Ryan Oman
Abhishek Choudhary
Brad Bicknell

Project 1 Write Up

Overview

Our implementation of RPC protocols performs simple file procedures and reliable, in order message delivery, even packets are delayed or dropped, as well as in the case of client or server crashes. The code we were given at the beginning of the project did not take into account node crashes and did not implement any file storage procedures.

Protocols

Our message layer uses different protocols to complete reliable file storage procedures. The messages included in our protocol are described below (implementation details are described later in this writeup):

ACK – ACK messages are sent from server to client. There is no payload.

DATA – Data messages are used to wrap RPCs (which are described below). The payload is a RIOPacket which contains the RPC that is to be executed by the server.

DATA_RTN - Data return messages are sent from server to client. These messages contain data that is supposed to be returned to the client from the server. The payload is the data. DATA_RTN messages are used to return file contents when a client does a get request.

ACK_SESSION – Ack session messages are sent from server to client. These messages are sent in response to and ESTB_SESSION message from a client. The payload is the clients session id (set by the server) followed by a space, followed by the last sequence number that the server received from this client.

EXPIRED_SESSION – Expired session messages are sent from server to client. These messages can be sent in response to any RPC from the client to the server. They payload is the client's new session id (set by the server).

ESTB_SESSION – Establish Session messages are sent from client to server. These messages are used when a client does not have a session id with the server it is going to send commands to. The payload is blank.

RPCs

All RPCs are sent in a RIOPacket, which is wrapped in a DATA message.

CREATE – Create messages are sent by a client node to a server node. The payload is the name of the file that the server is supposed to create.

GET - Get messages are sent from client to server. The payload is the file name. Get messages tell the server to return the contents of a file to the client.

Ryan Oman
Abhishek Choudhary
Brad Bicknell

PUT – Put messages are sent from client to server. The payload is the file name followed by a space, followed by the contents to write to the file.

APPEND – Append messages are sent from client to server. The payload is the file name followed by a space, followed by the contents to write to the file.

DELETE – Delete messages are sent from client to server. The payload is the file name. The server then attempts to delete the file with the given file name.

Implementation Description

Reliability
As stated in the overview, our implementation provides reliable message delivery using "at most once" semantics with node failures, To solve the problem of reliability when nodes crash, we implemented a way for clients and servers to keep track of their connections using sessions. In our implementation a client must have a valid session id when sending commands to a server. A valid session id means that both client and server agree on the client's session id. When a client node is asked to execute a command on a server for the first time, the client has to first establish a session with the server node before any command can be sent to the server.

The initial command is placed into a command queue while the client sends a ESTB_SESSION message to the destination server. Any additional commands that are given to the client while the session is being established are also put into the command queue on the OutChannel on the client side. When the server sees the ESTB_SESSION message, it assigns a new and unique session id to the client and stores this session id along with the client id with the InChannel associated with that client node. The server then responds to the client with an ACK_SESSION message containing the clients session id and the last sequence number it recieve from the client. In the case of new connection, the last sequence number received would be -1, signifying the server has not yet received a command from the client. In the case of the client previously had a session established with the server, but then crashed, this will let the client know that it needs to update it's last sent sequence number and session id for that connection to be what the server sends back to it. Upon receiving the ACK_SESSION, the client uses the session id given by the server in every subsequent command sent by the client to the server.

When the server receives a message from the client, the server checks to see if the supplied session id is recognized, if it is then the server acks. If the sequence number is also in order, then the server will execute the command. If the message is not in sequence, the server will put the message in an out-of-order packet cache that keeps track of packets that were delivered out of order. The server executes these out of order packets when it receives any packets that should have been received before the out of order packets. In another case, if the session id is not recognized, as in the case of server failure where all session ids are lost, the server sends back an EXPIRED_SESSION message to the client and does not execute the command. The EXPIRED_SESSION message contains a new session id that the server has assigned to the client with the expired session. When the client receives this message, it resets it's OutChannel with the new session id.

Our session implementation guarantees that a command is executed at most once. If a client sends an

append command to a server there are three possible outcomes, the server may crash before executing the command, crash after executing the command but before acking, or successfully complete the command and ack to the client. In all cases the command is executed at most once, even when the client re-sends the command after timing out. In the case of a server crash that completed the append the client will receive a EXPIRED_SESSION message after the command is re-sent and be forced to create a new session. This means that a command will happen at most once per session. When a client receives an EXPIRED_SESSION message, it also removes any unacked packets because they are no longer valid.

Storage Procedures

In order to implement the file storage system in the project specification, we created our own protocol to execute commands sent from a client to a server. Each possible file command uses a specific message described in the protocol. For example, to create a node on a server the client sends a CREATE message that is wrapped in a DATA packet which is then sent to the server. The client and server send commands through the reliable in order message layer, so all commands follow at most once semantics. When an error occurs, the server prints the error.

Assumptions

We make a few assumptions in our implementation. We assume that there is infinite storage for un-acked packets, out of order messages, and queued commands in the implementation of the in and out channels in the reliable message layer. We also assume that when the server fails, the client has to restart its session with the server, which causes the client to have to figure out what happened before the server crashed. We also assume that reliable means “at most once” and not “exactly once.”

How to Use

Our implementation does not require any extra scripts to run or compile. It is compiled and executed using the provided compile and execute scripts in the project framework.

Outstanding Issues

We were not able to find any major outstanding issues, but a few minor ones are worth mentioning. When appending or putting a new file, we do not handle escaping special characters in the file contents except for newlines. We also do not check for invalid characters in file names.