Distributed Systems

Distributed Backup Service

Francisco Pinho - up201303744@fe.up.pt
Maria dos Santos de Abreu - up201306229@fe.up.pt

Enhancements

Chunk backup

In the default specification, the initiator peer sends a PUTCHUNK message (with retries in case the desired replication isn't achieved) and the other peers after a random delay store the chunk and only after storing the chunk do these peers start listening for STORED messages related to the chunk it just saved. This will result in basically every single peer storing the chunk and thus completely going over the desired replication degree and unnecessarily wasting storage space.

To solve this problem, peers now have volatile records that save information about every single STORED message received (regardless of whether the peer has stored the chunk or not) and before actually storing the chunk from a PUTCHUNK message, it checks in these records if that chunk has already achieved the desired replication degree and if that's the case he won't have to store it. This solution has been tested and in most cases the exact desired replication degree is achieved, occasionally going slightly over that number (for example if peers happen to be at the same stage of execution, if they happen to have very similar delays, improbable race conditions in a multithreading environment). The records are saved in a thread-safe data structure. In order to execute this enhanced protocol, the initiator-peer of the backup needs to be started up with command line argument for protocol: "2.0".

File Deletion

The issue with the proposed subprotocol lies with the fact that if a peer with a certain chunk of a file from another peer isn't running when the initiator peer sends a DELETE message for that file, which means that the chunk won't be deleted as intended.

Therefore, we decided to use the file system to our advantage and leave in the directory of dormant peers a file with commands to execute on start-up. Peers that are already running will delete this file through a frequently scheduled clean-up service and thus only peers that were not running when the DELETE message was sent will have access to these commands on the file.

Alternatively, we could have implemented some kind of ACK in order to determine which peers have executed the command but we decided that an enhancement without new messages was more desirable.

Concurrency

General Concurrency

A peer is an instance of the MulticastServer class and in each server there are three separate channels that run concurrently in their own thread, the Data Channel, Control Channel and Recovery Channel. Each channel continuously listens for data packages through a socket connected to their respective multicast group and when packets arrive an instance of a Runnable class "<ChannelName>PacketHandler>" is executed on one of the threads available in each channel's ThreadPoolExecutor, this means that each channel can concurrently listen for new and handle upcoming data packets without a thread being blocked in any of these actions. A protocol can be initiated by the client on any peer through RMI and depending on which protocol is initiated, the RMI remote object will call a function on one of the channels directly. The Data Channel handles backup and reclaim protocols, the Recovery Channel handles restore operations and the Control Channel handles delete operations, each protocol is executed within a Runnable Class instance on one of the threads available in each channel's ThreadPoolExecutor, thus allowing us to achieve true concurrency in all protocols. To support all of these operations thread-safe data structures such as ConcurrentHashMaps are used to store either temporary or permanent records (permanent records are regularly saved to file through a ScheduledThreadPoolExecutor instance within the main MulticastServer class) and these records are the way through which PacketHandlers and Runnables of

of protocols can access a single source of "truth" and thus have synchronized data across all threads.

Backup Protocol Concurrency

The Runnable inner class in the DataChannelListener "BackupService" will be initiated through RMI and delegated into its own thread through the Channel's ThreadPoolExecutor. Basically, we can have several instances of these runnables running on different threads and thus we can execute several backup protocols at the same time, the backup protocol is executed with or without enhancement depending on the inputted command line arguments of the server and each instance of "BackupService" has the necessary data for the backup of a single file and it will only run for as long as the backup operation lasts.

Delete Protocol Concurrency

The same principles as the backup protocol apply, there is a Runnable instance of "DeleteService" that executes its own delete operation within its own thread.

Restore Protocol Concurrency

The same principles as the backup protocol apply, there is a Runnable instance of "RestoreService" that executes its own restore operation within its own thread. RestoreService sends GETCHUNKs through a ScheduledThreadPoolExecutor thread spaced out with a 50ms delay as not to flood the peers with several instantaneous GETCHUNKs and the run method has a loop that checks if all chunks have arrived (every second there is a check for this as not to waste constant processing power) and if they have all arrived the chunks will be merged into the restored file and the RestoreService will end its execution.

Reclaim Protocol Concurrency

The same principles as the backup protocol apply, there is a Runnable instance of "ReclaimService" that executes its own reclaim operation within its own thread with a few differences. Chunks that have been deleted through the execution of this service cannot be stored again by the peer that executes this "ReclaimService" until 5 seconds have passed (by using temporary records that are cleared after that time). 5 seconds is enough time to prevent possible storings of upcoming PUTCHUNKs resulting from the sent REMOVED messages (of course this is only for the chunks that were removed through the reclaim operation, the peer can still store chunks from other unrelated backup operations, meaning that we do not allow protocols to conflict with each other).