WHITE PAPER

Under the Hood: Redis CRDTs (Conflict-free Replicated Data Types)

Cihan Biyikoglu, VP Product Marketing, Redis Labs

CONTENTS

Introduction

Twelve years after the original CAP theorem, Eric Brewer wrote a great article on how the rules have changed on CAP. The summary is that using CRDTs (Conflict-free Replicated Data Types), one can create a new balance between C, A and P—commonly referred to as "strong eventual consistency."

Redis Enterprise implements CRDTs using a multi-master replication architecture. Redis CRDTs provide great benefits:

- Sub-millisecond latency reads and writes for globally distributed apps: Redis CRDTs create a globally spanning database that reaches across multiple datacenters, providing low latency reads and writes to apps in each datacenter.
- Uninterrupted availability with active-active multi-region deployments: With Redis CRDTs, applications gain the ability to read and write without interruptions across multiple data centers in distant regions. Read and write availability is continuous, even when some data centers are completely unavailable or the network between data centers splits communication.
- Simplified development with smart auto-conflict resolution: You can streamline the development of complex, mission-critical applications with global workloads using built-in Redis types and commands while Redis CRDTs automatically handle conflicting concurrent reads and writes.

Let's look under the hood to understand how Redis CRDTs provide high performance, availability and smart conflict resolution.

Under the Hood: Redis CRDT

Redis Enterprise Overview

Redis Enterprise is a distributed database platform composed of identical nodes that are deployed within a datacenter or stretched across local availability zones. Each node contains services that compose a management path (depicted in the blue layer in Figure 1 below) and data access path (depicted in the red layer in Figure 1 below).

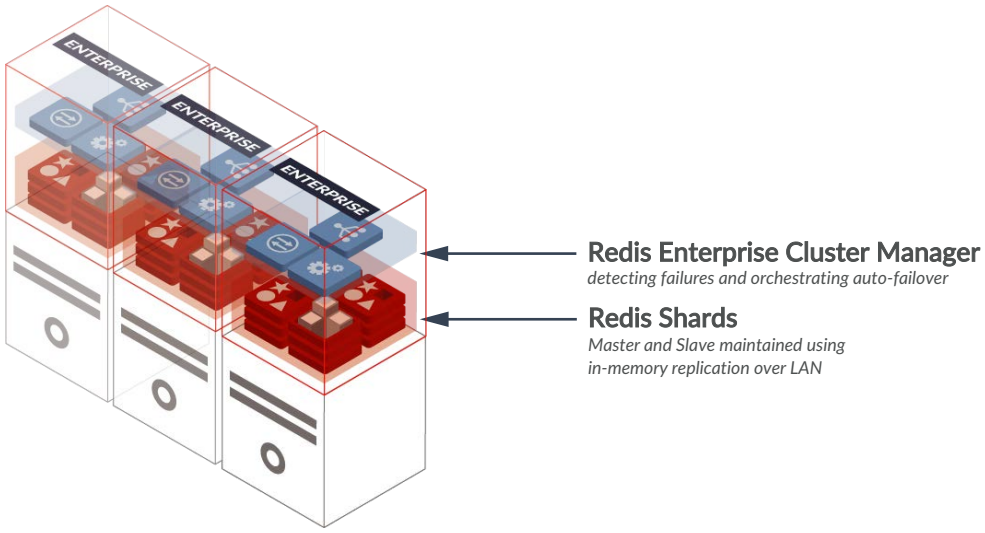

Figure 1. *Redis Enterprise nodes, with blue tiles representing the management path and red tiles representing the data access path with Redis as the shards.*

- The Management path includes the cluster manager, proxy and secure REST API/UI for programmatic administration. The cluster manager is responsible for orchestrating the cluster and the placement of database shards, as well as detecting and mitigating failures. The proxy helps scale connections with smart connection management. The management path coordinates the creation and ongoing management of Redis CRDTs.
- The Data Access path is composed of multiple master and slave Redis shards. Clients perform data operations on the

master shard. Master shards maintain slave shards using the in-memory replication for protection against failures that may render the master shard inaccessible. There can be multiple databases in a single Redis Enterprise cluster and each database can contain multiple shards.

Redis CRDT Architecture

Redis CRDTs are implemented in Redis Enterprise using a global database that spans multiple clusters. These globally spanning databases are called "Conflict-free Replicated Databases" or "CRDBs."

Global CRDBs are made up of local databases in each participating cluster called "member CRDBs." CRDBs establish a bidirectional replication between each one of the member CRDBs. It is important to note that from an application perspective, member CRDBs that applications connect to for reading and writing data behave just like a regular Redis database.

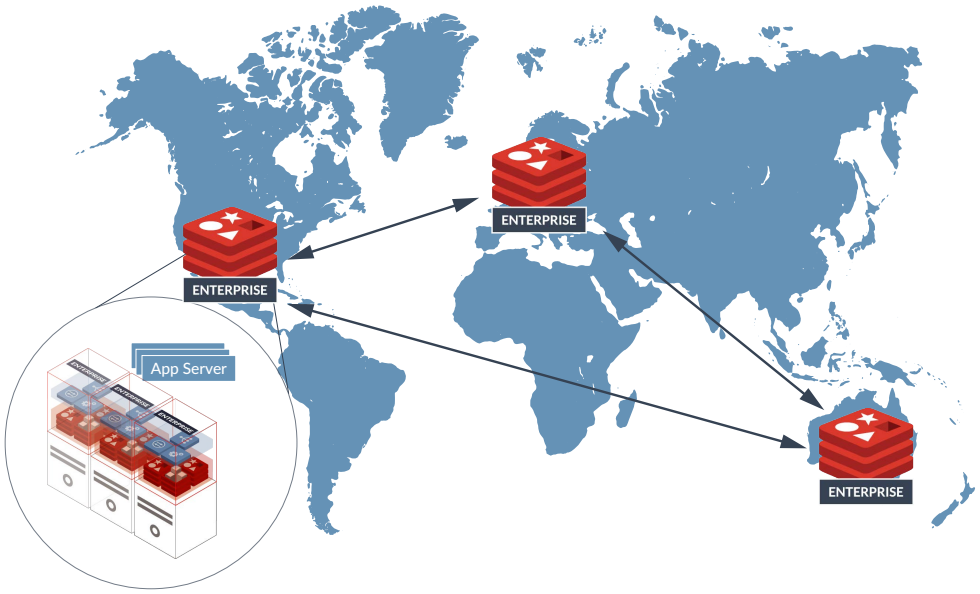

Figure 2. Bidirectional replication for geo-distributed, active-active reads and writes topology

Achieving High Performance Reads and Writes with CRDBs

Locally deployed applications connect to member CRDBs in participating clusters. This delivers the same sub-millisecond latencies Redis applications enjoy with CRDBs. Applications can read and write to the same keys across member CRDBs concurrently. With the bidirectional replication, changes made to each member CRDB are replicated and conflicting concurrent write operations are resolved based on the defined rules per data type.

Bidirectional Replication and CRDB Syncer

CRDB Syncer, is a process that coordinates the WAN-based replication between all member CRDBs. The Syncer process sits in all the clusters that are participating in the CRDB deployment and is responsible for both the initial and ongoing sync of data.

CRDB Syncer establishes concurrent connections to the other member CRDBs and reads changes from the slave shards. This is done to provide better assurance that only data that has been locally replicated and durable is replicated to the other member CRDBs. The changes are communicated in a streaming fashion and are applied to the local member CRDB. Data on the wire is compressed. This can provide bandwidth savings when crossing long distances via the WAN.

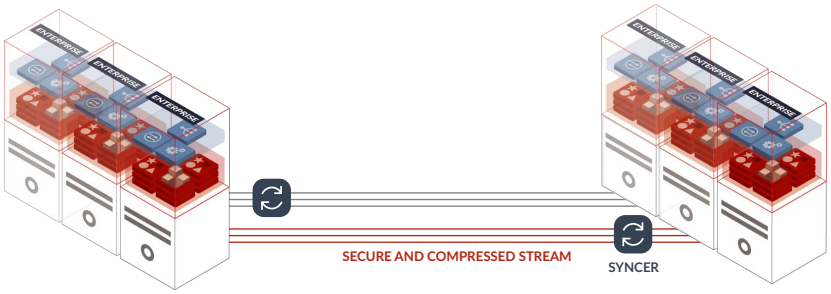

Figure 3. Architecture of WAN-based Replication with CRDB Syncer

CRDB Syncer automatically handles interruptions due to network failures or topology changes in source or destination databases. During interruptions, Syncer continues to poll and resume replication. In some cases, resuming replication may require a full sync of a stale member CRDB.

Uninterrupted Availability with Active-Active Multi-Region Deployments

Datacenters can experience either momentary or long-lasting failures, or simply undergo maintenance. Unlike the auto-failover Redis Enterprise performs under node, rack or zone failures, the geo-failovers with active-active deployments are not automatically executed by Redis Enterprise. Applications can simply do the geo-failover by redirecting their users to one of the participating clusters in another geography.

Geo failovers are a problem gracefully solved by CRDTs. Without CRDTs, handling geo failover is complex and can result in lost updates. Here is an example:

Imagine we have a shopping-cart maintained in a Redis Set and the following events take place over time:

time	US Data Center	EU Data Center
t1	SADD cart1 “costume”	
t2	US Data Center Fails - Sync Fails	
t3		SADD cart1 “mask”
t4	US Data Center Recovers - Resume Sync	
t5	SMEMBERS cart1 “costume” “mask”	SMEMBERS cart1 “costume” “mask”

- User in the west coast maintains a shopping cart under the key “cart1.” Cart is updated at time “t1” with the user adding a new product called "costume".
- At time “t2,” before the replication can sync the update to the shopping cart “cart1” to the east coast data center, the west coast data center fails. The user is then directed to the east coast data center for continuous availability.
- At time “t3,” the east coast data center has not yet received the item “costume” in the shopping cart “cart1.” However, the user adds a new product to the cart called “mask.” At this point the west coast data center shopping cart contains product "costume" and the east coast data center shopping cart contains product “mask.”
- At time “t4,” the west coast data center recovers and bidirectional replication resumes. If you happen to be using a database that does conflict resolution using mechanisms like LWW (last-writer-wins), the shopping cart can only contain either only "costume" or "mask" because only one of your updates to the “cart1” shopping cart key will survive. The result would be a lost update to the shopping cart! CRDTs make sure no update is lost to the shopping cart. With Redis CRDTs, the shopping cart correctly reflects all items user added because the SADD method with the Redis SET data type was used.

Simplified Development with Smart Auto-conflict Resolution

CRDTs provide a mathematical model for handling conflicting writes. The goal of CRDTs is to provide well defined conflict resolution behaviors for common use cases. Combined with Redis types and commands, CRDTs detect the "developers intent" during conflict resolution. For example, when using methods like INCR, Redis CRDTs behave in a way that allow you to build distributed counters. When using SET type with SADD or SREM methods, Redis CRDTs use a "Add-Wins with Observed-Remove Set" behavior defined in CRDTs.

Overall, Redis CRDTs simply allow users to focus on developing their business logic instead of coding custom conflict resolution logic to handle all kinds of concurrent write scenarios. Developers can pick the data type and conflict resolution rules that work for their applications and Redis CRDTs automatically resolve the conflicts using the well defined rules for each type and command. To achieve this effect, Redis CRDTs keep additional metadata per type. This additional metadata is later used in bidirectional replication to synchronize all participating clusters. You can find the detailed developer's guide and rules and behaviors governing conflict resolution for Redis types with CRDBs in Redis Enterprise Documentation.

Getting Started with Redis CRDTs

Getting started with Redis Enterprise and Redis CRDTs is simple. Please visit the following links:

- Getting started with Redis CRDTs
- Administering CRDBs
- Developing Applications with CRDBs
    - Working with Redis Hashes in CRDBs
    - Working with Redis String in CRDBs
    - Working with Redis Sets in CRDBs

For an overview of how active-active geo-distributed applications work, you can also watch the webcast featuring Redis CRDTs.

700 E El Camino Real, Suite 250
Mountain View, CA 94040
(415) 930-9666
redislabs.com