

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

TODO abstract

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 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 something fails, like a database node, network problems, etc...

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 the three properties stated before (Consistency, Availability, and Partition Tolerance).

Cloud services try to maintain the balance between these 3 properties. If it is important to have consistency, maybe decreasing the response time, availability, helps with that, spreading 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 Models

1.2.1 Data/Client-centric

Between the data center and the clients there's already a choice to be made, if the client should be a part of keeping the data consistent, or not.

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

- Weak - 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 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 client are always processed in the same order.
- Read-your-writes - Guarantees, for the same Client, that for each write the next read will never return an older result than that write.
- Writes-follow-reads - For the same Client, each write after a read, guarantees that will be executed on the same or more recent value of the previous read.

1.3 TODO

TODO

Chapter 2

Autoscaling tiered cloud storage in Anna [2]

2.1 Motivation

A wide variety of cloud-storage systems is available today, 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 are able to scale elastically, increasing/decreasing storage capacity and compute and networking capabilities, depending on the volume of a workload, by adding/removing nodes and repartitioning data among them.
- **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

TODO INSERT IMAGES

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 a client-facing 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 other threads that maintain 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 requires maintaining certain metadata to efficiently support the mechanisms initially described. Every tier has two **hash rings**. A global hash ring that determines which nodes in a tier are responsible for storing each key and a local hash ring that determines the set of worker threads within a single node that are responsible for a key. Each individual key K has a **replication vector** that has the number of nodes in each tier storing K, and the number of threads per node in each tier storing K. Anna also tracks **monitoring statistics**, such as 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 certain 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 objective and the compute occupancy is below a threshold, the system checks if it can remove nodes to save cost.

2.5 Anna API

TODO INSERT IMAGE

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, not plateauing, meaning that it can get better performance out of the same cost when compared to the current 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]

TODO

Chapter 4

Data auditing in cloud storage using Smart Contract [4]

TODO

Chapter 5

Key challenges and research direction in cloud storage [5]

TODO

Bibliography

- [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.