Cloud Storage Systems

Project - Phase 10 Essay

Tiago Carvalho `fc51034`

Diogo Lopes `fc51058`

Miguel Saldanha `fc51072`

João Roque `fc51080`

João Afonso `fc51111`

Group 14

25/05/2021

Abstract

Cloud Storage is a really complex field, and systems implementing it are growing, both in terms of complexity and scalability. Providing faster service to more and more users.

But everything comes at a cost, and cloud storage systems cannot have everything we wish for, like Consistency, Availability and Partition Tolerance, that is called CAP Theorem, and we must pick a ratio of them. Thankfully there are already a lot of different models and methods to attenuate this problem.

Another issue with Cloud Storage is that the current available solutions, even though they allow developers to make cost-performance trade-offs, they do not handle dynamic workloads very well. Luckily, improvements to these systems are being developed, and, as we will see, it is possible to have Cloud Storage systems that can auto-scale and adapt to the dynamics of workloads, while managing to keep costs as low as possible.

Chapter 1

A brief survey on replica consistency in cloud environments [1]

1.1 State of the art

Databases are, very usually, the core of a cloud service, either for enterprises or actual costumers. With increasing importance it's expected to perform well for its users.

A database is expected to be Consistent, where the latest read expects the most recent write; Available, where every read should receive a valid response (instead of an error); and Partition tolerant, meaning it's expected that the system keeps working even if hardware fails.d

And this can be possible on a single database, even assuring the ACID properties, however, on a cloud database there's synchronization that needs to take place to achieve consistency, and everything comes at a cost, and that's what the CAP theorem states: It is impossible for a distributed data storage to, simultaneously, provide more than two of those three properties (the C.A.P. properties).

Cloud services try to maintain the balance between these 3 properties. If it is important to have consistency, maybe decreasing the response time, or availability, helps with that, to spread changes to other partitions. Or maybe the partitions are not the most important aspect, and then its amount is reduced, meaning there are fewer partitions to update and synchronize, increasing once again the availability.

There are many example of usage, and there's always a sacrifice in either one of the three properties to improve on the others, as we will show in this chapter.

1.2 Consistency

1.2.1 Data/Client-centric consistency models

Between the data center and the clients there's already a choice to be made: should the client be a part of keeping the data consistent?

Each option, obviously, has its own pros and cons. For the Data-centric these are the general options, in order of consistency provided:

- **Week** - There is no "supervision", the first item found is returned.
- **FIFO** - There are no guarantees besides writes from the same process being synchronized, using a First In First Out approach.
- **Causal** - Consistency is only guaranteed between requests with a causal dependency.
- **Sequential** - All operations are serialized in the same order in all partitions, and every operation keeps its received internal order.
- **Strict** - For each received write it needs to be instantaneously spread to other partitions to be updated.

And then for the Client aspect:

- **Eventual** - All updates will eventually propagate far enough that every partition gets updated.
- **Monotonic read** - Guarantees that for each read the result will never be older than the same reads before.
- **Monotonic write** - Different writes from the same request are always processed in the same order.
- **Read-your-writes** - Guarantees, for the same request, that for each write the next read will never return an older result than that write.

- **Writes-follow-reads** - For the same request, each write after a read, guarantees that will be executed on the same or more recent value of the previous read.

1.2.2 Replica consistency methods

As stated a cloud database can have many partitions, and if we are replicating the data it needs to be identical, so any access doesn't get wrong results. These are the methods that are usually applied:

Fixed consistency methods:

- **Event sequencing-based**, simple method that hides replication complexity, focusing on availability.
- **Clock-based strict**, relies on timestamps and avoids concurrent accesses.

Configurable consistency methods:

- **Automated and self-adaptive**, enforces multiple levels of consistency over distinct data.
- **Flexible**, can have distinct approaches that dynamically change their consistency ratio.

Consistency monitoring methods:

- **Consistency verification**, to verify and detect possible protocol or contract violations by the cloud service.
- **Consistency auditing**, a large data cloud and multiple small audit clouds with groups of users that cooperate on a specific job. They share a service-level agreement and the audit cloud verifies if it's violated.

Chapter 2

Autoscaling tiered cloud storage in Anna [2]

2.1 Motivation

A wide variety of cloud-storage systems is available nowadays, and developers can select the one that better suits their application's needs, making cost-performance trade-offs. However, these systems are not very dynamic, which is not ideal when most applications deal with a non-uniform distribution of performance requirements.

2.2 Overview of Anna

Anna is an autoscaling, multi-tier, coordination-free, distributed key-value store service for the cloud that allows system operators to specify service-level objectives (SLOs), like fault tolerance or cost-performance. It is built on AWS components.

The performance of a system depends on the volume of a workload and on whether workloads make a lot of requests to a small subset of the keys or more uniform requests. Anna uses three mechanisms to adapt to these dynamics of workloads:

- **Horizontal elasticity** – Storage tiers can increase/decrease storage capacity and compute and networking capabilities, depending on the volume of a workload, by adding/removing nodes.
- **Multi-Master Selective Replication** – Hot sets (frequently accessed) are replicated onto many machines, with hot keys being

more replicated than cold ones.

- **Vertical Tiering** – Anna is able to promote hot data to the fast, memory-speed tier and demote cold data to cold storage.

2.3 Anna Architecture

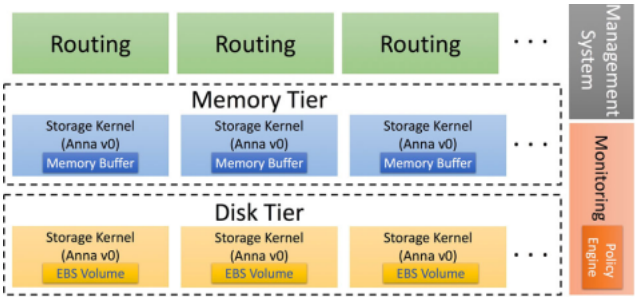

Figure 2.1: Anna architecture.

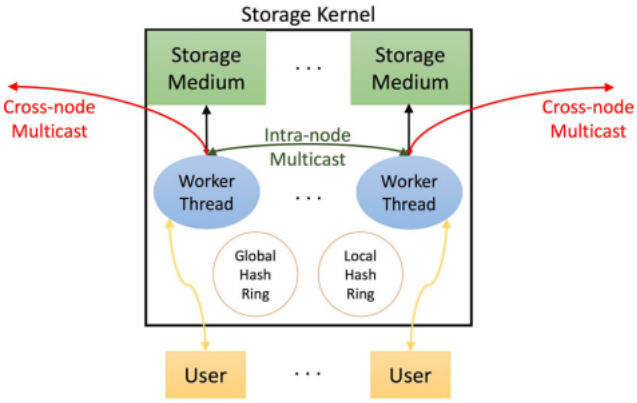

Figure 2.2: The architecture of storage kernel.

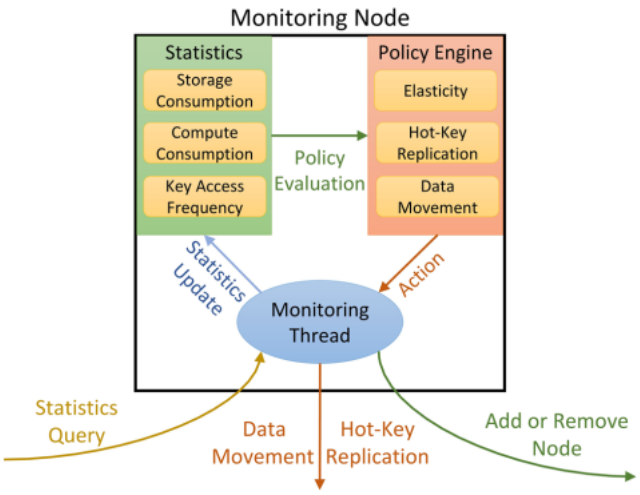

Figure 2.3: The architecture of a monitoring node.

Anna has two storage tiers: one that is fast but expensive, providing RAM cost-performance, and another that is slow but cheap, providing flash disk cost-performance.

The **monitoring system** and **policy engine** are responsible for adjusting the system to the workloads' dynamics and meet the SLOs. The **cluster management system** modifies resource allocation based on the decision of the policy engine (with Kubernetes). **Routing service** is an API that abstracts the internal dynamics of the system.

Each **storage kernel** contains multiple threads that interact with a thread-local storage medium (memory-buffer or disk volume, depending on the tier) and process requests from clients.

Shared-memory coordination and consensus algorithms decrease performance and cause latency and availability issues. Therefore, Anna is coordination-free. Periodically, threads multicast (gossip) updates to others that have replicas of their keys. Conflicts are resolved asynchronously. As a result, Anna exploits multi-core parallelism within a single machine and smoothly scales out across distributed nodes.

For different key replicas, although a set of gossips may be applied in different orders, **Commutativity**, **Associativity**, and **Idempotence** properties ensure that the state of the replicas eventually converges.

Anna uses metadata to efficiently support the mechanisms initially described. Each tier has two **hash rings**. A global one that determines which nodes are responsible for storing each key and a local one that determines the set of threads within a node that are responsible for a key. Each key has a **replication vector** that has the number of nodes in each tier, and the number of threads per node in each tier storing it. Anna also tracks **monitoring statistics**, like the access frequency of each key and the storage consumption of each node.

2.4 Policy Engine

Anna supports three kinds of SLOs: an **average request latency** (ms), a **cost budget** (dollars/h), and a **fault tolerance** (number of replicas that are allowed to fail). If the average storage consumption in a tier has violated configurable upper/lower thresholds, nodes are added/removed. Then data is promoted or demoted across tiers. Next, if the latency exceeds a certain fraction of the latency SLO and memory tier's compute consumption exceeds a threshold, nodes are added to the memory tier. However, if not all nodes are occupied, hot keys are replicated in the memory tier. Finally, if the observed latency is a certain fraction below the SLO and the compute occupancy is below a threshold, the system checks if it can remove nodes to save cost.

2.5 Anna API

API	Description
Get(key)->value	Retrieves the value of key from a single replica
Put(key, value)	Performs an update to a single replica of key
Delete(key)	Deletes key
GetAll(key)->value	Retrieves the value of key from all replicas and returns the merged result
PutAll(key, value)	Performs an update to all replicas of key
GetDelta(key, id)	Retrieves the value of key only when it has changed from the previously queried version, identified by id
Subscribe(key, address)	Subscribes to key and receives the updated value at address

Figure 2.4: Anna API.

GetAll allows users to observe the most up-to-date state of a key. Put relies on asynchronous gossip for the update to propagate to other replicas. If the node crashes before gossiping, the update will be lost.

2.6 Conclusion

Integrating the mechanisms described, Anna becomes an efficient, autoscaling system representing a new design point for cloud storage. In many cases, Anna is orders of magnitude more cost-effective than popular cloud storage services and prior research systems. Throughput increases linearly with cost, meaning that it can get better performance out of the same cost when compared to available cloud storage solutions. Also, the system is able to adapt to dynamic workloads while not violating the SLOs most of the time. Finally, the system is able to recover from node crashes, while not hurting performance too much, since it does not pause, providing high availability to the users.

Chapter 3

Threats and security issues in cloud storage and content delivery networks: Analysis [3]

In this paper is addressed the security threats to Cloud Storage (CS) and one of its uses as Content Delivery Network (CDN). A CDN caches a copy of a resource in near user replicas having in mind demand.

3.1 Issues and challenges of CS and CDNs

As cloud storage threats we have:

- Data breaches due to Misusing of Identity and Management systems.
- Misconfiguration of Control.
- Lack of Cloud Security Architecture.
- Lack of Cloud Security Architecture and Strategy.
- Insufficient Identity, Credential, Access and Key management to access resources.
- Account Hijacking.
- Insider Threat, from current or past employees.
- Insecure API's.
- Weak Control Plane in which security configuration is unknown.
- Metastructure and Applistructure Failures that lead to security flaws.
- Not knowing the normal pattern of utilization of Organization Cloud Services.
- Illicit use by users of cloud resources.

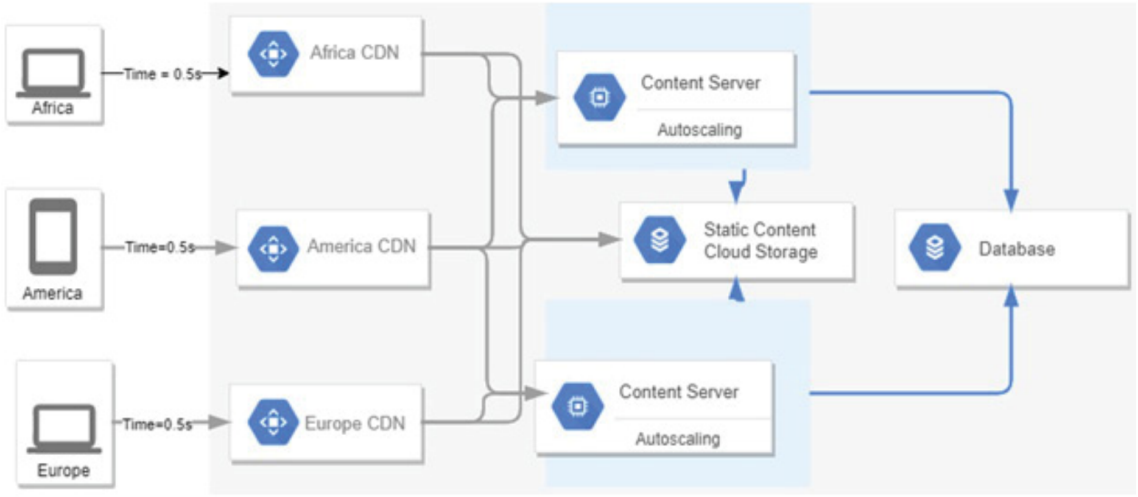

Figure 3.1: TODO

Content Delivery Networks (CDN's) can be used to perform attacks to the Cloud Storage or the service provided. So, as threats we have:

- Outages of provider.
- Internal threat due to misconfiguration or bad management of CDN.
- Malicious insider.
- Law enforcement and legal aspects (breaching data).
- Dynamic Content Attacks that deliver all incoming traffic to Cloud Storage.
- Secure Sockets Layer (SSL)-based DDoS attacks, since all SSL traffic is redirected to Cloud Storage.
- Direct Internet Protocol since attackers can address directly the Cloud Storage, compromising of web apps (due to little protection rules between both).

3.2 Threat model of CS used as CDN

TODO

CDN Threats Risk / Cloud Storage Threats	Outages of a Provider	Internal threat due to misconfiguration of equipment by technical staff	Internal threat due to a bad software deployment	Request header overflow	Law enforcement and Legal aspect	Denial of Service	Risk Level
Data breaches	·	☑	☑	·	☑	·	High
Data loss	☑	☑	☑	·	·	·	High
Insecure API	·	☑	☑	·	·	·	High
Account hijacking	·	☑	☑	☑	·	☑	High
Malicious insider	·	☑	☑	☑	·	☑	Medium
Denial of Service	·	☑	☑	☑	·	☑	High
Insufficiency Due Diligence	☑	☑	☑	·	·	·	High
Shared Technology Vulnerability	☑	☑	☑	·	·	·	High
Hardware failure	☑	☑	☑	·	·	·	Medium
Abuse of Cloud Storage	·	☑	☑	·	·	☑	High
Natural Disaster	☑	·	·	·	·	·	Low
Closure of the Cloud Provider	☑	☑	☑	·	☑	·	Low
Cloud Malware	·	☑	☑	·	·	·	High
Inadequate Cloud Planning/Design	·	☑	☑	☑	·	☑	High

Figure 3.2: Matrix of common content delivery network threats in cloud storage used as content delivery network

3.3 Threat model of CS used as CDN

Chapter 4

Data auditing in cloud storage using Smart Contract [4]

With any distributed storage system, such as cloud storage, there are many issues that one needs to be aware of: consistency, fault tolerance and integrity, to name a few. This paper focus especially on the integrity problem. Despite cloud storage systems using the usual mechanisms to avoid data corruption, like RAIDs and ECCs, there are still problems that arise, and usually it is needed to resort to a Third Party Auditor (TPA). This TPA is responsible to check for the integrity of the data to reassure that nothing has been corrupted. While this works, there are privacy, security and confidentiality concerns when using a TPA since they have access to all the data and things can be compromised. To address this problem, the authors propose the usage of smart contracts to perform this data auditing.

4.1 Smart Contracts in Data Auditing

To put it simply, a smart contract is a piece of code that is deployed on a blockchain for the participating nodes to execute in exchange for gas. Since a blockchain is a fully decentralized system, it avoids entirely the chance of a third party tampering with the data, since the blockchain technology on itself is specialized with dealing with this kind of problems.

4.1.1 Proposed System

To implement it, the researchers proposed an architecture which comprises three elements: the Cloud Service Provider (CSP), the Data Owner and a blockchain. The blockchain used was Ethereum-based since it is the more popular blockchain to support smart contracts. In this architecture, the user sends the data encrypted to the CSP, hashing it beforehand. This hash is then used in the creation of the smart contract, by the user. The smart contract includes information about the data such as the name of the files, hashes, size and details of the data own, as well as other relevant info. This smart contract is then deployed on the blockchain to perform the data auditing periodically. One thing worth noting is that the researchers used an optimized blockchain for data auditing, the DAB (Data Auditing Blockchain), designed for this sole purpose.

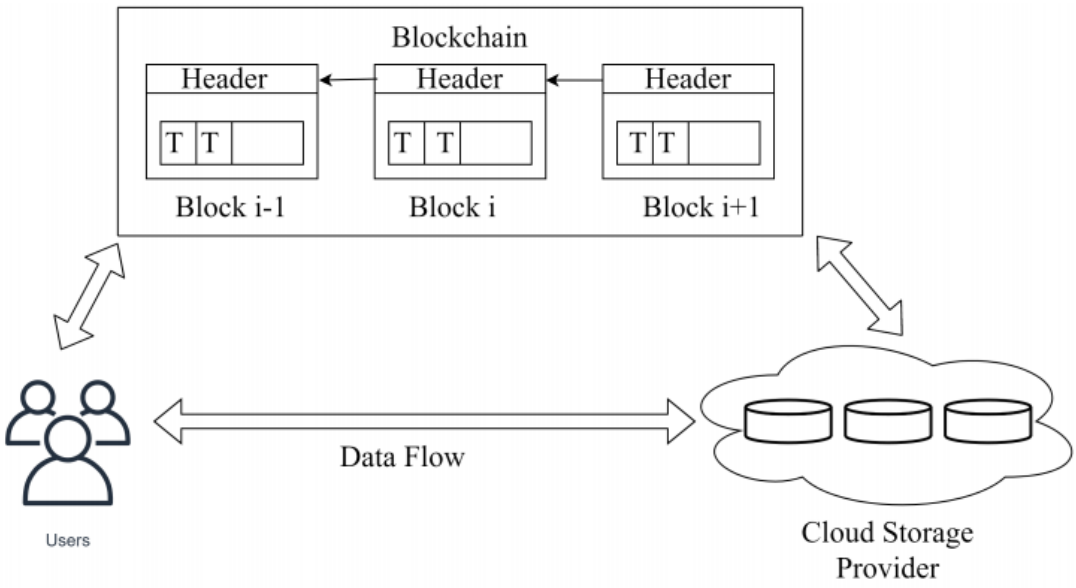

Figure 4.1: Architecture of the proposed system

4.2 Threat model of CS used as CDN

4.2.1 Performance Analysis

After the system was designed and implemented, the authors did an evaluation on its performance, with very positive results. They stated that the system provided reliable and confidential auditing, without requiring any effort from the CSP or user after the smart contract is deployed. The gas costs increased in a linear fashion depending on the number of blocks which is positive.

4.3 Conclusion

In this paper, the researchers presented the problem of data auditing in cloud storage systems and gave a valid solution to it, the usage of smart contracts for data auditing. Its implementation is relatively simple, while having very good results, which is definitely a step in the right direction and could see further developments in the future. While the results achieved by the researchers were good, there are probably optimizations that can be done when developing the system for a real world use.

Chapter 5

Key challenges and research direction in cloud storage [5]

This paper provides a general overview of the issues, challenges and future trends associated with cloud storage and compares the approaches that are being taken by researchers to deal with these problems.

5.1 Issues in cloud storage

- **Data management:** The management of data stored across multiple geo-distributed data centers is associated with certain challenges: scalability, parallelism, recovery, consistency in replication, reliability, and cost.
- **Data partitioning:** Partitioning is meant to improve scalability, optimizes performance, and reduce contention. There are two major types of partitioning techniques: horizontal and vertical partitioning. If fragmented data is distributed throughout all servers to take server capacity into account, then it is possible to achieve load balancing. Partitioning is supposed to solve the following problems: speed and availability.
- **Data placement:** Has an important role in terms of performance in the cloud environment. Some main issues include: Geographically Distant Data Sharing, Client Mobility, Data Interdependencies.
- **Availability through replication:** Process of sharing information to provide availability, reliability, fault-tolerance and accessibility and reduce response time / energy and bandwidth con-

sumption. Data resources can be replicated closer to the application to reduce network latency. Challenges of replication: Data selection, Consistency in replicated data, sophisticated management, load balancing and recovery. Researchers have proposed solutions to minimize the replication cost and maximize the expected availability of objects.
- **Data security:** Issues related to data security/theft, data unavailability, privacy, integrity.

5.2 Open research challenges and future trends

- **Multicloud:** Prevents downtime, facilitates replication and fragmentation and improves security and costs. This approach adds complexity to application management (increased effort needed).
- **DevOps in Cloud:** Bridges the gap between developers and operation team. DevOps tools can assist the management and configuration of resources (such as storage, networking, and computing). This approach improves both speed and productivity.
- **Machine Learning in cloud:** Researchers are taking advantage of distributed environments to research parallelization of machine learning and data mining problems.
- **Big Data in cloud:** There are still many challenges to overcome: scalability, availability, data integrity, data transformation, data quality, data heterogeneity, privacy, and legal issues.
- **IOT in cloud and Fog Computing:** Enables new opportunities: complex analysis, data mining and real-time processing. Fog computing is an alternative to cloud computing that aims to bring the cloud closer to where data is created and acted upon.

To solve these challenges it is necessary to solve the following issues:

- **Fragmentation:** Reducing data migration by providing workload aware dynamic fragmentation techniques.
- **Data placement:** Providing a proper data placement strategy by considering factors like prediction of resource consumption and data locality.
- **Data management:** Design of data management strategy with consideration of buffer management and memory cost.
- **Replication:** Design an efficient and dynamic data replication strategy with consistency and fragmentation strategy. Designing an efficient data replication and consistent management strategy in geo distributed cloud by considering data pattern and network load factors.

5.3 Conclusion

Cloud storage still presents many problems that have yet to be solved. As such, researchers have been taking into consideration factors like: optimizing monetary cost, reducing network latency, optimal storage, increasing performance or optimal resource utilization (among others). It is also important to take into consideration that the way in which these problems are addressed will depend on the characteristics of the application in question.

References

[1] R.A. Campêlo, M.A. Casanova, D.O. Guedes, and et al. A brief survey on replica consistency in cloud environments. *J Internet Serv Appl*, 11(1), 2020.

[2] C. Wu, Sreekanti V., and Hellerstein J.M. Autoscaling tiered cloud storage in anna. *The VLDB Journal*, 30:25–43, 2021.

[3] JDK Waguia and A. Menshchikov. Threats and security issues in cloud storage and content delivery networks: Analysis. *2021 28th Conference of Open Innovations Association (FRUCT) and 2021 Open Innovations Association (FRUCT)*, 28:194–199, January 2021.

[4] MM Lekshmi and N. Subramanian. Data auditing in cloud storage using smart contract. *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), Smart Systems and Inventive Technology (ICSSIT)*, 3:999–1002, August 2020.

[5] MNS Gajjam and DT. Gunasekhar. Key challenges and research direction in cloud storage. *Materials Today: Proceedings.*, January 2021.