Salvador: A Decentralized, Secure Backup System

Daniel Almeida[1] and Rodrigo Araujo[1]

[1]*Department of Computer Science, University of British Columbia*
{*daa, rodarauj*}*@cs.ubc.ca*

ABSTRACT

In this paper we present Salvador, a decentralized and backup system. Salvador relies on a Peer-to-Peer (P2P) architecture to enable users to store and access their files in a secure manner without sharing entire copies of files. We compare the overall network traffic and time taken to complete operations between a hybrid P2P design (Version 1) and a fully decentralized design (Version 2) while experimenting with different files and file block sizes.

I. INTRODUCTION

Many modern systems that offer file backup and sharing rely on distributed storage in order to achieve load balancing, fault-tolerance, and scalability. Systems like Dropbox provide a form of interaction that resembles a client-server from the user's perspective while leveraging a Cloud-based architecture and that is highly distributed. Dropbox is a file hosting service that enables users to store and share personal data with semantics similar to those of a local file system while storing the data across many geographically distributed servers. Other systems adopt a P2P architecture, such as pStore [1] and the Cooperative File System (CFS) [2]. pStore is a secure, distributed backup system that leverages unused disk space that offers features such as encryption and sharing and CFS is a read-only storage system.

In this paper we present Salvador, a system that draws inspiration from many of the systems mentioned. Salvador does not offer file sharing capabilities, which means that the only user with access to the entire file and allowed to backup or restore that file is its owner and creator of that file. At the same time, the semantics of *Salvador* are similar to that of Dropbox in the sense that users can easily create, edit, or delete files on their computer while being able to recover such files by collecting its parts from peers and reconstructing the file locally.

II. SOLUTION

In this project, we implemented a decentralized, P2P file backup system. We present the following scenario to define the terminology we use and give an example of how *Salvador* would be used.

A student at the University of British Columbia has just finished taking notes during a Distributed Systems lecture. He used a simple text file on his computer, but because those are precious notes that will help him prepare for the final exam, he decides to use *Salvador* to backup his notes. All he has to do is to use a command-line interface to issue the following command: *salvador backup /documents/DistSysNotes.txt*. The file DistSysNotes.txt is split into blocks that are encrypted and sent to peers to be stored. In this scenario, we refer to the student's computer as the owner. Given a file A, an owner is the only peer that has the entire file A and is able to backup or restore the latest version of A (which we call Backup and Restore operations, respectively). The computers storing file A's blocks are called peers.

A. Assumptions

Firstly, we assume that a peer is never compromised, i.e., only the owner of a file has the private key used to encrypt a file (or its blocks). Therefore, no other user in the system can successfully decrypt that file.

Secondly, we assume that peers execute the same code and the communication between peers is reliable, i.e., all messages are delivered in order and are not corrupted, but might be delayed.

Thirdly, we assume that a joining node knows at least one node already in the network. This node is used by the joining node to obtain a list of known nodes in the network.

B. Development process

We have split the development of Salvador into two phases:

1) *Version 1*. Initially, we adopted a hybrid P2P architecture in order to lay the foundation of Salvador and worked on its semantics and usability;
2) *Version 2*. Then, we adopted a full P2P architecture by removing the central server introduced in the first version. As the basic functionality was already achieved at this point, it was easier to safely change the internal architecture to support the removal of the central server and move its functionality to the peers.

C. Version 1

The very first version of *Salvador* is based on a hybrid P2P implementation and assumes no failures of either the server or the peers (owner and hosts). The Server is responsible for coordinating peers during Backup or Restore transactions. The responsibilities of the Server include determining which hosts an owner should send his blocks to and providing their locations. The server also keeps track of the status of an Operation and which blocks have been successfully stored by hosts, informing the owner once a Backup or Restore Operation has been completed. We started with a naive implementation as the foundation that represents how

Salvador should work in an ideal scenario and on which we can iterate towards a fully decentralized solution. The steps involved in a Backup Operation are as follows (Figure 1):

1) Owner requests metadata of File A, starting a new Operation. The Server responds with a list of available peers assigned to hold each of File A's blocks;
2) Owner encrypt the File A and splits it into blocks;
3) Owner sends blocks to peers according to metadata received from the Server;
4) Peers acknowledge (ACK) the blocks they received by sending a message to the server, which will trigger an update to the metadata of File A;
5) Server notifies owner that operation was successful and all of File A's blocks have been successfully stored.

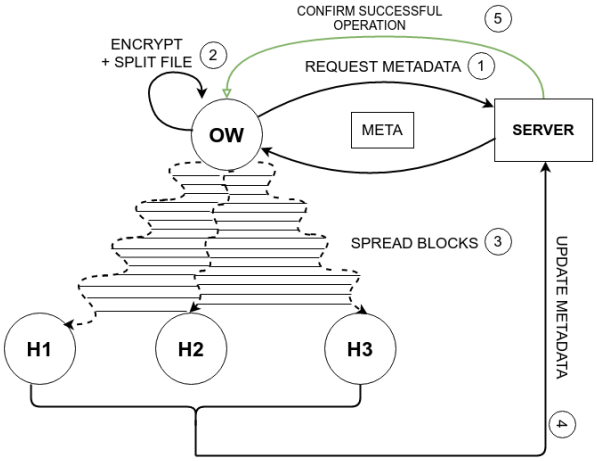

Fig. 1. Spread procedure on Version 1

The steps involved in a Restore Operation follow a similar approach to the Backup Operation:

1) Owner requests metadata about the blocks of File A. Server responds with a list of peers storing File A's blocks;
2) Owner sends request to retrieve blocks to each peer in the previous list;
3) Peers send the requested blocks to the owner and expect an ACK in return;
4) When the owner receives a block, it sends an ACK to the sender peer and to the server;
5) When the owner receives all File A's blocks, it reconstructs the File A and finishes the Restore Operation.

D. Version 2

The main difference in Version 2 is the lack of the central server, meaning that the responsibilities once performed by the Server have now been moved the Peers. The Peers themselves are responsible for coordinating and maintaining the metadata about their files and blocks.

Although the overall architecture has changed, the core features seem to be exactly the same to the user. Figure 2 shows how a new peer joins the network: a new peer (N1), when joining the network, sends a PING message to a known peer in the network (in this case N2). In return, N2 replies with a PONG message that contains a list of all peers it already knows.

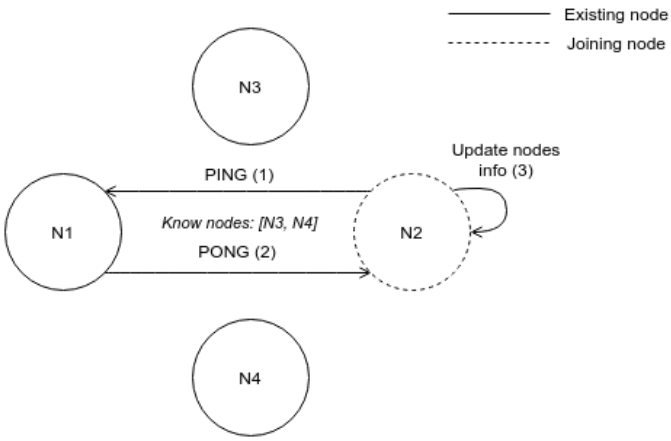

Fig. 2. New node joins the network - Version 2

In this version, the Backup Operation is more straightforward; the owner node sends a Backup Request to all known nodes, the ones that respond faster will be chosen to receive the blocks. The steps involved in a typical scenario of a Backup Request Operation are as follows (Figure 3):

1) Owner node (N1) sends a request to all known peers, in this scenario: N2, N3 and N4. It will request the result of (number of blocks to be backed up / number of nodes in the network) blocks from each peer;
2) N2 and N3 respond with an "Ok" message for all requested blocks, however, N4 can store only 2 out of the 4 requested blocks;
3) After a timeout, N1 starts the backup and realizes it is missing 2 blocks space;
4) N1 then sends another round of requests for the remaining space necessary to fully backup all blocks;
5) N2 responds quicker;
6) N1 starts to backup the 2 remaining blocks. Messages from N3 and N4 will be ignored;
7) After a timeout, N3 and N4 will free the blocks reserved for N1.

A mechanism for reserving blocks space and eventually releasing them is necessary. When a peer receives a Backup Request, it must either accept it or reject it. However, due to the concurrent nature of the system, a second Backup Request may arrive after the first request but before the related blocks have been transferred. Without a mechanism to reserve blocks for a certain request, the same storage space might be promised to two or more peers. *Salvador* uses a timeout to limit for how long the requested blocks are kept reserved for a certain peer. If the expected operation doesn't start soon enough, the blocks are released and made available

to other peers.

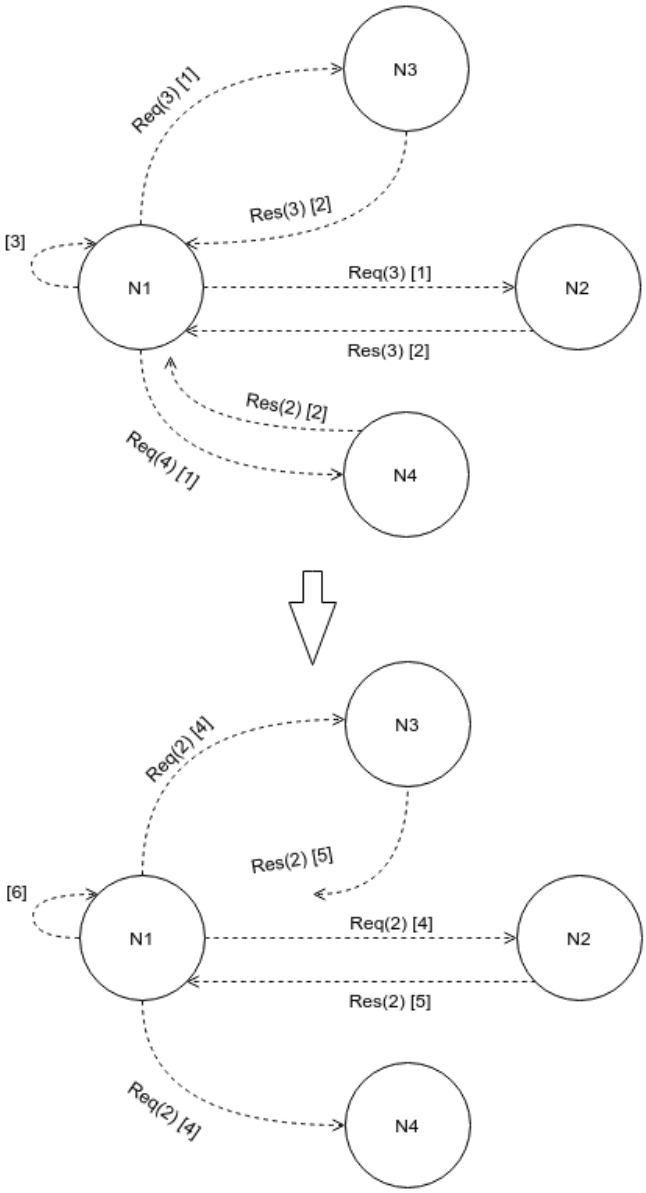

Fig. 3. Backup Operation - Version 2

III. EVALUATION

We had two main goals in this evaluation:

1) Evaluate the network load between owner and peers;
2) Evaluate the performance of both backup and restore operations in order to understand how it changes as the variables change (e.g., block size, file size).

We have experimented with different number of peers in the network, different block sizes and file sizes.

For the first version of Salvador, Figure 4 shows the correlation between the time to backup a file and the size of the block being stored among the peers. It is easy to see that the bigger the block size, the faster the backup (and restore) is. However, this trade-off is more complex than it seems; bigger block size means less granularity and we will have to depend on the availability of a node that might be storing big chunks of our file. Even though it will be slower to backup/restore, with more granularity we can spread more blocks, which we assume is safer in the long run. Yet, to make this evaluation precisely, we need to test this prototype in real world and large scale scenarios.

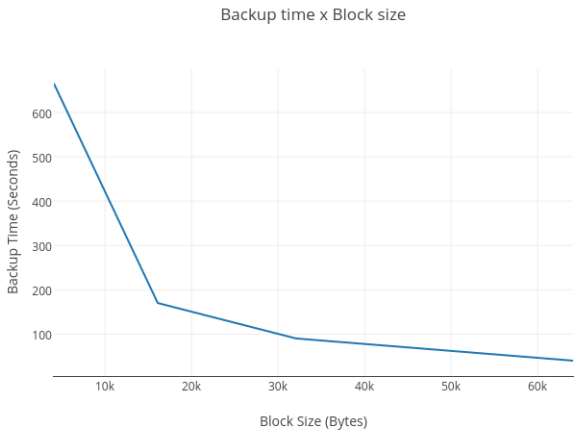

Fig. 4. Version 1. Correlation between backup time and block size

Similarly, Figure 5 shows the correlation between network load and block size in the first version. As the block size gets bigger, the network load to exchange messages gets smaller but we fall in the same granularity trade-off. Our initial expectation was to see the network load decrease when we moved from Version 1 to Version 2 (full P2P).

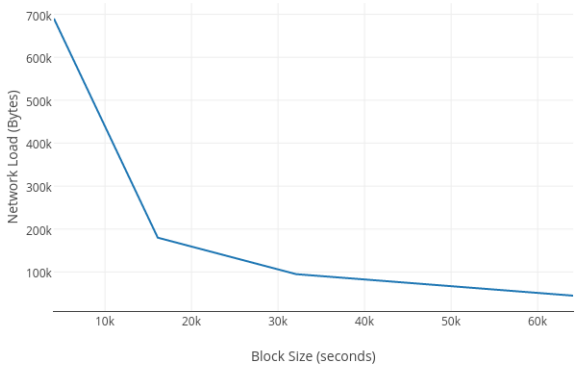

Fig. 5. Version 1. Correlation between network load and block size

In this evaluation of the full P2P version, we experimented with 2 different file sizes. Figure 6 shows a decrease in the backup time, whereas Figure 7 and Figure 8 display a considerable decrease in the network load.

These experiments show that moving from a centralized client-server architecture to P2P architecture not only improved the speed to fully backup and restore a file, but also

reduced the amount of messages being exchanged among the nodes, which resulted in decreased network load in a node.

However, this experiment was executed in a controlled and local environment and the results of this evaluation could change significantly in a real world scenario with real and geographically distributed users. That being said, we believe this evaluation provides good and useful insights.

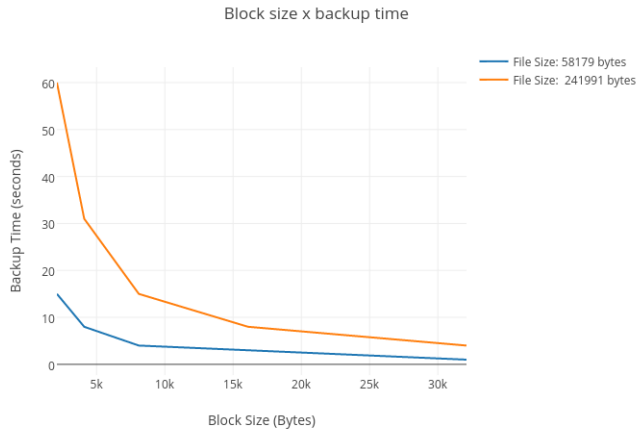

Fig. 6. Version 2. Correlation between backup time and block size

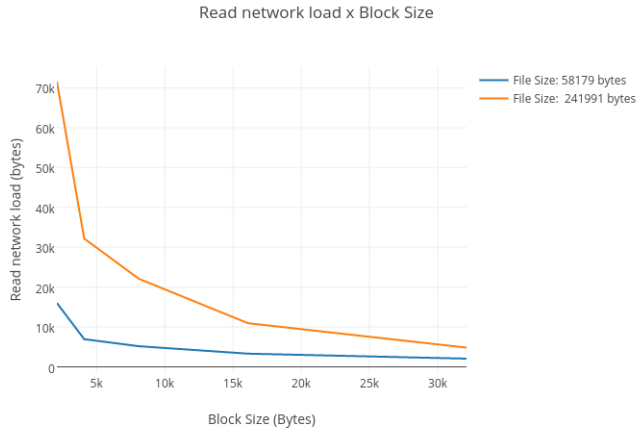

Fig. 7. Version 2. Correlation between read network load and block size

IV. CONCLUSION AND FUTURE WORK

In this work we implemented Salvador, a P2P file storage system that enables the user to backup and restore files without sharing the entire copy of a file with a central server. We implemented this by encrypting and breaking a file into block and then spreading them among peers in this network. When a user wants to recover a backed up file, the system will request the blocks from each peer and then reconstruct the file.

As a fully decentralized solution, *Salvador* has to be able to maintain the metadata of files consistent in the presence of failures of peers, which was challenging to implement. Other

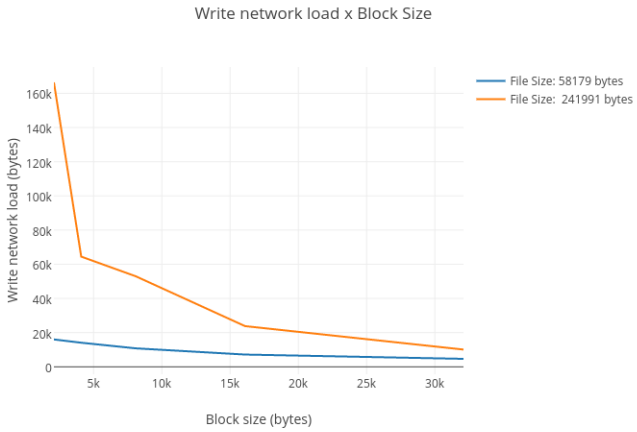

Fig. 8. Version 2. Correlation between write network load and block size

important issues are liveness and availability since peers can arbitrarily disconnect. The size of file blocks is extremely important to make our system useful in practice, affecting not only the robustness and performance of our system, but also the availability of files. The trade-off between many variables can strongly affect the system; thus, an initial evaluation of the prototype was made, which gave us insight about how the system behavior changes as the variables change. In the end a decision has to be made between increasing granularity for availability by having more spread blocks and decreasing granularity to gain more speed of backup and restore. Yet, the latter depends too much on a single peer's availability to recover the file.

Following this discussion, the question on how to motivate users to keep peers' data remains open. Finding an answer to this question would be one of the next steps of this work, since it is very important to make sure that there is a symbiotic relationship between users.

At this point we can concurrently handle peers joining the network at any time; however, another future work is to gracefully handle peers leaving the network. In other words, handling a peer failure or a peer logging out of the network temporarily. The current version does not detect the disconnection, but a plan to attack this problem is in progress.

Another future work is redundancy of blocks: It is useful to create a mechanism to redistribute some blocks to different peers. This process would make the backup more available by storing more than one copy of a block in different peers. Thus, if a peer is not available at the time, another peer could provide the requested block.

We conclude that a distributed file storage system is not only safer (by not giving an entire copy to a single company) but also is high in performance, given the many challenges and trade-off decisions.

REFERENCES

[1] Batten C, Barr K, Saraf A, Trepetin S. pStore: A secure peer-to-peer backup system. Unpublished report, MIT Laboratory for Computer Science. 2001 Dec:130-9.

[2] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, Wide-area cooperative storage with CFS. MIT Laboratory for Computer Science.
[3] Dropbox website: https://dropbox.com
[4] Michael Piatek, Arvind Krishnamurthy, Arun Venkataramani, Richard Yang David Zhang, Alexander Jaffe. Contracts: Practical Contribution Incentives for P2P Live Streaming