Cooperative Backup System

Sameh Elnikety Rice University
Mark Lillibridge Compaq SRC
Mike Burrows Microsoft Research
Willy Zwaenepoel Rice University

Abstract

This paper presents the design of a novel backup system built on top of a peer-to-peer architecture with minimal supporting infrastructure. The system can be deployed for both large-scale and small-scale peer-to-peer overlay networks. It allows computers connected to the Internet to back up their data cooperatively. Each computer has a set of partner computers and stores its backup data distributively among those partners. In return,

such a way as to achieve both fault-tolerance and high reliability. This form of cooperation poses several interesting technical challenges because these computers have independent failure modes, do not trust each other, and are subject to third party attacks.

1 Introduction

Traditional data backup techniques rely on storing backup data on highly reliable storage devices off-site. These techniques are reliable and widely used. However, they are expensive because of the cost of the drives and the media, as well as the cost of managing the media and transferring it off-site. Internet backup sites offer another alternative. Typically, they offer the backup service for a monthly fee. Customers upload their data to the backup site and their data is made available if needed later on during their subscription period. Internet backup sites charge a high fee because of the dedicated resources necessary to store the backup data.

This paper proposes a cooperative backup system that uses slack resources at participating computers to back up their data. Hence, this technique is cost effective because it is a form of cooperation in which each computer receives as much service as it offers.

The system requires the participating computers be running and normally connected to the Internet to allow backup and restore operations to occur at any time. These computers form a peer-to-peer overlay network. The system uses minimal supporting infrastructure. It does not require the nodes to have unique identities, certified public keys, or decentralized message routing algorithms. It needs only to do node discovery, which can be implemented as peer-to-peer service or through out-of-band means.

Each computer has a set of partners from different geographic locations. This geographic diversity is important because it guarantees independent failure modes for the computers; it is analogous to taking the backup tapes off-site for traditional backup systems. The backup relationship is reciprocal. However, the partners need not form a clique such that the backup relationship is transitive.

As explained later, there may be bad partners. For example, a bad partner is always down or disconnected

rs

is bad, it will cancel the backup agreement with this partner and will find a new partner instead.

2 Design Issues

As the backup data of each computer is stored distributively among its partners whom the computer does not trust, the system must ensure the confidentiality of the backup data. Moreover, the system must be robust

enough to enable a computer to restore or back up its data even if some of its partners are down. When a computer restores its data, it needs to make sure that it is restoring the latest copy that it had backed up, and that the copy has not been tampered with. The next subsections explain these design issues and the techniques that we use to resolve them. We have devoted the next two sections to discuss security aspects of the system.

2.1 Confidentiality

To ensure the confidentiality of the backup data, which is stored remotely among the backup partners, each computer encrypts its data before sending it to the partners. We use secret key cryptography; each computer has its own secret key (and does not share this key with any other computer), and encrypts the backup data using block ciphers. In particular, we use IDEA to encrypt the backup data. The secret key (along with some other small pieces of information) has to be stored separately off-site.

2.2 Robustness

For a computer to do backup or restore, it needs to communicate with its partners. However, a few partners may be down at any instant of time. The system uses erasure codes; if some parts of the data are missing, it is possible to reconstruct the whole data. The main idea of erasure codes is to add redundancy to the data by partitioning the data into blocks and augmenting the blocks with redundant information so that error recovery is possible if some parts are lost. We select Reed Solomon erasure codes rather than Tornado codes because they require less network traffic and optimally minimal storage. Although Reed Solomon erasure codes require more processing time than Tornado codes, the time used to encode the data is typically less than the time used to do the encryption and to compute the cryptographic checksums.

Each data block consists of several sub-blocks. After applying erasure codes, we get an augmented block that consists of the same original sub-blocks plus extra redundancy sub-blocks. Each sub-block of the augmented block is stored at a different partner. So, the failure model is that of an *erasure*; a whole sub-block is either available or missing. We use other means to detect errors inside the sub-blocks. In particular, as explained later, these sub-blocks are self-checking. For each individual computer, configuring the number of original sub-blocks and redundancy sub-blocks can achieve very high reliability at the expense of more space overhead. For example, adding 6 redundancy sub-blocks to each 6 original sub-blocks gives %99.995 reliability at the expense of 100% extra space overhead.

2.3 Integrity

When a computer restores its data, it needs to ensure that the data that it is restoring has not been modified. Thus to guarantee the integrity of the data, each computer signs its data by using a keyed hash function such as HMAC-MD5, which is a one-way cryptographic function of both data and hash-key to a cryptographic hash value. The hash-key must be known to generate or to verify the hash value. During backup operation, the computer attaches a cryptographic hash value to each sub-block. Therefore, the sub-blocks become self-checking and any modification to them can be detected.

2.4 Authentication

Authentication is necessary to prevent imposters. However, we need neither *certification* nor a public key infrastructure. We use the Diffie-Hellman method to establish a shared secret key between each two partners.

Partners authenticate all messages by adding a sequence number to each message and a cryptographic hash value of both the message and the sequence number. Using Diffie-Hellman introduces a variant of the man-in-the-middle attack, which we discuss in the next sections.

3 Security Issues

The security issues are a major concern in such a peer-to-peer environment because partners do not trust each other and the system is vulnerable to third party attacks. We believe that some of these issues are novel and have not been encountered in other systems, and possibly there are potential security holes that we may have not discovered. We classify security issues into cheating attacks and spite attacks. We discuss cheating attacks in this section and spite attacks in the next section.

For cheating attacks, an attacker tries to gain a direct benefit (e.g., extra storage space or free backup service) as a result of misbehaving. In this section, we assume that computer X is malicious whereas computers Y and Z are good. The next subsections describe the following: how a computer ensures that its partners are holding its data, how to prevent having backup for free even if partners are allowed to be down for long periods of time, and how to prevent storage amplification.

3.1 Ensuring that partners are holding backup data

Computer Y issues periodic challenges to its partners to ensure that they are holding its backup data. A challenge is a request to read a randomly chosen sub-block. When computer Y receives the requested sub-block, it checks the sub-block number, date stamp and integrity.

A challenge remains outstanding till the sender receives the expected reply. If a challenge remains outstanding for a long time, the sender denies all random read and write requests from that partner. However, a request for *restoring* the data is still granted as described in the next subsection. Later on, if the challenge remains outstanding for too long, the sender decides to replace this partner and finds a new partner.

3.2 Having backup for free

The system allows some tolerance for participating computers. For example, a computer can accidentally be down for a week or two without losing its partners. This period of time is called the *grace period*. However, this introduces the possibility of having the backup service for free as in the following scenario: computer X joins the system, acquires a set of partners, and backs up its data. Then, it pretends to be down for the grace period. And just before the grace period expires, it rejoins the system, gets a new set of partners and repeats the same steps. During the grace period, computer X had *free* backup service as it could restore its data at any time and it neither stored the data of its partners nor responded to any challenge or request.

To prevent this possibility, we introduce a probation phase that we call the *commitment period*, which is longer than the grace period (e.g., double the grace period). The commitment period applies only between any two new partners. During the commitment period, the two partners are required to hold each ot
without allowing any restore operation. In other words, backups are allowed as well as challenges and random read and write requests, but restore operations are disallowed. It worth mentioning that random reads are different from restore requests because for the former, it is possible to exchange challenges concurrently with the read requests, whereas for the latter, challenges cannot be exchanged as the computer which issues the restore request would have lost its partners data along with its own data.

To show that this method works, select the grace period to be two weeks and the commitment period to be four weeks. If computer X cheats to have free backup service, it joins the system and holds its partners data for the commitment period. Then, it uses the grace period of two weeks. In this way, computer X is forced to hold its partners data for four weeks without any backup privilege in order to have two weeks of free backup which is a definite loss for computer X. Paying the commitment cost does not prevent a computer from having the backup service when it replaces one of its partners because it can restore its data from the remaining partners.

3.3 Storage amplification

A malicious computer X can abuse the system to get access to as much storage space as possible. Let us consider the following scenario where the commitment period is double the grace period. Computer X gets a new partner (Partner1). Then, after the grace period, computer X gets another new partner (Partner2), and so on. During the third grace period, computer A has a read/write access to its data on computers Partner1, Partner2 and Partner3 and has to keep the data for only computers Partner2 and Partner3. This means that computer X can access 150% of its storage space; so the duty cycle is 3/2. Computer X can pipeline this process to increase its storage space exponentially. A simple solution to thwart this attack is to prevent random read and write access if a computer is not responding to pending challenges.

4 Spite Attacks

Spite attacks are less important than cheating attacks because in spite attacks attackers try to prevent the backup service for parts of the system without gaining any direct benefit.

4.1 Man-in-the-middle attack

In this attack, a malicious computer X declares that it has infinity storage space and that it is willing to partner up with any other computer. Whenever it manages to have two new partners: computers Y and Z, it it does not store the backup data of its partners locally, but it keeps only the mapping information and swaps data between each two partners. Similarly, it forwards requests and challenges from computer Y to computer Z and vice versa. Computer X can potentially have a very large number of partners. Moreover, if there are several computers behaving as computer X, almost all other computers in the system will have some of these malicious computers as partners. If all these malicious computers come to a sudden stop at the same time, many computers will find that a large ratio of their partners are down and will not be able to do backup or restore operations. We developed a protocol that commits the two partners to a random sub-block number and then the partners exchange special challenges, which guarantee that the data is stored locally.

5 Conclusions

We presented the design of a backup service as a peer-to-peer application. This method is cost-effective because it uses slack resource at participating nodes. The system scales up and down as it can be deployed among a few nodes, contrary to most peer-to-peer applications. We developed several protocols that prevent a bad computer from exploiting the collective system storage. We discussed several novel security attacks. We are considering a wide scale deployment of this system. Also, we plan to investigate other types of peer-to-peer applications what require minimal supporting infrastructure.