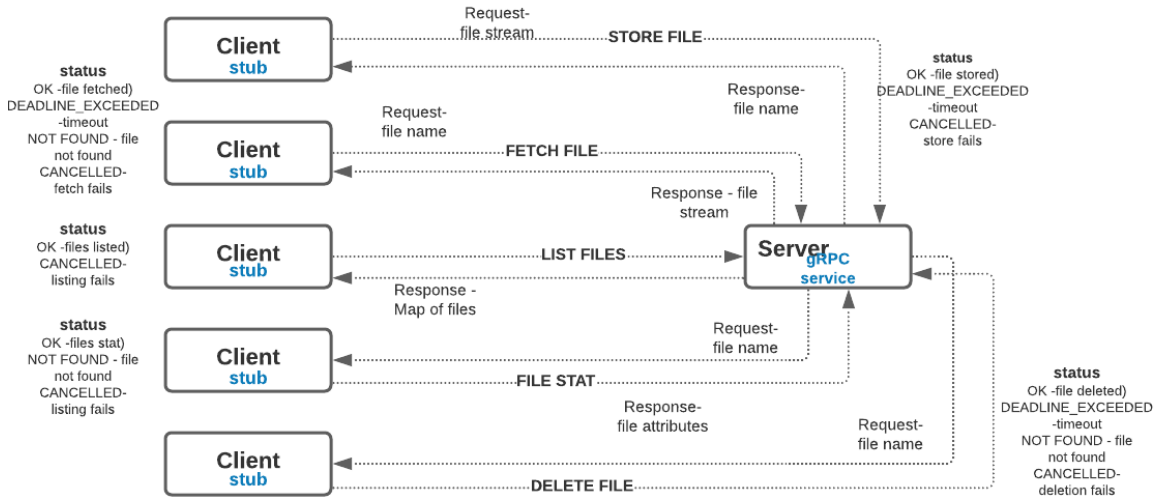
README

Introduction

This Readme file is a consolidated documentation of the two parts of project IV.

1. RPC Protocol Service.
2. Distributed File System.

Part I : RPC Protocol Service



In this part of the project, we are to create various RPC calls using the gRPC framework to store, fetch, list and get attributes of a file in a remote server. Using the protocol buffer mechanism, each RPC method is declared in a proto file with data, structured as message types. The gRPC framework generates the necessary services required by the client to establish connection ,communicate and pass data to the remote server.

The server implements the RPC service methods that are generated by gRPC framework while the client creates a stub that provides the same methods as the server. To avoid service latency and resource exhaustion ,during each RPC call the client context is set with deadline timeout.

For fetching of file from remote server, the RPC method in the proto file is defined with string request and streaming response message types. The file name is set to the request object and the fetchFile service method is invoked using the service stub by passing the client context and request. The ServerWriter API for server streaming RPCs writes the file data from the server to the response. The ClientReader API reads file data from the server response.If the requested file is unavailable, the server returns the status NOT_FOUND and for deadline timeout, status DEADLINE_EXCEEDED is returned. Anytime during file read operation, when the server encounters I/O exception, status INTERNAL is returned and finally if the file is successfully read and transferred, status OK is returned.

To store a file from the client to the server, the storeFile service method is invoked with the context and response objects which returns the ClientWriter. The ClientWriter API for client streaming RPC calls streams the file content from client to the server. The ServerReader API reads the file content from the request and saves it . If the context gets cancelled before the file gets completely stored in the server, status DEADLINE_EXCEEDED is returned.

To delete a file in the server, the file name is set to the request and passed to the service method via the stub. The gRPC service in the server checks for file existence (if not status NOT_FOUND is returned) and removes the file from the server . Status OK is returned if file deletion is successful, otherwise status CANCELLED is returned.

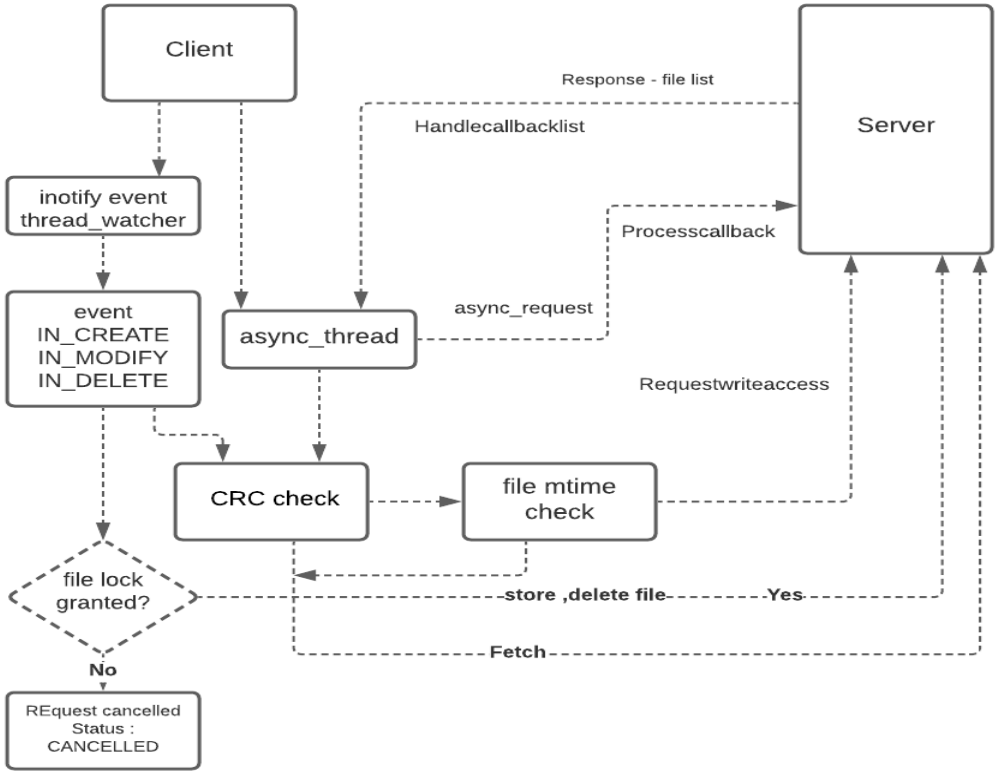
To list the files, present in the server RPC method listFiles is defined with request and response message type containing a repeated field called Items of message type item which can be interpreted as an array of structs with fields for file name and modified time. Upon receiving the request, the attributes of each file in the server path are added to the response. In the client , the server response is iterated to get each file attribute and are inserted into a map of files. Status DEADLINE_EXCEEDED or OK is returned for timeout and file listing success.

To get the attributes of a file, the gRPC service upon receiving the file name request, checks for file availability and performs a stat system call to get the file properties. The file modified time and creation time are set to the response. Status NOT_FOUND and DEADLINE_EXCEEDED is returned if file not found and session time out.

Tests done locally:

1. After launching the server, the client is run with store command with file name to be stored, file successfully stored in the server.
2. Server launched, client run with fetch command with file name, file requested successfully fetched from server.
3. Client run with command delete file_name , file deletion was successful in the server side.
4. Client run with list command , files in the server path were listed along with their modified time.
5. Server launched; client run with stat file_name command. The file modified time , creation time and file size were printed successfully in the console.
6. Server returned appropriate status for deadline timeout, file not found, I/O exception and successful file operation.

Part II : Distributed File System



In this part of the project , implemented a distributed file system where a client connects to a remote server via gRPC framework, sends asynchronous requests (async thread) to the server in a timely manner , and updates its local file system accordingly.

The client also runs a file watcher thread that monitors file changes that happen locally and update the server .

Each asynchronous request to the server returns a list of files and file attributes in the server. If the file is unavailable in the client mount path, the file is fetched from the server, else the modified time of the file in client path is checked and if its latest to the file in the server, the file is stored from client to server. Otherwise, the file is retrieved from the server to the client.

The file watcher thread uses the inotify utility to monitor creation, modification and deletion of files in the local file system. When a file is created , the IN_CREATE event is triggered which in turn makes the RPC call to store the newly created file in the server. When a file is modified, the IN_MODIFY is triggered and when a file is deleted the IN_DELETE event is triggered.

Before making the RPC for file store or file delete , the client requests for write access from the server which maintains a map of filename and client id. This allows for one creater / writer per file. When a client makes the RPC call for write access the server maps the file name to the client id and returns the status as OK if mapping is successful. If the file name is mapped to another client, status RESOURCE_EXHAUSTED is returned and if deadline timeout occurs status DEADLINE_EXCEEDED is returned. If the RPC is successful, the client makes RPC for storing or deletion of file after which the lock is revoked and assigned to a new client.

In order to prevent race conditions between the async thread and the file watcher thread, the inotify call back and the callback for handling asynchronous server responses, mutex lock and unlock is placed while invoking the inotify callback and iterating through the list of files from server. Similarly, the filename to client id map on the server is protected with mutex lock and unlock during inserting and erasing operation.

Tests done locally:

1. Server is launched; Client is mounted to the path mentioned, the async thread sends request to the server , and all the files from the server are downloaded.

2. A file is modified in the client, the inotify IN_MODIFY event triggered and if the file lock request is successful, the file is stored in the server.
3. A file is deleted in the client , the IN_DELETE event triggered and file is successfully deleted in the server.
4. A file is modified in the server, the client recognizes the change and downloads the file.

References

Part I

- Codeproject.com. 2020. *Protocol Buffer - A Beginner's Walkthrough*. [online] Available at: <https://www.codeproject.com/Articles/1260597/Protocol-Buffer-A-Beginners-Walkthrough> [Accessed 24 November 2020].
- 2020. [online] Available at: <https://www.youtube.com/watch?v=Y92WWaZJl24> [Accessed 24 November 2020].
- C++, U., 2020. *Upload File To A Server Using Protocol Buffer In C++*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/60997050/upload-file-to-a-server-using-protocol-buffer-in-c> [Accessed 24 November 2020].
- gRPC. 2020. *Grpc And Deadlines*. [online] Available at: <https://grpc.io/blog/deadlines/> [Accessed 24 November 2020].
- C++/C++11/C?, F., 2020. *Fastest Way To Check If A File Exist Using Standard C++/C++11/C?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/12774207/fastest-way-to-check-if-a-file-exist-using-standard-c-c11-c> [Accessed 24 November 2020].
- GitHub. 2020. *Avinassh/Grpc-Errors*. [online] Available at: <https://github.com/avinassh/grpc-errors/blob/master/cpp/server.cpp> [Accessed 24 November 2020].
- GitHub. 2020. *Claudioscordino/Grpc-Example*. [online] Available at: <https://github.com/claudioscordino/grpc-example/blob/master/client.cpp> [Accessed 24 November 2020].
- Cplusplus.com. 2020. *Input/Output With Files - C++ Tutorials*. [online] Available at: <http://www.cplusplus.com/doc/tutorial/files/> [Accessed 24 November 2020].
- ifstream, b. and B, A., 2020. *Ifstream, Bytes Read?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/1937408/ifstream-bytes-read> [Accessed 24 November 2020].
- GitHub. 2020. *Yitzikc/Grpc-File-Exchange*. [online] Available at: <https://github.com/yitzikc/grpc-file-exchange/blob/master/file_exchange_client.cc> [Accessed 24 November 2020].

- c++, H., Rathod, R. and Rathod, R., 2020. *How To Add Metadata To Streaming Grpc Calls In C++*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/38885422/how-to-add-metadata-to-streaming-grpc-calls-in-c> [Accessed 24 November 2020].
- Groups.google.com. 2020. *Google Groups*. [online] Available at: <https://groups.google.com/forum/#!topic/grpc-io/tNlNfYgWv7k> [Accessed 24 November 2020].
- O'Reilly Online Learning. 2020. *C++ Cookbook*. [online] Available at: <https://learning.oreilly.com/library/view/c-cookbook/0596007612/ch10s07.html> [Accessed 24 November 2020].
- GRPC, I., kumar, R., Anderson, E. and Mastrangelo, C., 2020. *Import Timestamp In Proto File Of Protobuf For GRPC*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/44468734/import-timestamp-in-proto-file-of-protobuf-for-grpc> [Accessed 24 November 2020].
- Clickhouse.tech. 2020. *Time_Util.H Source Code [Clickhouse/Contrib/Protobuf/Src/Google/Protobuf/Util/Time_Util.H] - Woboq Code Browser*. [online] Available at: <https://clickhouse.tech/codebrowser/html_report/ClickHouse/contrib/protobuf/src/google/protobuf/util/time_util.h.html> [Accessed 24 November 2020].
- rpc, H., 2020. *How To Return An Array In Protobuf Service Rpc*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/43167762/how-to-return-an-array-in-protobuf-service-rpc> [Accessed 24 November 2020].
- Groups.google.com. 2020. [online] Available at: <https://groups.google.com/g/protobuf/c/cJEWx1O_lfk> [Accessed 24 November 2020].
- GeeksforGeeks. 2020. *Map Insert() In C++ STL - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/map-insert-in-c-stl/> [Accessed 24 November 2020].
- C++?, H., litb, J., Bonini, T. and Kloberdanz, C., 2020. *How Do I Get A List Of Files In A Directory In C++?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/306533/how-do-i-get-a-list-of-files-in-a-directory-in-c> [Accessed 24 November 2020].
- C++?, H., 2020. *How Do You Add A Repeated Field Using Google's Protocol Buffer In C++?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/1770707/how-do-you-add-a-repeated-field-using-googles-protocol-buffer-in-c> [Accessed 24 November 2020].

Part 2:

- Example, I., 2020. *Inotify Example: Introduction To Inotify With A C Program Example*. [online] Thegeekstuff.com. Available at: <https://www.thegeekstuff.com/2010/04/inotify-c-program-example/> [Accessed 24 November 2020].

- gRPC. 2020. *Asynchronous-API Tutorial*. [online] Available at: <https://grpc.io/docs/languages/cpp/async/> [Accessed 24 November 2020].
- grpc c++ async server example, i., Chow, T., Chow, T. and Casey-Allen, H., 2020. *Grpc C++ Async Server Example, Is A Mutex Needed In The Processing State?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/58724528/grpc-c-async-server-example-is-a-mutex-needed-in-the-processing-state> [Accessed 24 November 2020].
- Groups.google.com. 2020. [online] Available at: <https://groups.google.com/g/grpc-io/c/-8dV3vF9RNY> [Accessed 24 November 2020].
- Reddit.com. 2020. [online] Available at: <https://www.reddit.com/r/cpp/comments/beh9dh/grpc_c_async_server/> [Accessed 24 November 2020].
- Cplusplus.com. 2020. *To_String - C++ Reference*. [online] Available at: <http://www.cplusplus.com/reference/string/to_string/> [Accessed 24 November 2020].
- value, s. and Seymour, M., 2020. *Std::Map Default Value*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/2333728/stdmap-default-value> [Accessed 24 November 2020].
- Cplusplus.com. 2020. *Stoul - C++ Reference*. [online] Available at: <http://www.cplusplus.com/reference/string/stoul/> [Accessed 24 November 2020].
- C++?, C., Tóth, Š. and Tóth, Š., 2020. *Client Grpc Connection With Reconnect In C++?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/61889726/client-grpc-connection-with-reconnect-in-c> [Accessed 24 November 2020].