redislabs
home of redis

White Paper

Redis for Real-Time Analytics

Table of Contents

Executive Summary

Next generation applications need to embed analytics inline in order to generate intelligent and delightful customer experiences. Given the performance needs of interactive applications, this calls for high performance databases that are capable of handling a variety of application scenarios at the lowest complexity and costs, with uncompromising performance. Redis is such a database and a large chunk of Redis[e] users use it for real-time analytics. This whitepaper describes its analytic capabilities, fit for a variety of real-time scenarios along with a guide to handling vast amounts of data in Redis.

Business Drivers for High Performance Real-time Analytics

Next generation applications, at the cutting edge of user interaction, often tell the user, "We noticed you liked X. We think you will like Y." Or when the user is presented with choices on his next step or next purchase, they get a message "Many of our users seem happiest if they choose C with A. We thought you should know." Or sometimes, "You just booked Q – some reviews of R are here, you may want to check out R afterwards."

These types of savvy applications are presenting users with offers in CONTEXT of their other actions/decisions, when they are still warm, and ready to enhance their experience. Such offers are often much more powerful than those made through a cold email or a cold phone call several days later. This additional intelligence in applications also makes for a delightful user experience, a user who just booked tickets to a show at "The Second City in Chicago", might want to make a dinner reservation prior to the show and would be delighted to see discount offers or deals businesses close by. Lastly, this type of intelligence makes a ton of business sense, because competing businesses could easily grab loyal customers, were they to offer a better experience.

A measure of "smartness" of applications could also include detecting odd or unusual behavior that marks different types of fraud from gaming fraud, to identity fraud or abuse of promotions. This scenario is even more intense. To be effective, analytic processing of current behavior and detection of such fraud needs to complete before the user concludes their fraudulent activity. These types of applications could be using data from several different sources and comparing with current user behavior, to detect and prevent misuse.

With the increasing availability of cheap compute capacity, data processing tools and learning frameworks, it is becoming increasingly achievable to incorporate "intelligence" into applications. However, it is crucial to add this intelligence in without disrupting performance or responsiveness of applications. Customer facing applications in particular have to respond to users in under 100ms, to meet user experience expectations. With Internet latencies often at up to 50ms round trip, application processing, data access and responses have to be generated in 50ms. To achieve this at scale, the database used must be capable of delivering sub-millisecond response times under conditions of any load. Add in the need to achieve intelligence, and this means that your analytic operations on live data need to be accomplished at the same speeds and often as the same operation. In other words, in-database analytics become critical to meeting performance and functionality requirements.

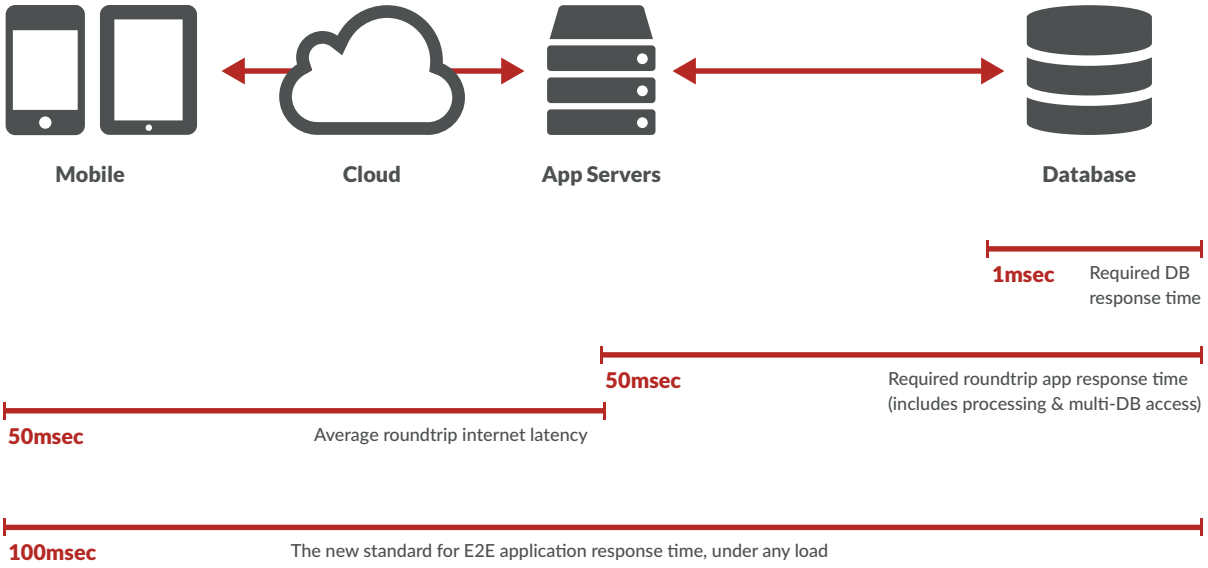

Figure 1. End-to-end application response time requirements

Analytical Capabilities of Redis

Redis is an in-memory database platform most well-known for its high performance, extreme versatility with data structures and modular extensibility to any data processing, analysis or storage use case.

When it comes to performance, Redis has been benchmarked to handle >1M operations/second at sub-millisecond latencies, with a single modest AWS instance. When it comes to end to end application throughput and latencies, Redis handily outperforms all other NoSQL databases in the market with the fewest resources.

But when it comes to analytics, the true performance boost is provided by its data structures which include Sets, Sorted Sets, Hashes, Lists, Strings, Bitmap and Hyperloglog. The data structures provide not just mechanisms to store variably structured data, they come with built-in operations that perform complex in-database analytics on the data, right in memory, where it is stored. This approach eliminates network and computing overhead while also radically simplifying application development complexity.

As an example, take the Sorted Set structure with Redis.

Sorted Sets order their members by a score. Retrieving members by score ranges is trivial with Sorted Set operations and make them a natural fit for time-series data, real-time bid management, purchases by order amounts, most viewed articles, top scores etc. Sorted Sets are built in Redis with mechanisms that provide high performance sorting, operations like ranking, range operations, counts in a range and also set operations like generating intersections, unions, are executed with the maximum possible efficiency and simplicity. Using Sorted Sets for analyses like time-series data analysis is usually an order of magnitude or two faster using Redis than with any other regular key/value store or with disk based databases.

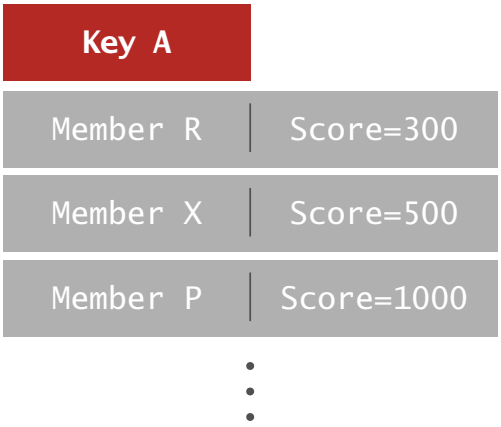

Figure 2. Sorted Set representation

Similar analytical operations come built into other data structures. Geo data structures include commands that analyze geospatial data and calculates distances, members within a particular distance from each other and more. Hyperloglog returns probabilistic cardinality estimates without having to store the actual items being added to a set – thereby maximizing memory space efficiency when there are millions of items.

These analytical operations exist side by side with data processing structures like Lists, Hashes, Sets, functionality like Publish/Subscribe and commands like key expiration, increment/decrement by values, pushing and popping from lists and many more, allowing developers to use different data structures like LEGO building blocks within their application.

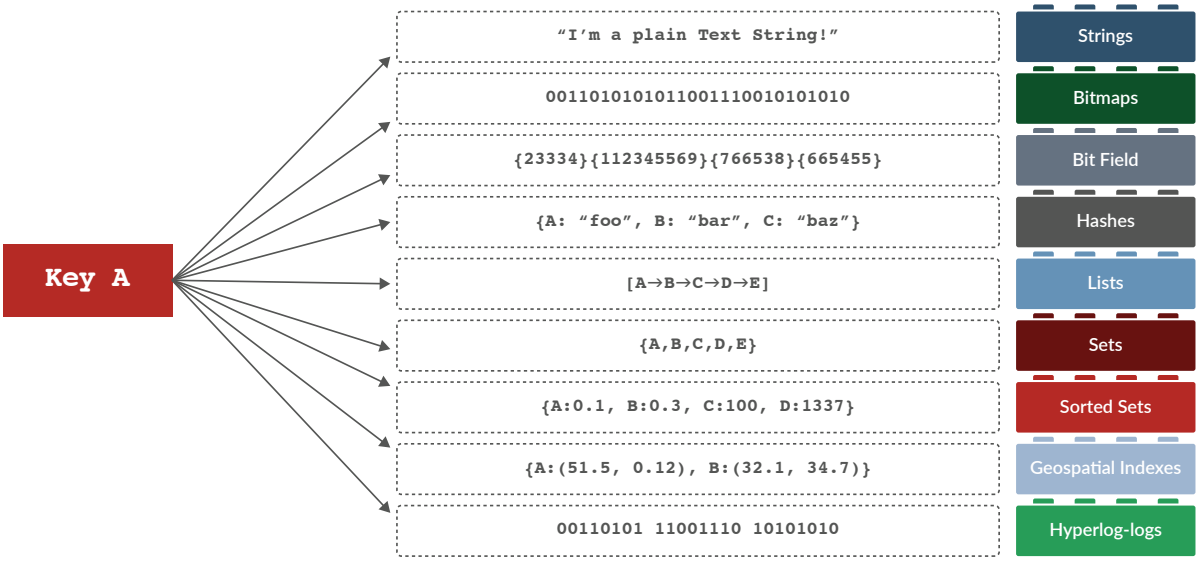

Figure 3. Redis data structures are like building blocks

Redis also supports an embedded scripting language, Lua, that extends the range of complex analytics you can execute on your data, with its simple syntax, convenient data constructs and capable base libraries.

Redis Modules further extend Redis capabilities when it comes to analytics. Recent modules include:

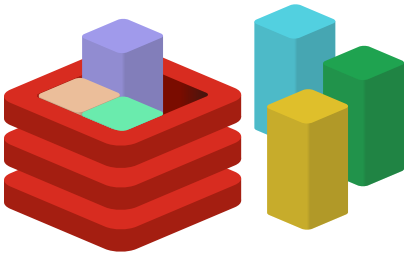

Figure 4. Redis Modules extend Redis infinitely

neural-redis: a simple feed forward neural network as a native data type in Redis

redis-ml: extends Redis to be a serving layer for machine learning models

redisearch: allows Redis to be used for full text indexing and search

rediJSON: JSON engine for Redis

Redis secondary index: allows creating and querying secondary indexes in Redis.

Redis graph: Redis module that implements a graph database.

topk: tracks the k most frequent elements in a stream

countminsketch: counts items approximately with the least memory usage

redablooms: implements scalable counting bloom filters

Redis Capabilities Applied to Transactional and Analytical Scenarios

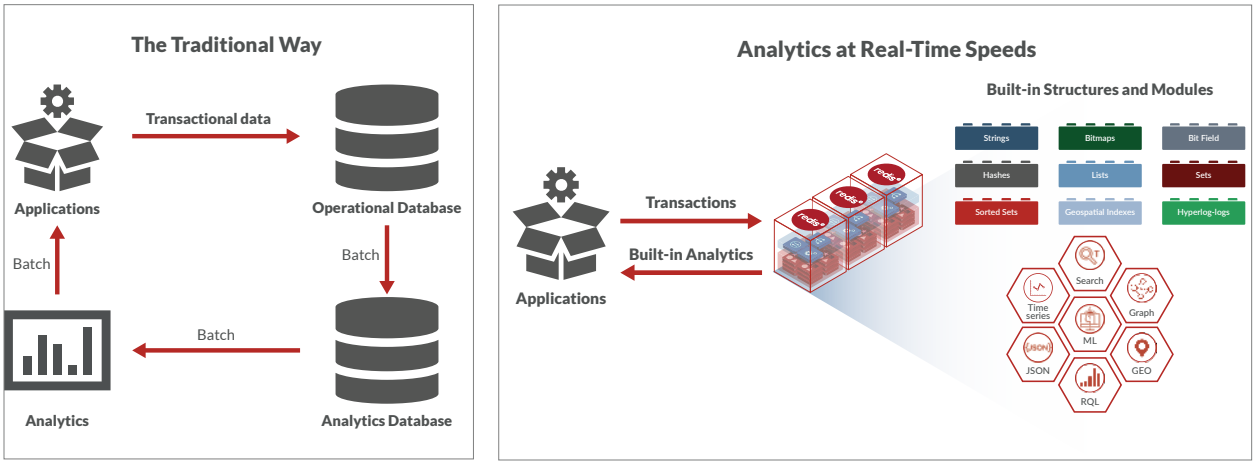

Figure 5. Modern day transactional and analytical scenarios

Given the high performance and scale enabled by Redis' analytical capabilities, smart applications have already started incorporating some of these as underpinnings to their key functionality.

Personalization/Recommendations

Personalization typically incorporates two dimensions:

- A user profile which could be comprised of their history, location, demographic, stated preferences
- Business rules related to offers being made to the user such as conversion targets, segmentation of offers, relevant behavioral context, reference sales

As a user interacts with the application, Redis is often used as a transactional store, providing high speed updates to user profiles while capturing the user's ongoing behavior. Concurrently it is also used to generate analytics such as counts, scoring, ranking etc that trigger the presentation of the

right offer. Additionally, its data structures like Sorted Sets can be used to process similarity scores at blazing fast speeds, compute and serve up the right recommendations with sub-millisecond latencies.

The Redis advantage in this scenario is that it can simultaneously update and analyze data in memory at blazing fast speeds, so "intelligence" is not a trade off with "performance".

A sample implementation of a Redis-based recommendations engine using Sorted Sets can be found on github: https://github.com/RedisLabs/redis-recommend.

Machine Learning For Predictions

When the ability to detect segments within users or similarities/dissimilarities in behavior requires complex algorithms that extract correlations on the fly, machine learning is commonly used to tease out connections. Open source frameworks like Apache Spark, Tensorflow, Torch, etc. are typically employed to run machine learning on large datasets to glean predictive algorithms. To date, incorporating machine learning based models has been a mostly offline or batch exercise. Frameworks like Apache Spark process large chunks of loosely related data to automatically detect patterns and generate models that serve as algorithmic methods to implement "intelligence".

These models are stored on disk and updated as a batch process. Often, they can only be retrieved, updated or executed from applications that are written in the same language because databases that store these models in their native format don't exist. But more importantly, complex models cannot be served fast enough at scale by standard ML platforms. If a simplified model is used, the results are typically not accurate enough. And complex models are usually too large to be accommodated at the application layer. Serving these fast enough with homegrown methods in under 50 msec requires too many servers.

Redis-ML, the machine learning module from Redis Labs allows storage of machine learning models such as decision trees generated by random forest algorithms, in native format, in-memory. This allows these models to be stored, retrieved, executed and updated in memory at the scale and speed required for inline processing, with very few resources. Redis-ML is, like Redis, written in C, and delivers ML serving with the least overhead and resource usage. It inherits the high availability and scale of Redis[e], and executes complex operations like replacing a decision tree in a random forest with simplicity and high performance. Added advantages include the ability to handle multiple different types of machine learning models such as Random forest, Logistic Regression, Gradient Boosted Trees and allows their simultaneous use by applications written in different programming languages.

For machine learning scenarios that do not require processing of visual data, and require simple feed forward neural networks, Neural-Redis, an open source Redis module enables implementation of such neural networks in memory.

The advantages of such built-in machine learning capabilities are enormous. As predictive intelligence gets implemented in production, the accuracy of machine learning models and the capability to keep them up to date makes the difference between disgust and delight. With the powerful performance of Redis and its ability to natively serve and update machine learning models, applications can handle hundreds and thousands of instantaneous decisions rapidly, and with accuracy.

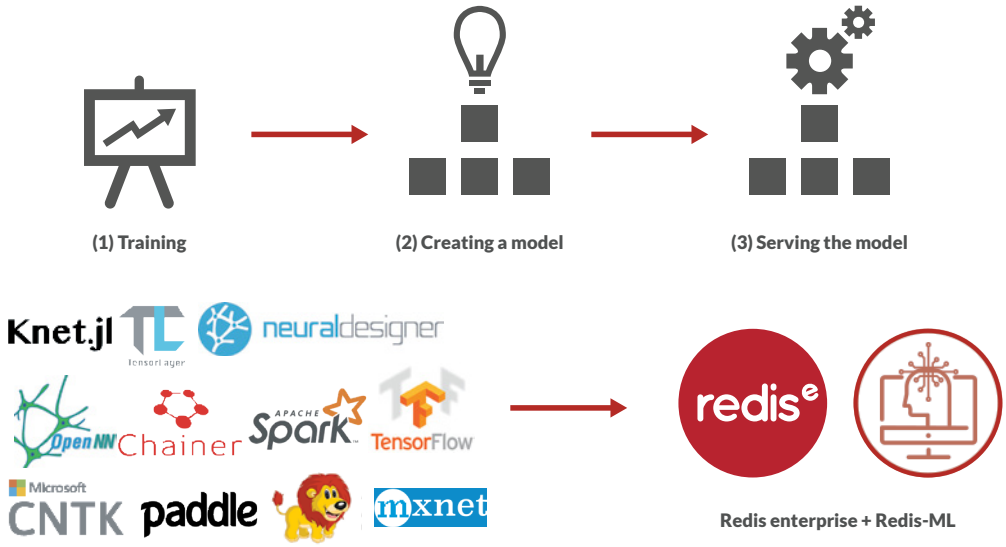

Figure 6. Typical machine learning lifecycle

Fraud Detection

Fraud detection often relies on statistical techniques or artificial intelligence mechanisms, as well as human intervention, while combing through vast volumes of data and transactions. Redis usage in fraud detection scenarios is most often fueled by its capability to handle immense volumes of data at very low latencies.

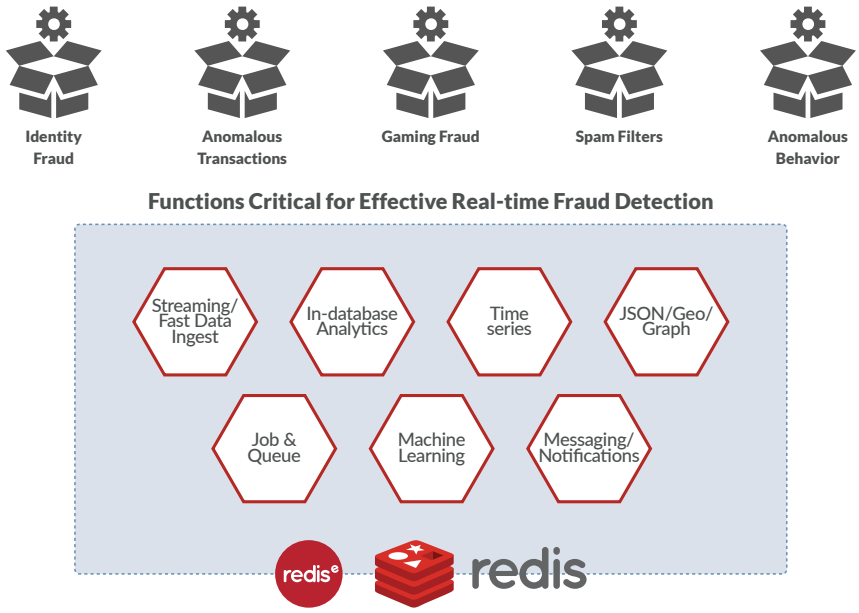

Figure 7. Redis[e] capabilities make it the most efficient database for implementing fraud detection

Fraud detection systems typically need to consume tens of millions of data points and are often inline to many customer transactions to be most effective. Redis' high performance coupled with

its ability to handle streaming data through pub/sub functionality and data structures such as lists make it the best choice for fast data ingest.

Built in analytics such as set cardinality and Hyperloglog based counters make it an efficient way to implement fraud estimation functionality. Native data structures such as Geo and Sorted Sets as well as modules that handle JSON, Graph and other data help accelerate analysis based on location, time, relationships, and other parameters.

Redis' ability to handle high speed transactions (reads consistent with writes) as well as the use of its List data structure for job & queue management and its Hash data structure for lightning fast updates to user session data, make it a versatile choice for powering in-line fraud detection, where application latencies need to be in the 100ms range, requiring data access and processing to need the sub-millisecond response times of Redis. Built-in job & queue and messaging functionality help trigger alerts or notifications when fraudulent transactions are detected.

Machine learning (as noted in the above section) can be implemented simultaneously – giving Redis an enormous advantage over other databases where numerous functions need to be implemented either at the application level or a variety of specialty databases need to be deployed to handle separate needs of the application.

> *"Using Redis[e] Pack in our fraud detection service was an excellent decision for our organization. It is enabling us to easily manage billions of transactions per day, keep pace with our exponential growth rate, and speed fraud detection for all of our clients."* **- Ravi Sandepudi Head of Engineering, Simility.**

Interactive Reporting

When users are creating meaningful reports over millions of data points interactively, response time expectations for data pulls are in the <10ms range and pagination times are expected to be in <3ms. Most large data repositories are unable to handle such low latency requirements, and often use Redis as an intermediary data store. Redis Sorted Sets, with their optimized pre-sorting are capable of meeting extremely low latency requirements even with millions of records to report from.

Redis is often used in front of RDBMS as well as large disk based NoSQL repositories like HBase, Cassandra etc to provide the high throughput and low latency needed by users.

How to Manage Terabytes of Data in Redis

Redis runs in memory, and two questions usually come up in relation to large volumes of data. The first is about how to scale Redis and the second about how manage enormous volumes of data with high performance but low costs.

Scaling Redis with High Availability

Redis is single threaded and can use only one processing core at a time. It can be scaled in memory by creating additional instances or shards. Redis can be used in persistent mode as a full-fledged database, with additional availability options of in-memory replication across racks/zones/datacenters/regions and even clouds.

With Redis Labs' downloadable software Redis[e] Pack, you can create a shared pool of memory across group of servers in an automated fashion, so that your Redis datasets can grow seamlessly beyond the largest memory available on a single server. Redis Labs also provides Redis[e] Cloud, which runs on all the different IaaS public clouds including AWS, Azure, GCP, IBM Softlayer as well as PaaS clouds like Heroku, OpenShift, CloudFoundry. You interact with Redis as if it is a single instance, while Redis Labs takes care of the provisioning, scaling, sharding, re-sharding and re-balancing operations.

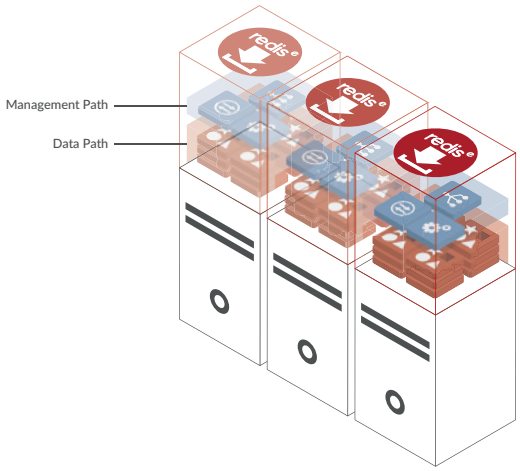

Figure 8. Shared nothing dynamic cluster architecture of Redis[e] Pack

Built into Redis[e] is a full high availability suite including persistence, in-memory diskless replication across racks, zones, datacenters, regions and even clouds. Redis instances are continuously monitored, triggering failover within instants, avoiding the possibility of data loss. Redis Labs' has been benchmarked to provide the most robust high availability for Redis deployments.

Redis[e] includes a distributed cut-through Proxy, a shared nothing cluster architecture, with watchdogs at the node and cluster level that are constantly monitoring and optimizing Redis deployments. Numerous performance optimizations ensure a stable, predictable performance benchmarked as the fastest.

Optimizing Costs while delivering High Performance

Generational shifts in Flash memory have extended performance without a corresponding shift in price. Redis[e] Flash takes advantage of this high performance alternative to offer near RAM like performance with up to 70% lower costs. Flash is treated as an extension of RAM (and not used for persistent storage). Asynchronous and multi-threaded access to Flash assures optimal performance. But the real performance benefit is achieved by taking a tiered approach to data – keys and hot values are stored in RAM and cold values are stored in Flash memory. This enables sub-millisecond latencies, even with extremely high throughput using a combination of Flash and RAM. Recent benchmarks demonstrate Redis on Flash performance of up to 3M ops/second, with <1 ms latency using a combination of RAM and Flash ratio of 10:90.

Analytical scenarios with Redis[e] Flash include genome data analysis at a prominent university where Redis accomplished the much needed analysis, at blazing fast performance even with 30TB of raw data.

	Redis on RAM	Redis^{e} on Flash
Datset Size	10TB	10TB
Databse size with replication	30TB	20TB*
AWS instance type	x1.32xlarge	i3.16xlarge
Actual instance size (RAM, and RAM+Flash)	1.46TB	3.66TB
# of instances needed	21	6
Persistent Storage (EBS)	154TB	110TB
1 year cost (reserved instances)	$1,595,643	$298,896
Savings	-	81.27%

* Redis enterprise only needs 1 copy of the data because quorum issues are solved at the node level

Figure 9. A cost comparison of Redis^{e} on Flash vs Redis on RAM, using AWS instances

Conclusion

As the world moves towards analytics at the speed of business, enabling technologies such as Redis are being increasingly adopted to increase the effectiveness of "intelligence" in applications. Implementing these analytics while simultaneously handling operational tasks requires the robust high performance and low latencies of in-memory databases, while delivering simultaneously analytics requires built-in analytic operations in Redis data structures and versatile data processing capabilities of Redis modules. Redis capabilities provide both a tremendous boost to performance and a reduction in application complexity. Redis Modules extend Redis to numerous analytic scenarios while Redis^{e} Flash makes it possible to reduce operational costs by up to 70%. For additional information about how you can deploy Redis in your analytic scenarios, email expert@redislabs.com.