byte Size Byte size the number of bytes in buffer.

Buffer Buffer a pointer to the data to be written.

1.7 Vote (Message Type 0xC5)

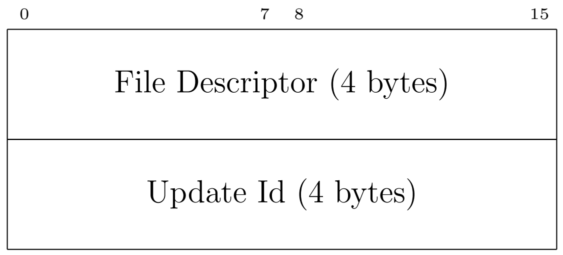
Two phase commit is used to ensure that all updates in WriteBlock are received on servers. The client sends Vote message as the first step in commit request phase to check if all servers receive all updates and are ready to do final commit.



File Id File Id for the opened file at client side.

1.8 VoteAck (Message Type 0xC6)

Servers send VoteAck message with an update Id which indicates the next Update Id that the server needed. When the client receives this message, the client checks if update Id in this message is identical to the total update count. If update id does not match, the client will do message retransmission. If VoteAck message returns -1 file descriptor, abort message will be sent. Will talk more about vote and retransmission in protocol operation flow.

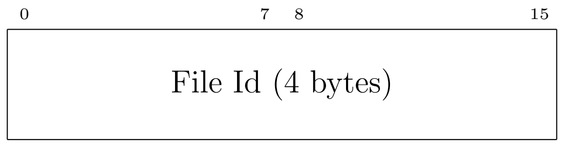


File Descriptor File descriptor that is used at server side. Returns -1 if error happens at the server side, and 0 if no error.

Update Id The next update id that is needed on the servers.

1.9 Commit (Message Type 0xC7)

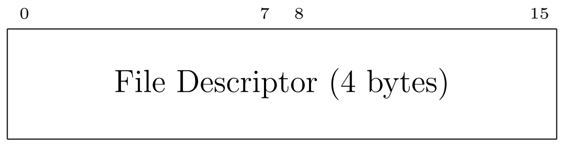
The client sends Commit message to all servers to commit all updates to the file.



File Id File Id for the opened file.

1.10 CommitAck (Message Type 0xC8)

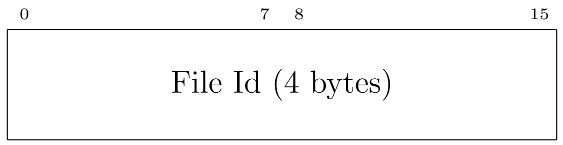
Servers send CommitAck message to tell the client which updates have been received.



File Descriptor File descriptor that is used at server side. Returns -1 if error happens at the server side, and 0 if no error.

1.11 Abort (Message Type 0xC9)

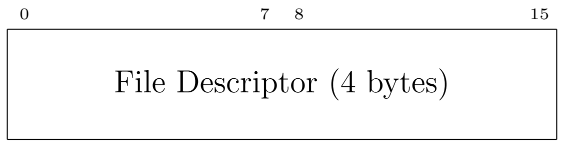
The client sends Abort message to all the servers when receive VoteAck message with -1 file descriptor or client wants to do rollback. All servers undo this transaction and rollback file updates.



File Id File Id for the opened file at client side.

1.12 AbortAck (Message Type 0xCA)

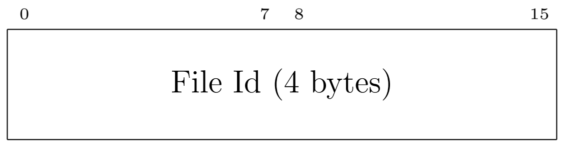
Servers sends AbortAck message to the client. This message informs the client that the server completes file rollback.



File Descriptor File descriptor that is used at server side. Returns -1 if error happens at the server side, and 0 if no error.

1.13 Close (Message Type 0xCB)

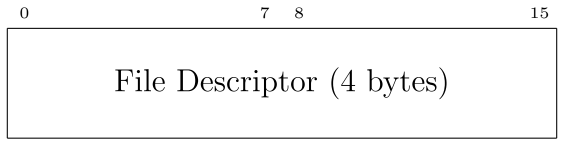
The client sends Close message to the servers to close the opened file.



File Id File Id for the opened file at client side.

1.14 CloseAck (Message Type 0xCC)

Servers send CloseAck message to indicate the file is closed.



File Descriptor File descriptor that is used at server side. Returns -1 if error happens and 0 if no error.

2 Protocol Details

2.1 Client and Server Id generation

Client and server Ids are generated randomly as a 32-bit integer with a good seed for number generator. The probability of conflict is 5.32101e-20. This is negligible. In real world settings, each client or server is assigned an unique identifier, e.g. UUID.

2.2 File Id generation

File Id on the client side is a monotonically increasing number starting from 0.

2.3 Protocol Operation Flow

- On InitReplFs(), the client sends init message to the multicast group to detect if there are enough servers in init timeout time. If responding servers count is smaller than numServers, the client returns error. Otherwise, if there are more servers, the client selects the first numServers, and only accepts acknowledge messages from these selected servers.

- On OpenFile(), the client sends OpenFile message to servers to open a file and waits for OpenFileAck messages.

- OpenFileAck message is sent from the servers. If file Id is -1, this means that OpenFile fails and the client returns error. If response selected servers count is less than init phase servers count, the client returns error. The assumption is that there is only one client running, so no need to cache client id on servers.

- On WriteBlock() sends WriteBlock file update message to the servers without acknowledge. Client caches all the updates before one commit. Each server caches all the received updates before receiving commit messages.

- On Commit(), the client sends vote message to all servers, and waits for vote response. Servers receive vote message. Servers store the next needed update id. Servers check from the latest valid update number one by one to see if the next update is received. If the next id is not received, servers send VoteAck message with the next needed update Id. If all updates received, servers still send the next update Id. Client must make sure the returned update id matches all update count. Any error happens on the server side, VoteAck message returns -1 file descriptor.
  Give one example. The client sends update 0, 1, 2, 3 to the servers, so the client total updates count is 4. If the server receives update 0, 1, 3, but 2 is lost, this server's next update id is 2. VoteAck message returns not -1 file descriptor, and next update id is 2. Client will do retransmission.

- Retransmission. Client receives VoteAck messages after receiving all servers VoteAck messages or timeout. Client reads all update ids and finds the smallest update id. Client starts to do retransmission from the smallest update id to all the remaining WriteBlock. Servers update received update cache. Client sends vote message again when all remaining WriteBlock is done.

- When Client receives all servers VoteAck message with update id matching servers total update count. Client sends commit message to complete this transaction. If client receives VoteAck message with -1 file descriptor, client sends Abort message. Any server timeouts for VoteAck message, client sends Abort message.

- The client sends Commit or Abort message to servers to complete the transaction or rollback. The client waits CommitAck or AbortAck message. If this is commit message, servers update the file with all update in the cache sequentially.

- On Abort(), the client sends abort message to servers. Waits for AbortAck messages.

- The client receives CommitAck or AbortAck message. The client returns error if file descriptor is -1 or receiving messages timeout.
- On Close(), the client sends close message to close the file. The client returns error if file descriptor is -1 or not sufficient server CloseAck messages are received in timeout time.

2.4 Protocol Timing

2.4.1 InitReplFs

The client sends Init messages to servers every 200ms, and client waits for 2s. The client only accepts servers InitAck message in init phase timeout.

2.4.2 OpenFile

The client sends OpenFile messages to servers every 200ms. OpenFile phase timeout is 4s. The client returns -1 if not sufficient server OpenFileAck messages are received in this 4s timeout time.

2.4.3 Vote

The client sends Vote messages to servers every 200ms, and Vote phase timeout is 4s. If any server has no response in 4s, the client treats this case as replies VoteAck message with -1 file description. The client sends abort message.

2.4.4 Commit and Abort

The client sends Commit or Abort messages to servers every 200ms, this Commit or Abort timeout is 4s. If any server has no response in 4s, the client returns error but servers keep the commit that has already committed. On the server side, if any server cannot receive any commit and abort message in commit or abort phase for 4 seconds, the server rollbacks file updates automatically.

2.4.5 Close

The client sends Close messages to servers every 200ms, and Close phase timeout is 2s. The client returns -1 if not sufficient server CloseAck messages are received.

3 Evaluation

3.1 Advantage

- This design is transactional. Files on all servers are consistence. Do not need to do commit on every write. If use conventional reliable transport, If any error happens on one of the commit, the entire file needs to rollback, this wastes a lot of network bandwidth.

- There is no acknowledge for each writeblock call. Before one commit, the client can do some other jobs. Commit and Abort timeout along with error check can guarantee that servers store replicated files. Servers cache all writeblock messages in memory can reduce block retransmission times and handle message receiving out of order problems.
- If any server crash during any operation, the client can detect this problem when the operation times out, and returns an error.

3.2 Disadvantage

- There is no writeblock acknowledgment before one commit. If the client did a lot of file write, and then do a commit with error, all file updates need to be reverted. This wastes a lot of time and network bandwidth. But traditional method can find the error earlier. The design should handle the tradeoff between these two designs.
- Servers are a state machine. If any client crashes in one operation. Servers stay in some middle state, and cannot go to the initialize state until the client recovers.

4 Future Directions

- In real distributed file system, servers supports more operations, such as read, search and delete. The design only supports create and update. The system can be extended with more features.
- When the system scales up, for one commit, the client needs to wait all servers acknowledge message to do the next operation. If any server crashes or does not available, the system will be very fragile and short time of timeout may not enough for a lot of servers. To fix the problem, the design can introduce server groups or quorum-style commit which means after receiving some portion of servers responses, the system can continue for next job.
- Storing all write blocks in memory is not a good idea for large-scale system with large amount of file data updates. The design should use partial updates log files to store some updates till some levels. When all updates are done, merge all updates log files into the final file.