Benchmark a Redis on Flash enabled database

Redis on Flash (RoF) on Redis Enterprise Software lets you use cost-effective Flash memory as a RAM extension for your database.

But what does the performance look like as compared to a memory-only database, one stored solely in RAM?

These scenarios use the `memtier_benchmark` utility to evaluate the performance of a Redis Enterprise Software deployment, including the trial version.

The `memtier_benchmark` utility is located in `/opt/redislabs/bin/` of Redis Enterprise Software deployments. To test performance for cloud provider deployments, see the `memtier-benchmark GitHub project`.

For additional, such as assistance with larger clusters, contact support.

Benchmark and performance test considerations

These tests assume you’re using a trial version of Redis Enterprise Software and want to test the performance of a Redis on Flash enabled database in the following scenarios:

- Without replication: Four (4) master shards
- With replication: Two (2) primary and two replica shards

With the trial version of Redis Enterprise Software you can create a cluster of up to four shards using a combination of database configurations, including:

- Four databases, each with a single master shard
- Two highly available databases with replication enabled (each database has one master shard and one replica shard)
- One non-replicated clustered database with four master shards
- One highly available and clustered database with two master shards and two replica shards

Test environment and cluster setup

For the test environment, you need to:

1. Create a cluster with three nodes.
2. Prepare the flash memory.
3. Configure the load generation tool.

Creating a three-node cluster

This performance test requires a three-node cluster.

You can run all of these tests on Amazon AWS with these hosts:

- 2 x i3.2xlarge (8 vCPU, 61 GiB RAM, up to 10Gbit, 1.9TB NMVe SSD)
These nodes serve RoF data

- 1 x m4.large, which acts as a quorum node

For instructions on how to install RS and set up a cluster, go to either:

- **Quick Setup** for a test installation
- **Install and Upgrade** for a production installation

These tests use a quorum node to reduce AWS EC2 instance use while maintaining the three nodes required to support a quorum node in case of node failure. Quorum nodes can be on less powerful instances because they do not have shards or support traffic.

As of this writing, i3.2xlarge instances are required because they support NVMe SSDs, which are required to support RoF. Redis on Flash requires Flash-enabled storage, such as NVMe SSDs.

For best results, compare performance of a Flash-enabled deployment to the performance in a RAM-only environment, such as a strictly on-premises deployment.

**Prepare the flash memory**

After you install RS on the nodes, the flash memory attached to the i3.2xlarge instances must be prepared and formatted with the `/opt/redislabs/sbin/prepare_flash.sh` script.

**Set up the load generation tool**

The `memtier_benchmark` load generator tool generates the load on the RoF databases. To use this tool, install RS on a dedicated instance that is not part of the RS cluster but is in the same region/zone/subnet of your cluster. We recommend that you use a relatively powerful instance to avoid bottlenecks at the load generation tool itself.

For these tests, the load generation host uses a c4.8xlarge instance type.

**Database configuration parameters**

**Create a Redis on Flash test database**

You can use the RS admin console to create a test database. We recommend that you use a separate database for each test case with these requirements:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>With replication</th>
<th>Without replication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>test-1</td>
<td>test-2</td>
<td>The name of the test database</td>
</tr>
<tr>
<td>Memory limit</td>
<td>100 GB</td>
<td>100 GB</td>
<td>The memory limit refers to RAM+Flash, aggregated across all the shards of the database, including master and replica shards. RoF always keeps the Redis keys and Redis dictionary in RAM and additional RAM is required for storing hot values. For the purpose of these tests 30% RAM was calculated as an optimal value.</td>
</tr>
<tr>
<td>RAM limit</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>
Replication

A database with no replication has only master shards. A database with replication has master and replica shards.

Data persistence

No data persistence is needed for these tests.

Database clustering

A clustered database consists of multiple shards.

Number of (master) shards

- With replication: One master shard and one replica shard on each node
- Without replication: Two master shards on each node

Other parameters

Keep the default values for the other configuration parameters.

Data population

Populate the benchmark dataset

The memtier_benchmark load generation tool populates the database. To populate the database with N items of 500 Bytes each in size, on the load generation instance run:

```bash
$ memtier_benchmark -s $DB_HOST -p $DB_PORT --hide-histogram --key-maximum=$N -n allkeys -d 500 --key-pattern=P:P --ratio=1:0
```

Set up a test database:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>With replication</th>
<th>Without replication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication</td>
<td>Enabled</td>
<td>Disabled</td>
<td>A database with no replication has only master shards. A database with replication has master and replica shards.</td>
</tr>
<tr>
<td>Data persistence</td>
<td>None</td>
<td>None</td>
<td>No data persistence is needed for these tests.</td>
</tr>
<tr>
<td>Database clustering</td>
<td>Enabled</td>
<td>Enabled</td>
<td>A clustered database consists of multiple shards.</td>
</tr>
<tr>
<td>Number of (master) shards</td>
<td>2</td>
<td>4</td>
<td>Shards are distributed as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- With replication: One master shard and one replica shard on each node</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Without replication: Two master shards on each node</td>
</tr>
<tr>
<td>Other parameters</td>
<td>Default</td>
<td>Default</td>
<td>Keep the default values for the other configuration parameters.</td>
</tr>
</tbody>
</table>

Centralize the keyspace

With replication

To create roughly 20.5 million items in RAM for your highly available clustered database with 75 million items, run:
$ memtier_benchmark  -s $DB_HOST -p $DB_PORT --hide-histogram --key-minimum=2725000 --key-maximum=47750000 -n allkeys --key-pattern=P:P --ratio=0:1

To verify the database values, use Values in RAM metric, which is available from the Metrics tab of your database in the admin console.

Without replication

To create 41 million items in RAM without replication enabled and 150 million items, run:

$ memtier_benchmark  -s $DB_HOST -p $DB_PORT --hide-histogram --key-minimum=54500000 --key-maximum=95500000 -n allkeys --key-pattern=P:P --ratio=0:1

Test runs

Generate load

With replication

We recommend that you do a dry run and double check the RAM Hit Ratio on the metrics screen in the RS admin console before you write down the test results.

To test RoF with an 85% RAM Hit Ratio, run:

$ memtier_benchmark -s $DB_HOST -p $DB_PORT --pipeline=11 -c 20 -t 1 -d 500 --key-maximum=75000000 --key-pattern=G:G --key-stddev=5125000 --ratio=1:1 --distinct-client-seed --randomize --test-time=600 --run-count=1 --out-file=test.out

Without replication

Here is the command for 150 million items:

$ memtier_benchmark -s $DB_HOST -p $DB_PORT --pipeline=24 -c 20 -t 1 -d 500 --key-maximum=150000000 --key-pattern=G:G --key-stddev=1025000 --ratio=1:1 --distinct-client-seed --randomize --test-time=600 --run-count=1 --out-file=test.out

Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access pattern (--key-pattern) and standard deviation (--key-stddev)</td>
<td>Controls the RAM Hit ratio after the centralization process is complete. Controls how many connections are opened to the database, whereby the number of connections is the number of threads multiplied by the number of connections per thread (-t) and number of clients per thread (-c)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Pipelining (--pipeline)</td>
<td>Pipelining allows you to send multiple requests without waiting for each individual response (-t) and number of clients per thread (-c)</td>
</tr>
<tr>
<td>Read\write ratio (--ratio)</td>
<td>A value of 1:1 means that you have the same number of write operations as read operations (-t) and number of clients per thread (-c)</td>
</tr>
</tbody>
</table>

**Test results**

**Monitor the test results**

You can either monitor the results in the **metrics** tab of the admin console or with the memtier_benchmark output. However, be aware that:

- The memtier_benchmark results include the network latency between the load generator instance and the cluster instances.
- The metrics shown in the admin console do not include network latency.

**Expected results**

You should expect to see an average throughput of:

- Around 160,000 ops/sec when testing without replication (i.e. Four master shards)
- Around 115,000 ops/sec when testing with enabled replication (i.e. Four master and 2 replica shards)

In both cases, the average latency should be below one millisecond.

**Updated:** September 24, 2021