Backup Enhancement

Specification

This scheme can deplete the backup space rather rapidly, and cause too much activity on the nodes once that space is full. Can you think of an alternative scheme that ensures the desired replication degree, avoids these problems, and, nevertheless, can interoperate with peers that execute the chunk backup protocol described above?

Solution

Our solution to this first enhancement was simple. Since the replication degree can be bigger than the desired, we implemented a method that makes sure the desired replication degree can't be bigger that what the user specified.

After waiting a random interval between 0 and 400 ms, we check if the peer already has the chunk information sent by other peers. If it does, we check if his current replication degree is bigger or equal to the desired. If it is, we don't allow the peer to store that chunk, hence the current replication degree stays equal to the desired. If not, we proceed normally and the peer stores it.

Restore Enhancement

Specification

If chunks are large, this protocol may not be desirable: only one peer needs to receive the chunk, but we are using a multicast channel for sending the chunk. Can you think of a change to the protocol that would eliminate this problem, and yet interoperate with non-initiator peers that implement the protocol described in this section?

Solution

It was clear we needed to get rid of the multicast information in this enhancement, so we established a UDP connection to transmit the information using a Datagram Socket.

The initiator peer binds port 4572 and creates the following header:

GETCHUNK <Version> <SenderId> <FileId> <ChunkNo> <IPV4:PORT> <CRLF><CRLF>

Using this new header message we are able to inform the peers about the IP and the PORT to use if they have the requested file.

Next we create a thread to be responsible for the messages reception. This thread analyses the received messages and stores them accordingly in the initiator peer, ignoring duplicates.

When the other peers receive the new message, they proceed with their default behavior, except when it's time to send the message. If they're using the enhanced protocol, they establish a connection using the IPV4 and the PORT information in the header message, and send the chunk file data using that connection. They also send a default **CHUNK** multicast message (only with the header, no need for the body) to inform the others peers there's no need for them to send that chunk, since it has been handled.

Once we get all the chunks required for the requested file, we join them to create that same file, where we go back to the default behavior.

File Deletion Enhancement

Specification

If a peer that backs up some chunks of the file is not running at the time the initiator peer sends a DELETE message for that file, the space used by these chunks will never be reclaimed. Can you think of a change to the protocol, possibly including additional messages, that would allow to reclaim storage space even in that event?

Solution

For this enhancement we decided to write and load a file which we called **peers_to_delete.txt**

Here's the general structure:

<FileId>..<PeerId_1>..<PeerId_2>..<PeerId_N>

Basically, when a peer receives the DELETE message it sends a confirmation message, allowing the initiator peer to keep track of the peers that have effectively removed the chunks of the file.

We creates a new HashMap variable, with the key as the file id and the value as an ArrayList<Integer> with the peers that still need to send a delete confirmation message.

Before sending the DELETE message, the initiator peer puts in the HashMap variable the file id and all the peers that have chunks of that file. Every time a confirmation message is received, the initiator peer removes the peer that has just sent the confirmation on the HashMap variable, and writes to the file mentioned above the content of the variable.

The confirmation message structure is as follows:

ENH_DELETED <SenderId> <FileId> <CRLF><CRLF>

Now when a peer is down or crashes, the initiator will not receive the confirmation message of that peer. When the peers reboots and becomes active again, it sends a multicast message saying he just awoke, and the peers that have in the HashMap variable files that weren't deleted on the peer with that id, will send the DELETE message to that peer, allowing the peer to remove the chunks of those files.

The awoke message is structured as follows:

ENH_AWOKE <Version> <SenderId> <CRLF><CRLF>