redislabs
home of redis

whitepaper

Why Your MongoDB Needs Redis

Author: Itamar Haber, Chief Developer Advocate, Redis Labs

Table of Contents

Executive Summary	2
Introduction	2
Popular Redis Use Cases	2
MongoDB and Redis	3
Customer Examples of Joint MongoDB and Redis Usage	4
Conclusion	6

Executive Summary

MongoDB is one of the most popular NoSQL databases on the planet. Redis, another extremely popular in-memory database, is used very frequently to augment its capabilities. This whitepaper discusses some reasons to use Redis with MongoDB, and presents customer use cases where the two technologies are used side by side – often to solve Big Data velocity problems.

Introduction

MongoDB is a document-oriented, disk-based database. Redis is an in-memory, persistent, data structure store. Both databases fall into the NoSQL category – in that they are used to solve computing and data storage problems for which use of traditional RDBMS solutions would be like using a hammer to kill an ant.

But Redis and MongoDB are quite different and optimized for different uses. Disk based MongoDB is optimized for operational simplicity, schema-free design and very large data volumes.

Redis, on the other hand, is an in-memory, NoSQL data structures store, frequently used as a database, cache, or a message broker. Unlike other in-memory stores, it persists data and is remarkably versatile due to its data structures (sets, sorted sets, hashes, lists, strings, bit arrays). Redis enables developers to perform common operations on these structures at the lowest possible complexity and highest possible performance. In other words, Redis is purpose-built for performance and simplicity.

Recent benchmarks have found Redis to be the best database for millions of operations/second with sub-millisecond latencies and with the least amount of hardware. Read about the benchmarks here.

Popular Redis Use Cases

Redis use cases are numerous – but some of the popular ones include:

1. **User session storage:** When you have hundreds, thousands or millions of user sessions to manage and session data needs to be instantly retrievable on the server-side for lightning fast decisions, Redis' hash data structures and automatic key expiration features deliver the lowest complexity and fastest response times.
2. **Real time analytics:** When you have thousands or millions of users playing a game or users collecting points on an ecommerce site, implementing real time rankings or leaderboards becomes simple with Redis' in-memory sorted sets and set operations. Simple Redis commands help you achieve blazing fast retrieval of top users, user ranks, high scores and more.
3. **As a message broker/for queues:** When you have many items to be worked on in a queue, by several different worker processes, a single list operation with Redis can manage complex operations like removing a job from one queue and adding it to another. Managing queues as lists is a very popular Redis use case.

4. **In-App Social functionality:** Lets say you have tons of items and you manage sorting them with tags. Single Redis set commands can execute complex set membership questions rapidly for operations such as “all articles with a particular tag” or “all articles intersecting on specific tags.” Other use cases include computing friends of friends, or users with similar preferences.
5. **High speed data ingestion:** Given how fast Redis is, another popular use case is to act as a data ingestion buffer. For high volume mobile, Internet of Things (IoT), API or web applications where throughput is required to be very high, Redis can store data points before they are processed by the application or stored in slower disk-based databases.

Of course a classic use case of Redis is as a cache, but Redis is an intelligent cache and usage of the data structures can lead to faster, better application performance. Redis also includes the ability to run Lua scripts, which comes handy when crunching through massive computations, in real time.

MongoDB and Redis

A very common use case relevant to MongoDB users is Redis’ usage as a data ingestion buffer. MongoDB is awesome at handling any volume or variety of data being generated – however, extreme velocity poses a challenge to any disk – based database. When you have millions of data points coming at your application, using a disk-based database directly simply means that you run the risk of losing data or creating excessively long queues to store data yet to be processed.

Mobile, IoT, APIs and massively online applications usually have this problem – there is just too much data being generated for it to be digested easily by any disk-based database, even if it is for storing and batch analytics. Real-time, instantaneous processing is something that can’t be achieved with disk based databases, without throwing a lot of expensive hardware at the problem.

This is where Redis comes in. With Redis running very efficiently in memory, it becomes radically simple to enable real-time analysis of fast moving data by shifting some of the processing to Redis.

As the first-responder database, not only does Redis get you very efficient and high performance data processing (1.5 Million operations/sec at sub-millisecond latency with a single AWS EC2 server), but you also get a stable, reliable database that can be persisted to disk as needed.

Another scenario is when several documents in MongoDB are being updated simultaneously or in very close succession (see the customer example below). Every update triggers a write to the database, requiring you to maintain multiple shards and throw a ton of hardware at the problem. By using Redis, which handles extremely high throughput at sub-millisecond latencies, instead, you can manage the updates in-memory and consolidate writes to the disk-based database. This approach reduces hardware needs and creates savings on operational costs.

Customer Examples of Joint MongoDB and Redis Usage

The below examples describe customers using Redis with MongoDB in real world scenarios:

Real time analytics service provider

This company provides a "Google Analytics" like service to large websites and web-based applications. Its flagship product is a platform that provides insights into user behavior by collecting millions of event data points, and aggregating and segmenting this data to glean intelligence.

The primary data storage used is MongoDB, however, the company also uses Redis for real-time analysis, calculations and display of the high volume event data that it collects. The data is also eventually fed to Hadoop for offline, longer term analyses and batch reporting.

Redis is used for session-izing data in this organization. Every event collected from an application or website belongs to a session that a user starts against it.

Documents with tens or thousands of events create a stream of data that requires some amount of unraveling. When there are hundreds and thousands of users, event streams related to many users are interleaved. Updating each document with many small updates is difficult to accomplish with disk based databases, but Redis' HASH data structure makes short work of the whole problem.

HASHES can be used to store events by session with something like the below:

HSET session:1 event:1 data

And update it in micro-seconds with:

HINCRBY session: 1 seq 1

Keeping track of sessions that are timed out is also usually non trivial when there are thousands of sessions, but Redis has built in key expiration and timeout setting functionality that you can use to end sessions. Keyspace notifications allow you to subscribe to expired events and trigger an offload to MongoDB. (like below)

```
import redis

import pymongo

r = redis.Redis()

session = r.hgetall('session: 1')

#{'event:1': 'data', 'event2': 'data', 'seq':'2'...}

m= pymongo.MongoClient()

db = m.rta

sessionid = db.sessions.insert_one(session)
```

Another Redis Labs customer runs a navigation service quite like Waze or Google Maps, but with a focus on public transportation. For tens of millions of users, the service tracks public transit, using MongoDB for static information such as locations, vehicles, etc. However, the company feeds real-time reporting from apps on mobile devices, user reports of current traffic conditions, as well as GPS transponder reports into Redis, so users can know the state of live traffic and plan their routes on the fly.

Massively multiplayer online game

This gaming company caters to big teams that compete for resources and control, serving hundreds of team members and thousands of teams. While it uses MongoDB to store static information such as tournament profiles, resources, and resource profiles, Redis is used to store game progress, user scores and everything else that requires rapid fire updates and displays.

Redis sorted sets are useful in such scenarios – where the ZADD command adds users to the sorted set in the order of their score, and the ZINCRBY command simply increments scores by a specified number:

```
ZADD game:1 10000 id:1 21000 id:2

ZADD game:1 34000 id:3 35000 id:4

ZINCRBY game:1 10000 id:3
```

To retrieve the top scorer you would simply use:

```
ZREVRANGE game:1 0 0
```

Which will return

```
id:3
```

And to retrieve a user's rank you would use:

```
ZREVRANK game:1 id:4
```

Which will return sorted from high to low

```
1
```

(since the count begins from 0)

Online dating application

While there are many thousands of dating applications, this one caters to immediately finding groups of people by geographical proximity. MongoDB is used for storing user profiles, while the service uses Redis for tracking the location of users in real-time and instantly reconciling their preferences with those of others.

Redis commands like GEOADD can add geospatial information such as latitude and longitude to a specified key, and data is stored in a sorted set in a way that allows retrieval by proximity commands such as GEORADIUS or GEORADIUS by member.

Internet of Things application

An Internet of Things company provides environmental monitoring of sensors that are dropped into heavily forested areas to monitor real-time information such as temperature, humidity and CO2 levels in order to detect fire hazards for combating forest fires and other environmental problems. The sensor data, temperature maps and other variables needed for real-time calculations and alerting are stored in Redis, with longer term storage in MongoDB.

Conclusion

Redis, the #1 fastest growing database since 2013 and #1 database deployed in containers is increasingly used as part of two mega-trends: the move to multiple NoSQL databases optimized for particular use cases, and the move to increasingly real-time applications for thousands and millions of users.

MongoDB users have typically already made the move to NoSQL databases and find that Redis functionality and performance complements MongoDB for increasingly real-time use cases. Customer adoption of this duo is a testimonial to the rising power of NoSQL.