BENCH MARK

Redis Enterprise Demonstrates True Linear Scaling of Database Performance

50 Million operations/second, <1 millisecond latency with 26 EC2 nodes

David Maier, Redis

CONTENTS

Executive Summary

Redis, the world's most popular in-memory NoSQL database, is used by developers worldwide for its powerfully efficient operational and analytical data processing. Redis typically runs in RAM and delivers extremely high performance; a single Redis instance can execute hundreds of thousands operations per second with sub-millisecond latency. Redis Enterprise enhances this experience with additional features that deliver higher performance and availability, local latencies on global deployments, zero data loss and seamless scaling.

Linear scaling of database performance is critical for applications to scale easily and cost-efficiently. Many cloud or on-premises databases claim to scale linearly but can rarely prove it in the manner that Redis Enterprise has demonstrated.

Redis Enterprise 5.0 introduced support for the Open Source (OSS) cluster API. The API allows a Redis Enterprise cluster to *scale infinitely and in a linear manner by simply adding shards and nodes*. The OSS cluster API allows Redis clients to directly access the shard that holds the key/value object with no additional network hop.

This, combined, with the shared nothing symmetric architecture of Redis Enterprise cluster ensures that data and control path are separate and the control path does not impose non-linear overheads in a scaled out environment.

Redis Enterprise has set a new industry performance record: **delivering over 50 million ops/second under 1 millisecond, in as little as 26 EC2 nodes.**

This paper describes the configuration, tools, workloads, and results of the benchmark.

Introduction to a Redis Enterprise Cluster

A Redis Enterprise cluster is composed of identical nodes that are deployed within a data center or stretched across local availability zones. Redis Enterprise architecture is made up of a management path and data access path. The cluster has a symmetric shared-nothing architecture - complete separation between the data-path and control & management-path, and it includes the following main components:

- Redis Shards (data-path) – a Redis instance with either a master or slave role
- Zero-latency Proxy (data-path) – built on a multi-threaded, stateless, cut-through architecture and is responsible for hiding cluster complexity, enhancing security (SSL, authentication, DDoS protection) and improving performance (TCP-less, connection management and pipelining)
- Cluster Manager (control & management-path) – built from a set of distributed processes that reside on each node of the cluster. The cluster manager takes care of cluster configuration activities, provisioning requests, resource management and monitoring, as well as acting as a resource watchdog and completely offloading the task of having Redis shards manage the health of other shards in the cluster or make failover decisions.

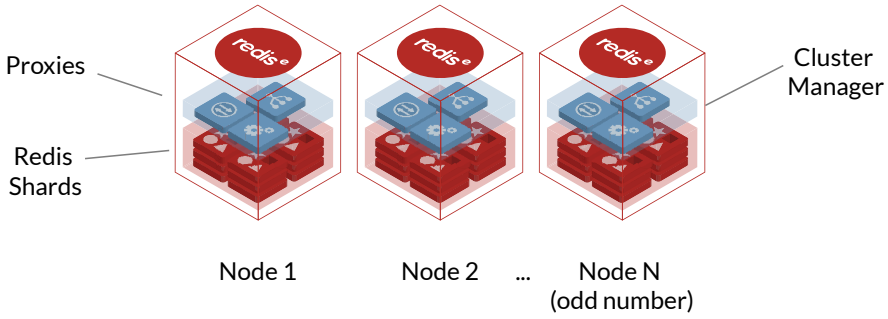


Figure 1: Redis Enterprise cluster architecture

A database in the Redis Enterprise cluster can be created in any one of the configurations below:

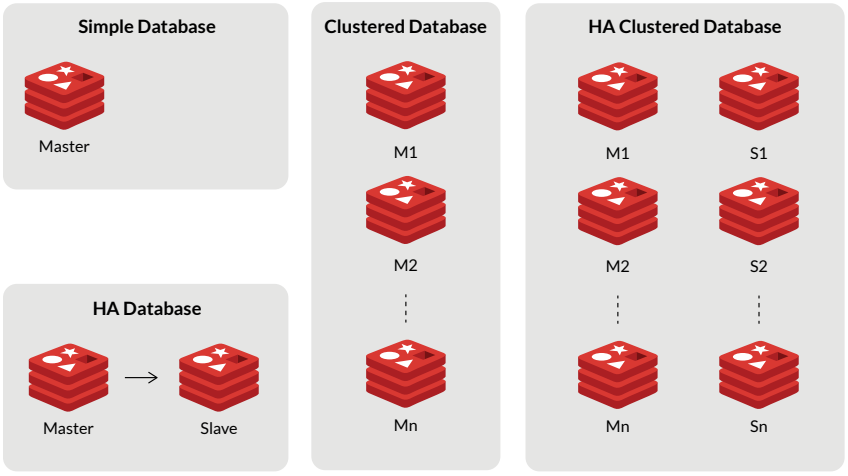


Figure 2: Redis Enterprise cluster configuration options

Based on its strong multi-tenant technology, a Redis Enterprise cluster can manage multiple databases of different types on the same cluster resources in a completely isolated manner.

Redis Enterprise with OSS cluster API

In order to utilize the new OSS cluster API, the following properties were used to create a database with the Redis Enterprise API:

- "shards_placement": "sparse" – distributes shards across Redis Enterprise cluster nodes
- "proxy_policy": "all-master-shards" – assigns a proxy on every node that has at least one master shard
- "oss_cluster": "true" – make the database accessible through the OSS cluster API

This will create a clustered database with properties similar to those shown below:

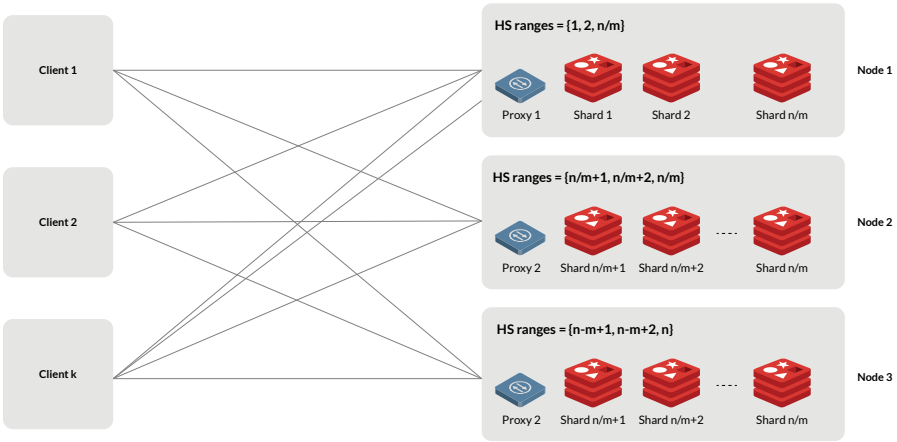


Figure 3: Redis Enterprise running a clustered database with OSS cluster API

Each proxy is responsible for a range of hash-slots (HSs) based on the shards that are running on the same cluster node. The client (which leverages the OSS cluster API) sends requests to the relevant proxy based on the key part of a key/value pair and the Redis cluster hashing function. Then the proxy uses the Unix Socket Domain connection to forward the requests to the relevant shard on the node by applying the same hashing function. In the case of a cluster topology change, e.g. shard migration, master/slave failover, etc., the proxy updates the hash slots it manages and if necessary redirects requests from

the clients to another node in the cluster (using MOVED reply).

Benchmarking Methodology and Configurations

Methodology

We used memtier_benchmark, an open-source load generation tool created by Redis. Using multiple instances of memtier_benchmark is essential for generating the required amount of traffic and avoiding network bandwidth and packet per second limitations.

Hardware and Software Setup

Hardware:

Given the expected large load for testing, and to demonstrate true linear scaling, this benchmark was performed on 2 different hardware configurations for different and extremely large throughputs.

Client machine and nodes configuration to handle 30M operations/second:

Role	Number of instances	EC2 instance type
Cluster node	18	m4.16xlarge (64 CPU cores, 256GB RAM)
Load generator	11	c4.8xlarge (36 CPU cores, 60 GB RAM)

Client machine and nodes configuration to handle 50M operations/second:

Role	Number of instances	EC2 instance type
Cluster node	26	m4.16xlarge (64 CPU cores, 256GB RAM)
Load generator	17	c4.8xlarge (36 CPU cores, 60 GB RAM)

Software versions:

Redis Enterprise v5.0

Memtier_benchmark v1.2.11

We used the Ubuntu Server 16.04 as our operating system and ensured that the machines were set up to support Enhanced Networking. All instances were placed in the same VPC, availability zone, network subnet and placement group.

Creating and Tuning a Clustered Database

To support a load of 30 million operations/second, we created a 360 shard database over an 18 node cluster.

For creating the database, we used the Redis Enterprise API with the following parameters:

```
{
      "name": "api-benchmark",
      "replication": false,
```

```
      "port" : 12345,
      "sharding": true,
      "shards_count" : 360,
      "version": "4.0",
      "memory_size": 100000000000,
      "type": "redis",
      "oss_cluster":true,
      "proxy_policy": "all-master-shards",
      "shards_placement": "sparse",
      "shard_key_regex": [{"regex": ".*\\{(?<tag>.*)\\}.*"}, {"regex": "(?<tag>.*)"}]
}
```

For a load of 50M operations/second, we created a 512 shard database over a 26 node cluster with the following parameters:

```
{
"name": "api-benchmark",
"replication": false,
"port" : 12345,
"sharding": true,
"shards_count" : 512,
"version": "4.0",
"memory_size": 100000000000,
"type": "redis",
"oss_cluster":true,
"proxy_policy": "all-master-shards",
"shards_placement": "sparse",
"shard_key_regex": [{"regex": ".*\\{(?<tag>.*)\\}.*"}, {"regex": "(?<tag>.*)"}]
}
```

Each shard is designed to support 83.333K ops/second and each node 1.666M ops/second.

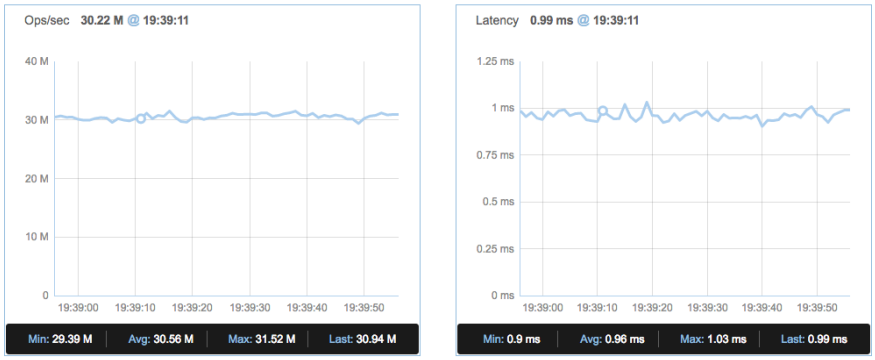
```
> curl -k -X POST -u "<user>:<password>" -H "Content-Type:
application/json" -d @create_db.json https://localhost:9443/v1/bdbs
```

We use thread proxy configuration for each cluster node.

```
> rladmin tune proxy all max_threads 24
> rladmin tune proxy all threads 24
```

Results Observed

The benchmark executed on a 360-shard database over an 18-node Redis Enterprise cluster, demonstrated the following results: over ***30 million ops/second at 0.99 millisecond latency.***



A 512 shard database over a 26-node Redis Enterprise cluster demonstrated similar outstanding results: over 51M operations/second at 1.1 millisecond latency.

The tests included additional benchmarking: n-shard database on a k-node Redis Enterprise cluster as noted below:

Number of shards	Number of nodes
60	3
120	6
240	12
360	18
480	24
512	26

We found linear performance scalability when scaling from a 3-node cluster to an 26-node cluster, as shown in the following graphs

6 nodes, 120 shards cluster | 12 nodes, 240 shards cluster | 18 nodes, 360 shards cluster | 26 nodes, 512 shards cluster

A deeper analysis of these results indicates that when scaling out, the maximum throughput variance per node is no more than about 13%m, likely related to different cloud resource conditions (network, VM, etc.) on each test iteration.

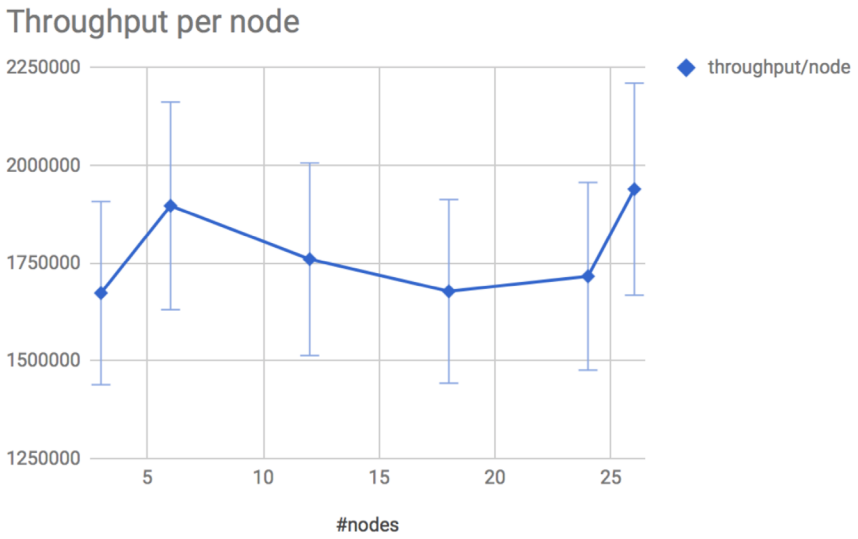


Figure 6: Throughput per node

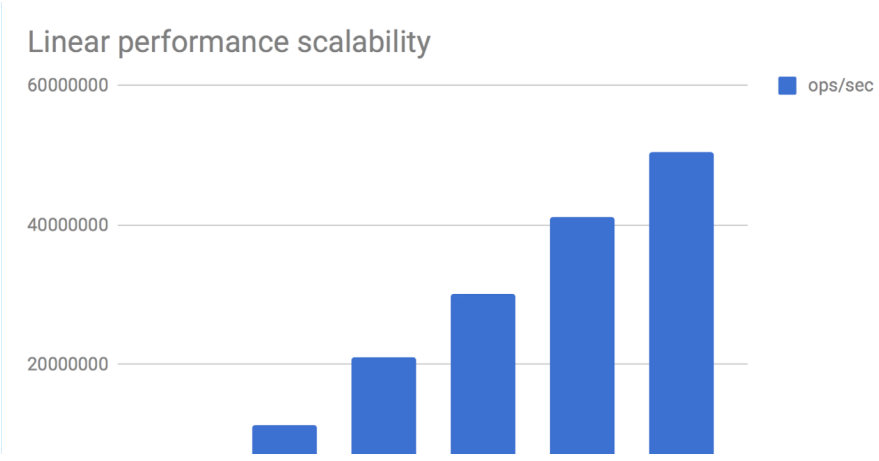


Figure 7: Linear performance scalability

Conclusion

The results of the benchmark clearly illustrate that Redis Enterprise outperforms any other database in the market, in its efficient use of resources for high throughput and low latency scenarios. By employing the OSS cluster API, Redis Enterprise 5.0 can easily scale linearly by just adding shards to the database and adding nodes to the cluster for extremely large workloads. ***This benchmark sets a new industry record of 50M operations per second at less than 1 millisecond latency, with only 26 cluster nodes.***

It is critically important that modern database scales predictably, perform seamlessly at any scale and reduce application development complexity. This new industry leading benchmark demonstrates the linear, non-disruptive scalability, superior and predictable performance at a small hardware footprint and cost, making Redis Enterprise the most versatile and powerful database in the industry.

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