Consistency as a Service: Managed Consistency Flexibility for Global-Scale Datastores on Clouds

(Work-in-Progress Paper)

Author1[*], Author2[†], Author3[†]
Department
Organization
Address1[*]; Address2[†]
{xxx, xxx, xxx}@email.com

ABSTRACT

Consistency becomes a critical aspect in cloud areas as the data storage trend migrates from centralized databases on premises to distributed datastores on clouds. Different datastores offer various built-in consistency models based on their implementations, e.g. etcd in Kubernetes provides strong consistency. However, a user may have a preference on a particular type of datastore given specific business needs, but may require different consistency levels, which cannot be satisfied on today's cloud. This paper proposes a Consistency as a Service (CSaaS) which provides users the flexibility of choosing any desired consistency models along with the desired compatible datastores. As a cloud service, CSaaS is designed for potential future use cases which are discussed in this paper as well. The feasibility of the proposed CSaaS is validated by a prototype system.

CCS CONCEPTS

• Information systems • Information storage systems • Storage architectures • Cloud based storage

KEYWORDS

Consistency, Replication, Cloud, Services

1 Introduction

Nowadays cloud computing has emerged as a paradigm that attracts enterprises, institutions and users to host services on clouds, e.g., e-commerce websites or real-time data analytics. Cloud adoption has been expanded in the past two decades and has been accelerated since the pandemic in 2020. Data warehouse and Database as a Service (DBaaS) become the most two utilized Platform as a Services (PaaS) revealed in a recent survey [1]. The rapidly growing data-related services on clouds raise new challenges and requirements to cloud-based data services compared to traditional on-premises data hosting and management.

High availability requires cloud service providers to replicate data across their physically separated servers or racks, or even across geographically distributed datacenters. Consistency among all the data replicas becomes an important technical challenge in distributed cloud environments. Various databases or datastores offer different built-in consistency levels, ranging from eventual consistency to strong consistency (strong consistency contains multiple specific models by varied definitions, e.g., linearizability, sequential consistency.). For example, etcd, the key-value (KV) store adopted in Kubernetes provides linearizability for all the basic operations except watch [2]; DynamoDB, as an Amazon's cloud-native database, provides eventually consistent writes and strongly consistent reads [3].

Although some databases can provide configurable consistency at a certain level, e.g., DynamoDB supports eventually consistent reads and strongly consistent reads, such limited capability still cannot satisfy cloud users' business demands in today's market. From users' perspective, they may have their own preferences or mature adoption on a certain database, but the built-in consistency model cannot meet the requirements when scaling or migrating to cloud. From database vendors' or cloud providers' perspectives, consistency models are predetermined according to the targeted use cases and implemented as a part of a database. Any specific business demands on consistency must compromise with vendor lock-in databases' capability.

This paper proposes a Consistency as a Service (CSaaS) to fill in the gap between varying users' needs on data consistency and vendors' inflexible database solutions. CSaaS provides a cloud service which allows users to choose any desired consistency models along with the desired compatible storage supported by a cloud platform. While there exist some previous literatures which discussed tunable or configurable consistency parameters with a particular database, this work, to the best of the authors' knowledge, is the first effort to design a generic cloud service in this area.

In this paper, Sec. 2 reviews all the related literatures; Sec. 3 presents the architecture and design of the proposed CSaaS; Sec. 4 discusses two future uses cases use CSaaS considering real business needs; Sec. 5 describes the validation of CSaaS using a proof-of-concept prototype; Sec. 6 summarizes the paper and discusses the future work.

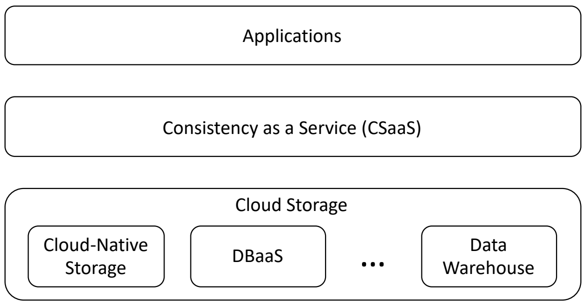

Figure 1: A High-level Overview of CSaaS.

2 Related Work

Many previous works have explored to seek for a balanced or tunable consistency model for distributed storage systems. In [4], a consistency model called explicit consistency was proposed to achieve causal consistent in data correctness, while providing similar latency to an eventually consistent system. A geo-distributed actor system is proposed in [5] to improve performance by caching actor states in one or more datacenters, and it supports updates with a choice of linearizable and eventual consistency. A layered middleware for service-oriented systems was propose in [6] to handle the trade-offs between consistency and scalability. A key-value storage system called Pileus was developed to offer a diversity of consistency guarantee options for distributed replicas in [7]. However, the flexibility on consistency models provided in these works results in either a tuned and compromised consistency model or some limited configurable parameters within a static storage system.

The terminology 'Consistency as a Service' (abbreviated as CaaS in [8]) was proposed in a literature in 2014 [8]. In [8], the CaaS uses two-level auditing structures, local auditing and global auditing, to audit causal consistency in distributed clouds. Although this paper shares a same term, the definition and functionality are fundamentally different from that in [8]. The Consistency as a Service in this paper is mainly for providing consistency flexibility, and auditing is only one module in the service. The reason of using this naming convention is exactly following that of Infrastructure as a Service (IaaS) or DBaaS, where cloud vendors provide a service framework, and allow users to choose service entities, e.g., Ubuntu or Fedora as an instance image in IaaS.

3 Consistency as a Service (CSaaS)

Consistency as a Service is designed as a cloud service to provide clients flexibility of adopting desired consistency level along with the preferred storage solutions. In this section, a high-level overview is introduced first, followed by the discussion of the detailed modules.

3.1 CSaaS Overview

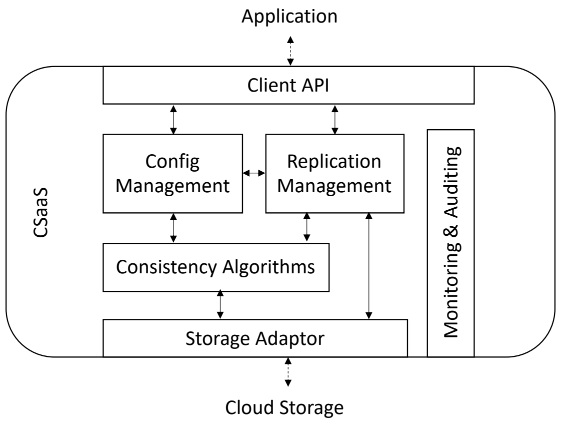

Figure 2: CSaaS Modules.

Today, clients directly deploy applications and interact with cloud storage to manage and operate their data on clouds. Cloud storage includes cloud-native storage, DBaaS, data warehouse, or any other forms of storage supported on clouds. The replication and consistency are controlled and managed within the deployed storage solutions.

CSaaS resides between applications layer and cloud storage layer, as illustrated in Fig. 1. The functionalities of replication and consistency are extracted and decoupled from the underlying cloud storage and are managed by the CSaaS layer. Clients have the flexibility of configuring different consistency levels along with the deployed storage.

With the system architecture in Fig. 1, the design of cloud storage becomes simpler since the replication and consistency are decoupled and placed to a higher layer. Additionally, CSaaS can coexist with the storage's replication and consistency even if the default built-in consistency is not disabled. CSaaS aims for a large scope and scalability, while the storage's built-in consistency can serve a small and regional cluster, forming a hierarchical or tiered structure.

Since the replication and consistency are implemented and managed by CSaaS, operations and interactions with data, e.g., reads and writes, must be through the APIs provided by CSaaS instead of the storage layer directly, to guarantee the ensured consistency level for clients.

3.2 Module Design

CSaaS consists of Client API, Config Management, Replication Management, Consistency Algorithms, Storage Adaptor, and Monitoring and Auditing, which are shown in Fig. 2. This subsection discusses each module in the following.

3.2.1 Client API. Applications submit requests through Client APIs to interact with the data. Here a subset of the basic APIs is listed and described. Note the semantics of APIs here are abstracted for the designing purpose instead of the real-implemented data structure.

`setConsistency(consistency_level, storage_type)`: Set the expected consistency level with selected storage. For example, `setConsistency(linearizability, redis)`. The `consistency_level` and `storage_type` must both be supported by the cloud platform.

`setReplica(num)`: Set the number of replicas. The given number must satisfy the requirements of the selected storage with the selected consistency level.

`read(consistency_mode, read_type, read_object)`: Query and read data. `consistency_mode` represents the consistency level supported by the set consistency, e.g. linearizably consistent read, or sequentially consistent read. `read_type` represents the type of a read, e.g. a single query, a range query, or a SQL-like query. `read_object` represents an object or a command for the query request, e.g. a key or a SQL query sentence.

`write(consistency_mode, write_type, write_object)`: Write data to storage. The three fields are similar to the definition of **`read()`** described above.

Some other data related operation APIs include **`delete(), list(), watch()`**, etc. Given the limited space in the paper, these APIs are not expanded here.

3.2.2 Config Management. Config Management module is responsible for storing, hosting and managing settings, configuration parameters, and module-specific information. The consistency and storage settings are stored in Config Management. The replica setting and replica instance information, e.g., IP address, port number, node role (e.g., primary or backup), etc., are stored here. The replica information is stored, updated, or deleted by interacting with Replication Management module. The clients' consistency and storage settings are consumed by Consistency Algorithms module. The consistency model specific parameters are stored by Consistency Algorithms as well, e.g., the chain configuration is saved if a chain replication algorithm is used.

3.2.3 Replication Management. Replication Management module is responsible for creating and managing replicas. This module heavily interacts with Consistency Algorithms module and Storage Adaptor module. For a write request passed by the Client API, the required number of replicas are created and operated following the selected consistency algorithm. For a read request passed by the Client API, a corresponding read request associated with the selected storage instance is passed to Storage Adaptor module. For example, a linearizably consistent read is translated into a read request from the tail node of a chain by Replication Management, if a chain replication algorithm is used.

An optional function for Replication Management module is sharding. Sharding algorithms, e.g., consistent hashing, can be adopted in this module. An alternative method is to interact with the output of the sharding algorithm from the underlying storage through Storage Adaptor.

3.2.4 Consistency Algorithms. Consistency Algorithms module provides and implements all the supported algorithms at different consistency levels. The granularity of a consistency algorithm should be at operation level, e.g. `(RAFT: {linearizable_read, sequential_read}, {linearizable_write, sequential_write})`. Categorizing consistency models can adopt some widely accepted criteria [9]. Here a subset of the common consistency models is listed in a descending order of consistency strongness- *linearizability, sequential consistency, causal consistency, strong eventual consistency, eventual consistency*. An implemented algorithm provides a certain consistency level or multiple consistency levels. As a few examples, chain replication algorithm [10] and primary-backup algorithm [11] can achieve linearizable writes, and quorum-based algorithm, e.g., RAFT, can achieve linearizable or sequential reads [12]. This module controls and operates the behaviors among replicas to achieve the expected consistency, and the actual operations (e.g. read/write) are passed through Storage Adaptor to the underlying storage instances.

3.2.5 Storage Adaptor. Storage Adaptor module is the interface between all the other modules in CSaaS and cloud storage. The basic operations and other control or management functions need to be exposed at storage instance and be callable from Storage Adaptor module. This module needs to integrate all the interfaces to the supported storage on a cloud platform.

3.2.6 Monitoring and Auditing. This module monitors the health of all the modules and participating entities. The consistency correctness and performance are audited by this module to meet guaranteed service-level agreement (SLA). Once there is any unhealthy activity, unusual event, or SLA violation, this module generates alerts, initiates auto-recovery process, or reports to the operation teams.

4 Use Cases

This section describes two potential use cases that use CSaaS. The use cases discussed here both originate from real business needs in multiple ongoing projects from the authors' organizations.

Global KV Service with Strong Consistency for Containers- Similar to etcd in Kubernetes, all of the information of about a pod's state is stored in a KV service. Differently the KV service is in a global scale, which is across datacenters and regions, possibly even across continents. The KV service has an abstract layer for sharding and uses CSaaS to manage and operate all the replicas globally. Strong consistency can be set and achieved in CSaaS with an in-memory KV store, e.g. Redis. The KV store can be changed according to performance evaluation or engineering requirements. Applications or services read, write, and watch the values for the

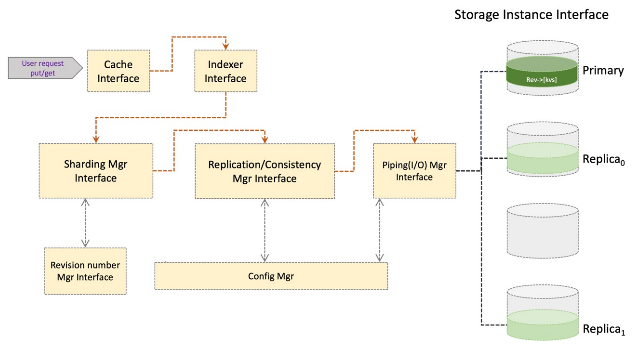

Figure 3: PoC Architecture.

interested keys in the KV service. This global KV service can be adopted by any current major cloud vendor, serving as a component of the global cloud infrastructure.

Datastore Supporting Global Virtual Conference with 10K Attendees- The need of hosting international virtual conference or venue has been expanding quickly since the pandemic. Distributed datastores for storing and maintaining the conference's and each attendee's state are required when involving a large number of users from all over the world. The information stored in the datastore requires different levels of consistency. Keynote announcements or presentation changes need to keep strong consistency among all the datastores. An attendee located in US or China, must receive such system message in time. The number of joined attendees in ongoing sessions needs to have strong consistency so that the organizer can track if the technical requirements can be met, or user experiences can be guaranteed. However, the shared content after a presentation for the subscribed attendees may only need eventual consistency. The datastore is targeted at supporting an international virtual conference with 10,000 attendees online.

5 Prototype Validation

A proof of concept (PoC) prototype has been developed to validate the feasibility of the proposed CSaaS. The source code of the PoC, written in C++, is hosted on GitHub [13]. The architecture of the PoC is shown in Fig. 3.

The prototype has implemented the configuration management, replication management, consistency algorithms, which are the three core modules in the proposed CSaaS. Two consistency algorithms- chain replication [10] and GFS-like primary-backup replication [11], are implemented as examples in the prototype system. Linearizably, sequentially and eventually consistent write and read can be selected in the prototype. In this system, a set of regular interfaces are exposed to end users such as CRUD, range query and list-watch. Users are free to choose the consistency models based on their needs independent of their backend store choices. A suite of data store adaptors is provided by the prototype to work with a wide variety of data storage solutions.

6 Summary and Future Work

This paper proposes a new cloud service Consistency as a Service (CSaaS) to manage data consistency for distributed storage systems on clouds. Users have the flexibility of choosing preferred consistency level based on business needs, avoiding vendor-lock-in storage solutions as well. The architecture of CSaaS and two potential use cases are discussed. As an ongoing work, a prototype system was developed to validate the feasibility and functionality of the proposed service.

The future work is to develop an alpha version of the proposed CSaaS and benchmark its performance on clouds in a real-world distributed environment. A tested and reviewed version will be published and open sourced under XXXXX (anonymous for submission) project [14] by Linux Foundation Projects.

REFERENCES

[1] Flexera, 2022. Flexera 2022 State of the Cloud Report. https://resources.flexera.com/web/pdf/Flexera-State-of-the-Cloud-Report-2022.pdf

[2] etcd. KV API guarantees | etcd. https://etcd.io/docs/v3.6/learning/api_guarantees/

[3] AWS. Amazon DynamoDB. https://docs.aws.amazon.com/

[4] Valter Balegas, Sergio Duarte, Carla Ferreia, Rodrigo Rodrigues, Nuno Preguica, Mahsa Najafzadeh, Marc Shapiro. 2015. Putting consistency back into eventual consistency. In *Proceedings of the 2015 European Conference on Computer Systems (EuroSys'15)*. ACM Sigops/Europs, Apr 2015, Bordeaus, Fance. Pp.6:1-6:16. https://doi.org/10.1145/2741948.2741972

[5] Philip Bernstein, Sebastian Burckhardt, Sergey Bykov, Natacha Crooks, et al.. 2017. Geo-distribution of actor-based services. *Proc ACM Program Lang*. ACM, vol. 1, no. 26, pp. 1:107–1:107, Oct. 2017.

[6] Tao Chen, Rami Bahsoon, Abdel-Rahman Tawil. 2014. Scalable service-oriented replication with flexible consistency guarantee in the cloud. Journal of Information Sciences. Elservier, vol. 264. pp.349-370, 2014.

[7] Douglas Terry, Vijayan Prabhakaran, Ramakrishna Kotla, et al. Consistency-based service level agreements for cloud storage. 2013. In *Proceedings of the 24th ACP Symposium on Operating Systems Principles (ACM SOSP'13)*. ACM. Farmington, Pennsylvania, US, Nov. 2013.

[8] Qin Lin, Guojun Wang, Jie Wu. 2014. Consistency as a service: auditing cloud consistency. *IEEE Transactions on Network and Service management*. IEEE, vol. 11, no. 1, pp. 25-35, Mar. 2014.

[9] Robson Campelo, Marco Casanova, Dorgival Guedes, Alberto Laender. 2020. A brief survey on replica consistency in cloud environments. *Journal of Internet Services and Applications*. Springer. 2020. https://doi.org/10.1186/s13174-020-0122-y

[10] Robert Renesse, Fred Schneider. 2004. Chain replication for supporting high throughput and availability. In *Proceedings of 6th Symposium on Operating Systems Design & Implemenation (OSDI'04)*. USENIX. Pp.1-14. 2004.

[11] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. 2003. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03).* ACM, Bolton Landing, New York, USA, pp. 1-15. Oct. 2003.

[12] Diego Ongaro, John Ousterhout. 2014. In search of an understandable consensus algorithm. In *Proceedings of 2014 USENIX Annual Technical Conference (ATC'14)*. USENIX, Philadelphia, PA, USA, pp. 305-319, Jun. 2014.

[13] Github repo: https://github.com/XXX/XXXX (hide per anonymous requirement for submission).

[14] LF open-source project. https://XXX.XX. (hide per anonymous requirement for submission).