Benchmarking MongoDB on Kubernetes with Percona Operator

The benchmark test comparing Percona Server for MongoDB 6.0 running on bare metal servers to Percona Operator for MongoDB in a Kubernetes environment revealed no performance penalties when operating in Kubernetes. Using a range of hardware configurations and a comprehensive set of tests, including mongo-perf benchmarks, the results consistently indicated equivalent performance levels across both environments. This demonstrates that Kubernetes, with the help of Percona Operator for MongoDB, is capable of hosting cloud-native databases without compromising on efficiency or speed.

Test stand

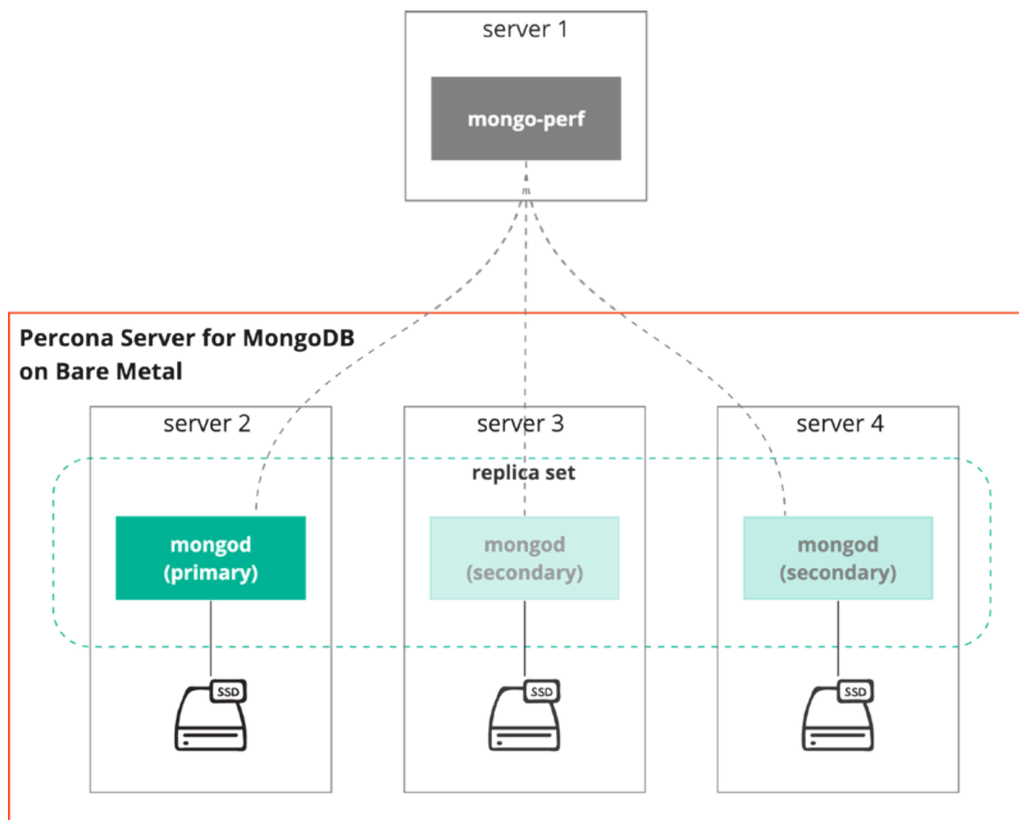
Seven servers. Each server has the following hardware:

- CPU: Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz - 80 cores
- RAM: 192 GB DDR4
- Disk: 3.2 TB NVMe SSD

Servers are connected to a 10GB network with each other through a top-of-the-rack switch.

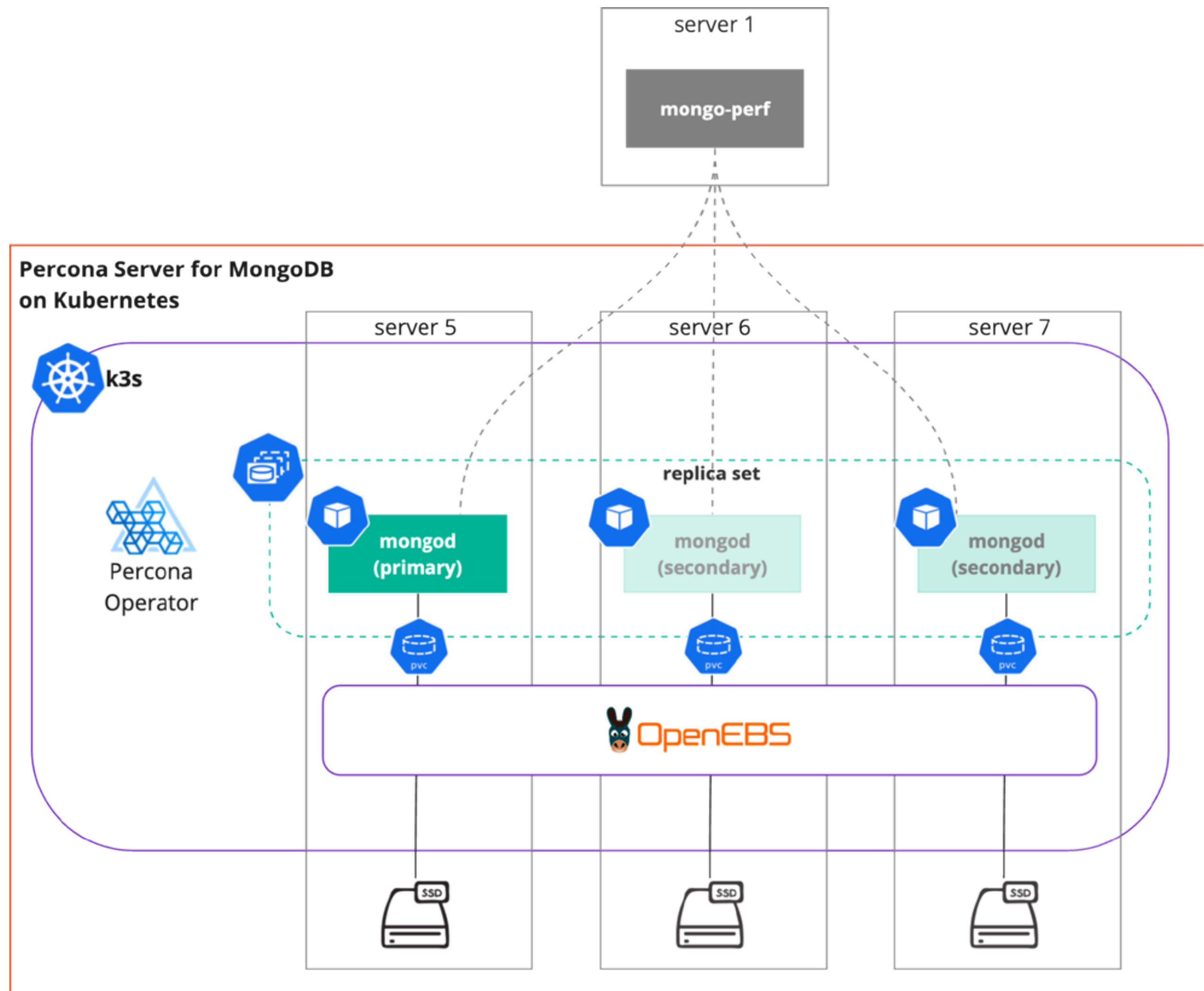
Topology

Percona Server for MongoDB on bare metal servers topology:



© Copyright 2024 Percona® LLC. All rights reserved.

Percona Server for MongoDB on Kubernetes topology, deployed with Percona Operator for MongoDB:



Configuration

Operating system

1. Set proper TCP KeepAlive: net.ipv4.tcp_keepalive_time=300.
2. Set noatime to a storage volume on the OS level.
3. Set ulimits to 64000.
4. Use XFS as a filesystem

MongoDB

As MongoDB wiredTiger storage engine uses both the WT internal cache and the file system cache, we set wiredTiger internal cache to 25% for small databases and 50% for larger ones (depends on tests).

Kubernetes

We deployed Kubernetes with k3s, keeping the default flannel Container Network Interface (CNI). For storage, we decided to use **OpenEBS** to get access to local NVMe SSDs through Persistent Volume Claims. We used openebs-hostpath storage class, which is available in the default installation.

Operator and Custom Resource Definitions (CRDs) are deployed as is from our default bundle.yaml manifest.

© Copyright 2024 Percona® LLC. All rights reserved.

What was changed from defaults:

1. Disable sharding
2. Expose replica set
3. Add corresponding configuration for MongoDB (see above).

Benchmarking tool

We utilized a benchmarking tool specific to MongoDB called **mongo-perf**.

The testing methodology itself is mirrored after multi-version testing completed by Yicheng Zhou. That blog can be found **here**.

mongo-perf uses Python-based scripts for specific test runs that target and address all aspects of performance, including looking at the aggregation pipeline. These types of activities can be very time- and resource-consuming. There are approximately 35 test case scripts **available**.

From those, we selected the eight master scripts listed below to run against all environments.

- simple_insert.js
- simple_update.js
- simple_query.js
- pipeline_update.js
- complex_insert.js
- complex_update.js
- partial_index.js
- simple_remove.js

We ran each test six times and filtered out two outliers. For the four remaining, we calculated the average that is presented to you.

Results

Below we share with you the most important metrics, but you can find all the details from our tests in this detailed blog post/github repo/etc (tbd). All tests measure operations per second (Ops).

Simple queries

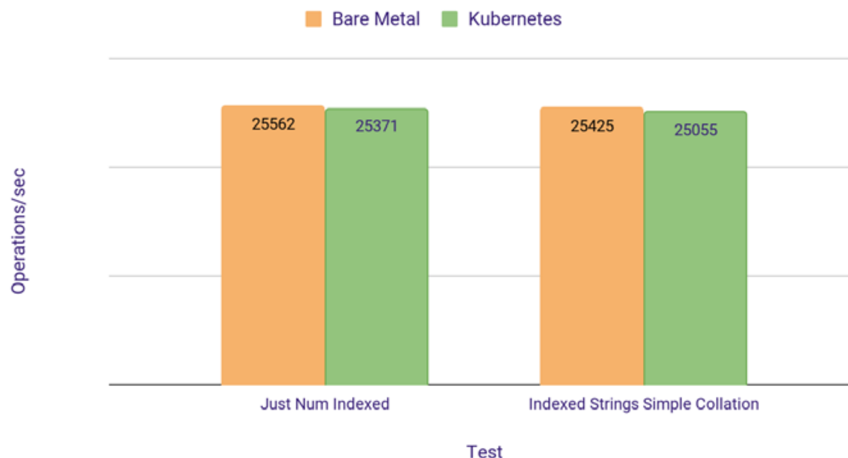
- Int ID Range - query for all documents with integer _id in the range (50,100). All threads are returning the same documents.

PERCONA
© Copyright 2024 Percona® LLC. All rights reserved.

- Int Non ID Range - query for all documents with x in range (50,100). All threads are returning the same documents and use index on x.
- String Range With Simple Collation - query for a range of strings using the simple collation.
- Int NonId Find Oe Projection Covered - query for random document based on integer field x, and use projection to return only the field x. Each thread accesses a distinct range of documents. Query should be a covered index query.

Kubernetes slightly outperforms (for example, 3.97% on Int ID Range) bare metal in simple queries. But this difference is negligible and will even out with a larger number of tests.

Simple Insert

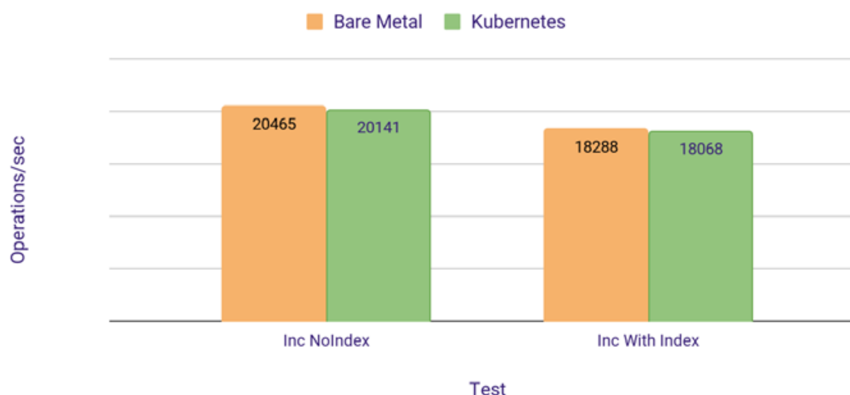


Here are two tests. Both tests measure the operations per second.

- Just Num Indexed - insert documents `{ _id : OID, x : NumberInt}` into a collection where x is indexed.
- Indexed Strings Simple Collation - repeatedly insert an indexed 10-character string.

For both cases, we see slight (-0.75% and -1.46% correspondingly) slower processing in Kubernetes.

Simple update

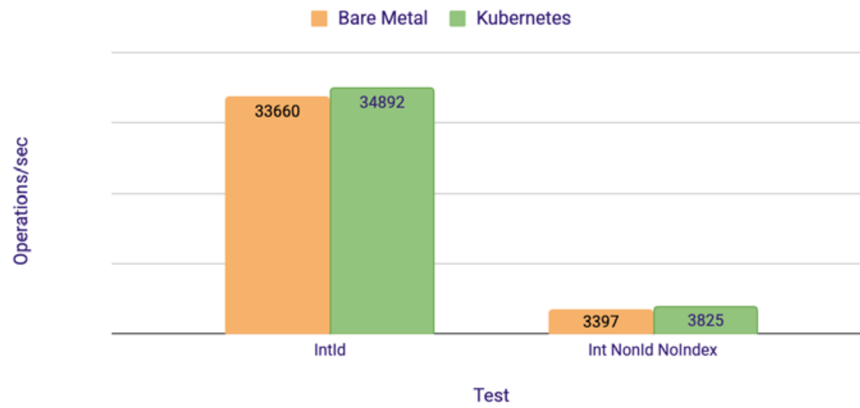


- Inc NoIndex - select a document based on the integer _id field and increment a field X.
- Inc With Index - same as previous test, but X is indexed.

© Copyright 2024 Percona® LLC. All rights reserved.

Similar to previous tests, we see a negligible performance difference within 1.5%.

Simple remove



- IntId - each thread works in a range of 100 documents; remove (and re-insert) a random document in its range using the _id field.
- Int NoId NoIndex - same as previous, but uses the int field instead of the _id for the query condition.

Learn more about Percona Kubernetes Operators and how to get started

© Copyright 2024 Percona® LLC. All rights reserved.