Sistemas Distribuídos

Project 1 Report - Distributed Backup Service

Students of class 3 group 6:
up201505439 - João Mendes
up201502960 - Bruno Pinto

Concurrency design explanation:

Use of ConcurrentHashMap:

You used ConcurrentHashMap because we needed very high concurrency in your project to assimilate all the work happening in the system and the information that must be kept. They are thread safe without synchronizing the whole map, reads can happen very fast while write is done with a lock but here is no locking at the object level. The information was relative to the tracking of a file’s replication degree and to help in the restore protocol.

Use of ScheduledThreadPoolExecutor

Thread.sleep() to implement timeouts between processes can lead to a large number of co-existing threads, that take up resources and running time.

To avoid this, we used java.util.concurrent.ScheduledThreadPoolExecutor, which allows you to schedule a "timeout" handler, without using any thread before the timeout expires. One example of this is when a GETCHUNK message is sent, it is scheduled a thread a certain amount of time later to check if all chunks have been received through CHUNK messages, to assemble them and effectively restore the file.

Each peer has a Pool of threads to manage at will, responding to the messages that are received, making the overall protocols much faster. There are 3 main Threads, 1 per each MultiCast Channel (Backup, Control, Restore) that are basically listeners and wait indefinitely until the shutdown of the server.

In each channel, everytime a message is received, it is created another thread which processes it. There is a thread that manages this - Mailman - that basically dispatches task according to the content of the message.

Several protocol related classes are also Threads launched by Mailman, to achieve concurrency in this tasks.

We also took part of the “synchronized” in Java in some function to make sure our resources wouldn’t be corrupt.

Database

For a peer to survive it’s own crash we took advantage of Java’s Serializable, using a shutdown hook, we save the information we need and when our peer is brought back up we read the information from the serialized file. In this cases, we allow the RMI registry to rebind instead of just binding so that we can use the same name for the access point as before the crash.

Enhancements

Backup

To survive the depletion of backup space rather rapidly or too much activity on the nodes once that space is full, we decided that each peer should keep track of how many STORED messages it has received for every different chunk. This allows us to, before saving a chunk upon receiving a PUTCHUNK, check if the desired replication degree has already been achieved. If it hasn’t we proceed to store the file, otherwise we ignore the message. The required random delay gives us time to do this operation.