Project 1 - Distributed Backup Service

T2G01

Francisco Tomé Macedo Martins Santos Moreira – up201607929

Antero Campos Gandra – up201607926

This report sets to describe the enhancements and concurrency implementation for the Distributed Backup Service proposed.

Concurrency Implementation

Our Concurrency implementation seeks to be efficient and robust by not resorting to any inactive waiting periods within or between protocols. To achieve this we divide individual tasks into threads as much as possible and focus on making sure all dependant tasks don't have to wait.

This can be seen in many regions of our implementation. To begin with, each Peer has a Socket Thread for each of their Multicast Channels. This allows the Peer to process messages coming from multiple channels at the same time. However if each of these threads processed the messages themselves that would stall the system of other messages coming from those channels. Therefore when a Socket thread receives a Datagram Packet it launches an Interpreter thread to handle processing the message and so the Socket thread can be ready to receive other messages from that Channel.

This allows the system to be much faster and responsive as well as more reliable. However, using multiple Interpreter threads means that much of the Peer information is going to be accessed and altered by many threads at the same time. To use this safely we make use of the many tools java provides to handle concurrency. Specifically we make use of ConcurrentHashMap as well as AtomicBoolean.

Enhancements

Due to time constraints we weren't able to implement any of the proposed enhancements even though we took notes and planned our approach.

- Backup

For the backup enhancement we considered an approach similar to that of the base version of the Restore Protocol, that is, have each of the peers that receive a PUTCHUNK message wait a random delay uniformly distributed between 0 and 400 ms. If during this period they receive a STORED message for the same chunk and the replication level of the chunk is satisfied then the chunk no longer needs to be saved by the peer and space can be saved.

- Restore

For the restore enhancement we considered using a direct TCP connection from the peers with chunks to the peer asking for them. This would require setting up new messages to exchange the IP and port.

- Delete

For the delete enhancement we considered keeping a registry of all the files a peer has asked to be deleted. Additionally, when a peer joins the network it would send a message signalling the other peers all the chunks it currently holds. With this, when a peer that deleted a file finds that a peer still has chunks for that file it would resend a message to delete that same file. This would guarantee peers don't waste space with chunks for files that are no longer used.