BLT Storage: Cloud-Based, Reliable Storage Service

Antony Toron, Princeton Class of 2019
Advised by Brian W. Kernighan

Motivation and Goal:

As storing data and finding affordable ways to do so becomes more prevalent among every day consumers, it seems natural to provide a cloud-based approach (as companies like Google and Dropbox have done). With the rise in importance and prevalence of storage systems, this project is aimed at developing a storage solution for public use that offers a variety of different services, built on top of underlying cloud-based storage. Companies like Amazon offer easy ways to utilize remote storage by providing solutions like Amazon S3 and EFS to developers, which allows for a clear middle layer to insert a user-facing application that uses these systems for storage.

Advances in these cloud-based technologies and services offered by major corporations have made remote storage systems much more efficient and feasible. As the volume of data and importance of it increases, many if not most storage systems have turned to both hardware and software solutions to adding reliability guarantees for the data. A well-known approach for providing reliability at the hardware level is RAID, or Redundant Array of Independent Disks.

The goal of my project is to understand and implement these methods of creating reliable storage through the use of cloud storage such as that mentioned above, to ultimately create a flexible storage service that can rely on a multitude of underlying storage providers while providing an easy interface for users. Using cloud services for storage, combined with the design of the application resting in-between users and the actual storage, allows for a cost-effective way to store data as opposed to owning physical devices. It also allows for a simple way to add these extra layers of reliability on top of the base reliability provided by the raw storage.

Problem Background and Related Work:

There exist many different applications that provide remote storage for users, like Google Drive and Dropbox, all of which are more than likely to use approaches like RAID to provide their reliable services. The concept behind RAID is to insert redundancies into the system and distribute the data to ensure that failures in one or possibly more drives may still cause no negative effects, because of abilities to restore the corrupted storage. RAID, which been around for quite a while now, has become ubiquitous in large storage centers (particularly in the hardware) because of its ability to recover lost data and also decrease latencies in write/read times on hard drives in some of its implementations.

What hasn't been explored as much by other applications in the industry is building various hardware-inspired algorithms on top of cloud-based storage, which is interacted with inherently via software. In particular, Amazon's EBS (Elastic Block Store) provides (reliable) storage which expects above 1 to 2 failures per 1,000 EBS volumes/drives per year, which is about 20 times more reliable than commodity disk drives[1]. This seems like a great opportunity to add extra layers of software-run RAID solutions to provide extra guarantees. By exploring this middle ground between the actual storage on the cloud and the final client-facing product, my project can implement interesting software on top of the base storage, but also make it very easy to abstract away the storage from the algorithms and potentially use more than one of these cloud services at the same time (e.g. Microsoft Azure, Google Cloud Services, etc.).

[1] https://aws.amazon.com/ebs/details/

Approach

The key idea that makes this application possible is the lack of reliance on any particular storage system. In the case that I was to work, for example, on physical drives purchased for the project, I would have been limited not only by the variety of approaches to ensure reliability, but also simply by the potential scalability. Purchasing and using storage at large volumes on the cloud is a very scalable approach to this problem that allows for this flexibility in what kinds of algorithms can be run on the storage. Most importantly, the interesting reliability- and performance-driven algorithms can run irrespective of the underlying storage. This makes for a significant amount of wiggle room in terms of the services that I can use for storage but also those that I can offer to users.

Plan

The overall design of the application will be a that of a typical web-application, with the interface being something similar to what one sees with Dropbox or Google Drive. The difference will be the variety of services that can/will be offered, which will include allowing choices of underlying storage system, focuses on reliability over performance (or vice versa), etc.

The first step in the process will be to develop a local command-line version of the application that saves files on the host computer that it runs on, with base RAID-type algorithms implemented. A major focus in these early stages will be on the ease of swapping out the underlying storage. This issue does not have an immediate solution - all of the services that have been mentioned before have different interfaces and APIs, and being able to swap one out for the other while still running a consistent algorithm on top of them is not extremely simple. This will be accomplished by writing code that operates on constants across systems, like streams of bytes.

Additionally, attention to creating correct software-versions of the RAID algorithms that also run efficiently will be central to the application in these beginning stages. As an example, an early choice of Golang for the server was made, motivated by the goal of efficient, parallelizable code that utilizes benefits that RAID algorithms can provide, while maintaining reliability.

Once the base implementation works locally, the leap will be made to run the server publicly on an Amazon EC2 Instance, and plug in Amazon EBS as the underlying storage system. The RAID algorithms will have to be modified slightly, as Amazon EBS works as block-level storage, which provides a great way to get closer to the hardware and provide a more realistic RAID implementation, but also complicates implementation details. After the command-line version is working with true cloud storage and on an Amazon EC2 instance (note that Amazon is chosen simply for its widespread use), the website interface needs to be added into the server to interact with these commands. This will ultimately lead to a finished web-app that people can interact with and upload/download files from.

Execution

Multiple metrics provide ways of evaluating the project: (a) speed and performance vs. hardware solutions, (b) explicit reliability tests with simulated corruption of data, (c) ability to swap out underlying storage, and finally (d) usability of the product. This application tries to accomplish hardware-tasks in software, so ensuring that at least comparable results on performance and reliability are vital. These can be measured with concrete timing and load tests. The ability to swap storage and usability of the product go hand-in-hand, as they are both essential goals. If the application properly inhabits the middle-layer in between storage and the user, then ability to simply add in other storage aside from EBS will indicate successful design.