YugaByte DB
Enterprise Evaluation Guide

A YUGABYTE TECHNICAL PAPER

Sid Choudhury
VP Product, YugaByte

February 2019

Table of Contents

Executive Summary

Built using a unique combination of high-performance storage engine, auto sharding, per-shard distributed consensus replication and multi-shard ACID transactions (inspired by Google Spanner), YugaByte DB is world's only distributed database that is both non-relational and relational at the same time. It supports two transactional NoSQL (Redis compatible key-value & Cassandra compatible flexible-schema) APIs as well as a distributed SQL (PostgreSQL compatible) API on a common massively-scalable document store.

YugaByte DB is purpose-built to power internet-scale online services on public, private and hybrid clouds with transactional integrity, high availability, low latency, high throughput and multi-region scalability while also providing unparalleled data modeling freedom to application architects. Enterprises gain more functional depth and agility without any cloud lock-in when compared to proprietary cloud databases such as Amazon DynamoDB, Microsoft Azure Cosmos DB and Google Cloud Spanner. Enterprises also benefit from stronger data integrity guarantees, more reliable scaling and higher performance than those offered by legacy open source NoSQL databases such as MongoDB and Apache Cassandra.

This guide helps users gain a deeper technical understanding of YugaByte DB while also highlighting key criteria for evaluating any business-critical database such as YugaByte DB. It is applicable to the version 1.1 of YugaByte DB Enterprise Edition.

How to Evaluate YugaByte DB?

YugaByte DB is a multi-model, cloud native database purpose built for serving internet-scale transactional as well as scale-out RDBMS workloads. The evaluation of such a database therefore involves aspects such as cloud native installation & software upgrade experience, workload performance, cluster expansion, multi-region scalability, infrastructure failure testing and more.

While all the above aspects can be evaluated in the context of the open source YugaByte DB Community Edition, the commercial YugaByte DB Enterprise Edition simplifies many of these tasks so that application development and operations teams can focus more on their application and less on their database infrastructure. The end result is development agility and operations simplicity that allows applications to be released faster than ever before without any compromises to feature completeness, quality assurance and high performance.

Redefining Transactional Databases for Cloud Native Era

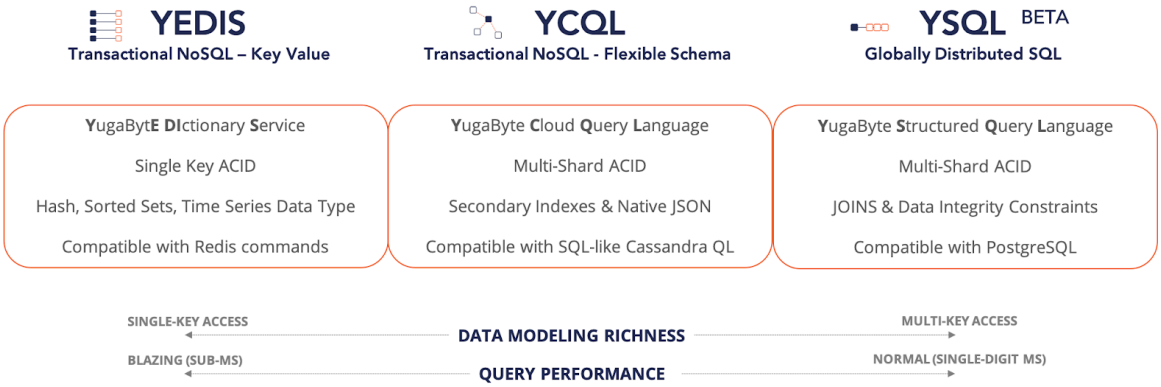
YugaByte DB is a multi-API/multi-model database with sharding, replication and distributed transactions architecture similar to that of Google Spanner. It redefines transactional databases by bringing together three must-have needs of user-facing cloud applications, namely high performance, ACID transactions and multi-region scalability, in a single database. As a **transactional NoSQL database**, it beats legacy distributed NoSQL databases that offer high performance and multi-region scalability but give up on transactional guarantees (because of eventual consistency). As a **distributed SQL database**, it also comes ahead of legacy monolithic SQL databases that offer transactions and performance but are unable to scale writes across multiple nodes or geographic regions.

Multi-API/Multi-Model Transactional App Development

Modern app architectures rely on data with different modeling constraints and access patterns. Polyglot persistence, first introduced in 2011, states that each such data model should be powered by an independent database that is purpose-built for that model. The original intent was to look beyond relational/SQL databases to the emerging world of NoSQL. However, polyglot persistence comes with the hidden cost of increased complexity across the board.

Using multiple databases during development, testing, release and production can be overwhelming for many organizations. While app developers must learn efficient data modeling with various database APIs, they cannot simply ignore the underlying storage engine and replication architectures for each of the databases. Operations teams are now forced to understand scaling, fault-tolerance, backup/restore, software upgrades and hardware portability for multiple databases. And of course the onerous part of "operationalizing" these databases across multiple different cloud platforms. In order to reduce the above operationalization costs, enterprises have gravitated towards to proprietary but managed database services such as Amazon RDS/DynamoDB, Azure Cosmos DB and Google Cloud Spanner. However, the end result is cloud lock-in and astronomical cloud provider bills.

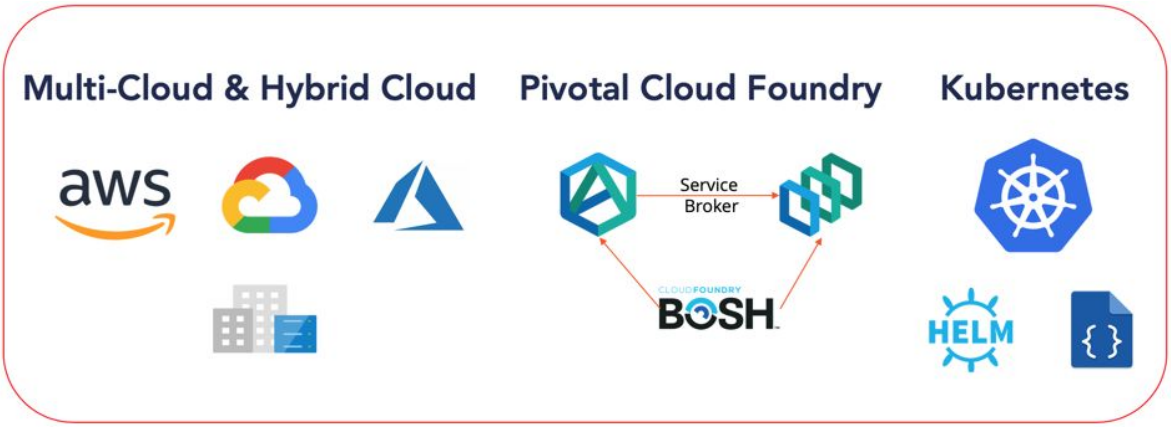
Multi-model databases are on the rise to free users from the complexity of polyglot persistence. YugaByte DB takes the multi-model approach one step further by being also multi-API. It supports two transactional NoSQL APIs and a distributed SQL API.



On one end of the spectrum is the Redis-compatible YEDIS API that is completely optimized for single key access patterns, has simpler data modeling constructs and provides blazing-fast (sub-ms) query performance. On the other end of the spectrum is the PostgreSQL-compatible YSQL API that supports complex multi-key relationships (through JOINS and foreign keys) and provides normal (single-digit ms) query performance. This is expected since multiple keys can be located on multiple shards hosted on multiple nodes, resulting in higher latency than a key-value API that accesses only a single key at any time. At the middle of the spectrum is the Cassandra-compatible YCQL API that is still optimized for majority single-key workloads but has richer data modeling features such as globally consistent secondary indexes (powered by distributed ACID transactions) that can accelerate internet-scale application development significantly.

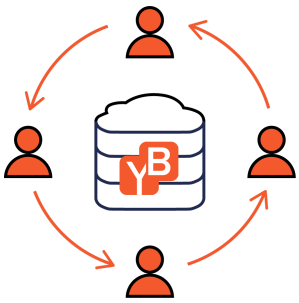
Multi-Region & Multi-Cloud Operations

YugaByte DB is ideal for multi-zone, multi-region, hybrid cloud and multi-cloud deployment options. Users can change infrastructure choices as often as business demands — moving from single region to multi-region as well as moving from single cloud to hybrid cloud or multi-cloud are now zero downtime and zero data loss operations with no application impact. Enterprises are free to expand their business globally anytime they choose without waiting for their database architecture to catch up.



Additionally, as a cloud native database built in the era of microservices, YugaByte DB lends itself extremely well to containerized deployments powered by Kubernetes orchestration. By automatically managing small data partitions and using strongly consistent replication to have multiple copies of such partitions readily available, a YugaByte DB cluster self heals in a few seconds in the event of any infrastructure failure (including container crashes and disk failures). This ensures that there is no noticeable impact on the overall cluster performance or availability.

Enterprise Edition vs. Community Edition



The open source YugaByte DB Community Edition (CE) is the best choice for the startup organizations with strong technical operations expertise. Such organizations would deploy YugaByte DB into production with traditional DevOps tools. The CE includes the core database engine as well as the support for all the three APIs (including the authentication/authorization models supported by these APIs). Data migration tools to help move data from existing databases are also included.

The commercial YugaByte DB Enterprise Edition (EE) includes all the features of the CE as well as additional features such read replicas, in-flight/at-rest encryption, distributed backups, multi-cloud orchestration, seamless cloud portability and comprehensive monitoring. Many of these features can be administered with the included YugaByte DB Admin Console. The EE is the simplest way to run YugaByte DB in mission-critical

production environments with one or more regions (across both public cloud and on-premises datacenters).

How YugaByte DB Works?

Architecture

DocDB is YugaByte DB's Log Structured Merge tree (LSM) based storage engine and Raft is the distributed consensus protocol used for replication of data across the nodes in a cluster. A YugaByte DB cluster is comprised of two types of distributed services, YB-Master and YB-TServer. As described below, both these services are built on top of the DocDB storage engine and the Raft replication protocol. Note that sharding (aka horizontal data partitioning) and replication are automatic in YugaByte DB, so applications do not have to do anything special to leverage these features.

YB-TServer

This service represents the data nodes responsible for hosting/serving user data in shards (also known as tablets). The total number of replicas of a shard is determined by the Replication Factor (RF) of the cluster. These replicas are distributed across all the available data nodes. The number of data nodes can be increased or decreased on-demand in a cluster which will lead to automatic rebalancing of the shards across the nodes.

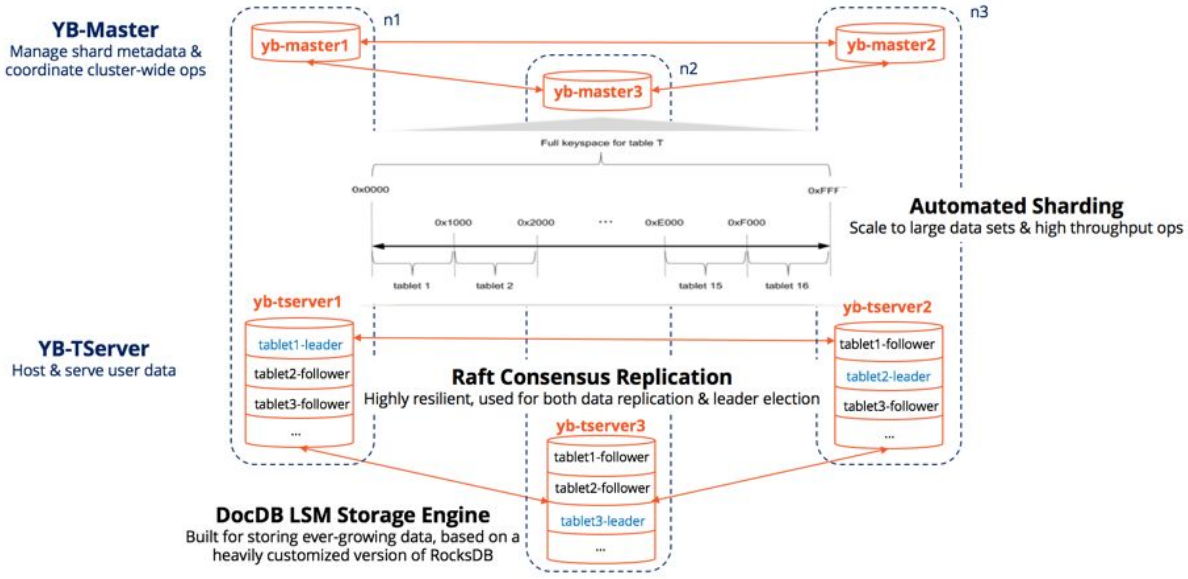
YB-Master

This service is responsible for keeping system metadata (such as shard-to-node mapping), coordinating system-wide operations (such as create/alter drop tables), and initiating maintenance operations (such as load-balancing). For increased fault tolerance, the number of YB-Masters equals the Replication Factor of the cluster.

Example of a 3-Node Cluster

The figure below highlights the architecture of a 3-node RF3 YugaByte DB cluster with 3 instances each of YB-Master and YB-TServer. User and system tables are auto-sharded into multiple tablets. Each tablet has a Raft group of its own (called as tablet-peers) with one leader and a number of followers equal to RF-1. Raft is used for leader election and

data replication in both YB-Master and YB-TServer. Once data is replicated via Raft across a majority of the tablet-peers, it is applied to each tablet peer's local DocDB.



DocDB Document Store

Built on a highly customized version of RocksDB, DocDB is a key-to-document storage engine that can store both primitive and complex data types for high volume data applications. The keys in DocDB are compound keys consisting of 1 or more hash organized components, followed by 0 or more ordered (range) components. These components are stored in their data type specific sort order; both ascending and descending sort order are supported for each ordered component of the key. This model allows multiple levels of nesting, and corresponds to a JSON-like format. More details about DocDB architecture can be found in YugaByte DB Docs.

Raft Distributed Consensus-Based Replication

As described in "How Does Consensus-Based Replication Work in Distributed Databases?", Raft has become the consensus replication algorithm of choice when it comes to building resilient, strongly consistent systems. Following are the key YugaByte DB features that leverage the Raft replication protocol.

- Strong consistency with zero data loss writes
- Continuous availability with self-healing per-shard leader election
- Rapid scale-out and scale-in with auto-rebalancing

- High performance with quorumless reads and tunable latency

The details of the how the above features leverage Raft are available at "How Does the Raft Consensus-Based Replication Protocol Work in YugaByte DB?".

Distributed Transactions

ACID transactions are a fundamental building block when developing business-critical, user-facing applications. They simplify the complex task of ensuring data integrity especially under highly concurrent workloads. While they are taken for granted in monolithic SQL/relational DBs, distributed NoSQL/non-relational DBs usually forsake them completely. MongoDB is an exception in the sense that it supports a restrictive single row/shard ACID option but does not support the fully distributed/multi-shard ACID option.

Implementing multi-shard ACID transactions in distributed databases requires the use of a transaction manager that can coordinate the various operations and then commit/rollback the transaction as needed. Popular NoSQL databases are designed to avoid this additional complexity in the fear that they will have to compromise on performance (in the form of increase in write latency and decrease in linear scalability). On the other hand, every YugaByte DB data node has a built-in transaction manager that is engineered to efficiently detect and optimally handle different scenarios involving single row, single shard and distributed ACID transactions without compromising performance. "Yes We Can! Distributed ACID Transactions with High Performance" details the various optimizations used to deliver low latency and high throughput transactions.

Key Evaluation Criteria

Following are the criteria that should be used to evaluate any mission-critical operational database including YugaByte DB Enterprise Edition. The remaining sections go into the details of each of these.

Cloud Native Operational Experience

YugaWare, the YugaByte DB Admin Console included in the Enterprise Edition, is a turnkey multi-cloud, multi-cluster management plane for YugaByte DB. It is deeply integrated with the infrastructure automation services of public clouds such as Amazon Web Services and Google Cloud (Microsoft Azure is on the roadmap). The net result is the ability to instantly spin up/down/expand YugaByte DB clusters as well as take distributed backups with a cloud native operational experience that was previously available only for

proprietary cloud services. The Admin Console also offers integration for on-premises datacenter environments and Kubernetes deployments.

Performance Testing

YugaByte DB performance tests can be executed against either a suite of pre-built sample apps that ship in YugaByte DB or through the industry standard YCSB benchmark. The sample apps include multiple key-value workloads that are either read-heavy or write-heavy or batch writes. The YCSB benchmark ships with 6 core workloads that cover the spectrum from write-heavy workloads to read-modify-write workloads.

Multi-Model Capabilities

YugaByte DB provides unparalleled data modeling freedom to internet-scale apps by providing two transactional NoSQL (key-value and flexible-schema) and a distributed SQL API.

Cluster Expansion

YugaByte DB is architected to ensure that adding nodes to an existing cluster irrespective of the current size of the cluster should be a fully online operation with zero application impact.

Multi-Region Scalability

YugaByte DB ensures that requests for strongly consistent reads are served from the tablet leader while requests for follower reads are served from the followers in the nearest datacenter.

Since strongly consistent reads can have higher latency if the tablet leader is located in a different region, latency sensitive applications can choose to read from nearby followers at a lower latency by going for timeline consistency as opposed to strong consistency.

Preferred Region Placement and Continuous Balancing

YugaByte DB has the ability to place the tablet leaders only in specific AZs of specific regions. If a workload depends on strongly consistent reads from a preferred region, then YugaByte DB can ensure that those reads can be served with the absolute low latency. This configuration can be changed even on a running cluster thus accounting for the fact that workloads change their access pattern with time.

Node/Zone/Region Failure Testing

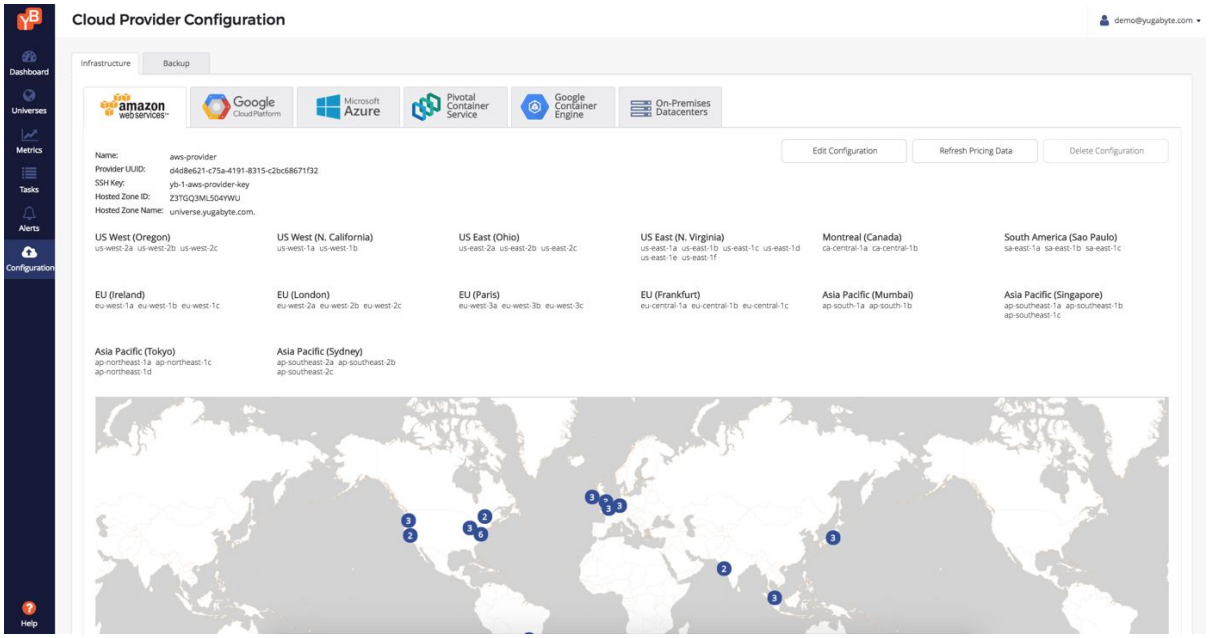
Infrastructure failures such as machine crashes, disk failures, network partitions and clock skews are bound to happen when using today's commodity hardware. As a highly fault tolerant database, YugaByte DB protects the cluster against data loss and self heals quickly to ensure continuous availability. Common tests in this area are testing a node/AZ failure in a single region deployment and testing a single region failure in a multi-region deployment.

Evaluation Steps

Install YugaByte DB Enterprise Edition

First, we will install the YugaByte DB Admin Console application using instructions from YugaByte DB Docs. We can then administer YugaByte DB EE clusters with extreme ease on any public or private cloud. A dedicated machine separate from the machines used for the DB cluster is needed to run this application. Both Internet-connected and airgapped installation options are supported for the Admin Console.

Configure One or More Cloud Providers



After the Admin Console is installed, we can configure any number of cloud providers so that YugaByte DB clusters can be created and dynamically managed on these cloud providers. As you can see above, major public cloud platforms as well as major Kubernetes distributions are supported. Even on-premises datacenters can be configured. Note that the Admin Console does not manage the underlying VM/bare metal instances in case of on-premises datacenters and is simply responsible for managing the DB cluster on those instances.

Test Performance with YCSB

As a strongly consistent distributed database, a fault-tolerant YugaByte DB cluster starts with a minimum of 3 nodes and a Replication Factor of 3. These nodes can be running on machines with either Centos 7.x or RHEL 7.x. Either local or remote-attached disks should be mounted on these machines. A complete system configuration for performance testing is highlighted below.

Note that single node local clusters can be used to learn more about the database but are not recommended for any functional test or performance test environments. A complete checklist for deploying YugaByte DB is available here.

Recommended System Configuration

Amazon Web Services

- Instance type: i3.4xlarge
- Disks: 2 x 1.9 TB NVMe SSDs (comes pre-configured with the instance)

Google Cloud Platform

- Instance type: n1-standard-16
- Disks: 2 x 375 GB SSDs

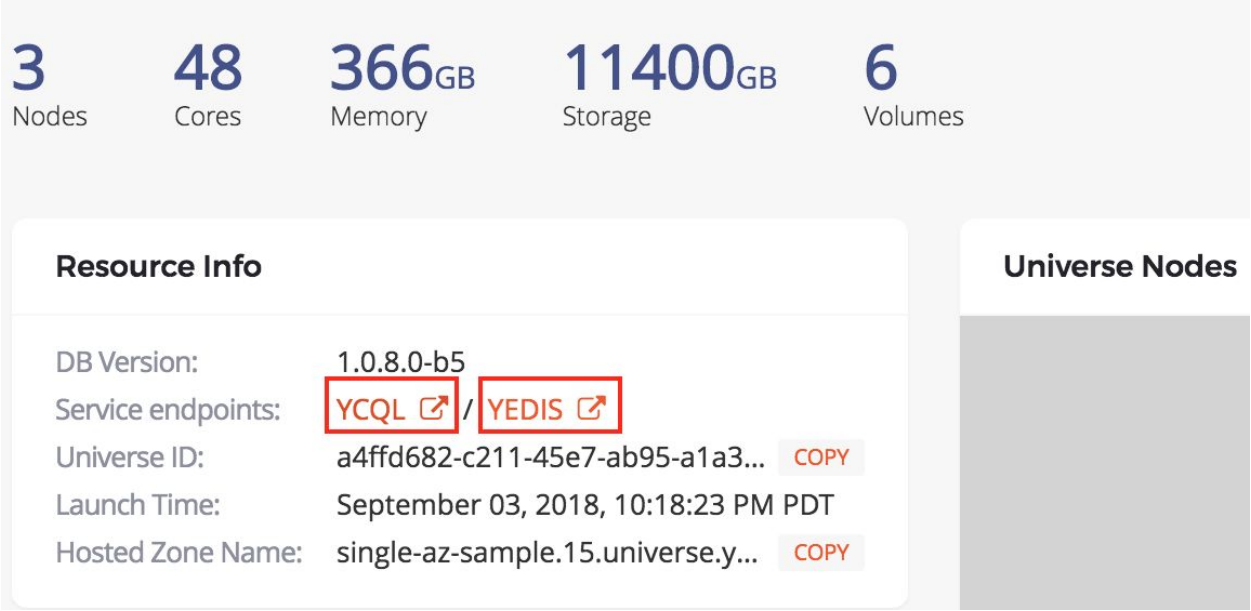
On-Premises Datacenter

- Instance: 16 CPU cores
- Disk size: 1 x 200 GB SSD (minimum)
- RAM size: 30 GB (minimum)

Create YugaByte DB Universe

From the main Dashboard page of the Admin Console, create a new YugaByte DB universe using the system configuration highlighted above. After the provisioning complete, the Universe Details page should show you the endpoints for the

Cassandra-compatible YugaByte Cloud Query Language (YCQL) API that we will use hereafter for the YCSB testing.



Assume $ENDPOINTS are the IP addresses (plus port) for the nodes, e.g. for YCQL:

```
ENDPOINTS="172.151.20.150:9042,172.151.23.97:9042,172.151.24.41:9042"
```

Setup YCSB

Yahoo Cloud Serving Benchmark (aka YCSB) is widely considered the de-facto standard for evaluating the performance of NoSQL databases using a variety of single-key access patterns used in highly scalable cloud applications.

Detailed instructions on how to setup the YCSB test suit for YugaByte DB are available in the Appendix of this paper.

Run YCSB & Review Results

Simply run the script that you prepared in the previous step. You can also run the workloads in parallel as documented here.

```
./run-yb-ycsb.sh
```

Note that the result for each workload will be in workload[abcdef]-transaction.dat (e.g. workloada-transaction.dat).

Test Multi-Model Capabilities

Along with YCQL, YugaByte DB also provides a transactional key-value API called YugaByte Directory Service (YEDIS). YEDIS is wire compatible with the Redis commands and client drivers. On the universe we had previously created, we will now run a YEDIS key-value workload to demonstrate the ability of a single cluster to handle different models/APIs at the same time.

Assume $ENDPOINTS are the IP addresses (plus port) for the nodes, e.g. for YEDIS:

```
ENDPOINTS="172.151.20.150:6379,172.151.23.97:6379,172.151.24.41:6379"
```

You can find this information from the Resource Info widget on the Universe Details page.

Read-heavy workload

Get the yb-sample-apps repository.

```
git clone https://github.com/YugaByte/yb-sample-apps.git
mvn -DskipTests package
ls target/yb-sample-apps.jar
```

Run the key-value workload with a higher number of read threads (representing read-heavy workload).

```
java -jar target/yb-sample-apps.jar --workload RedisKeyValue --nodes $ENDPOINTS --num_threads_read 128 --num_threads_write 16
```

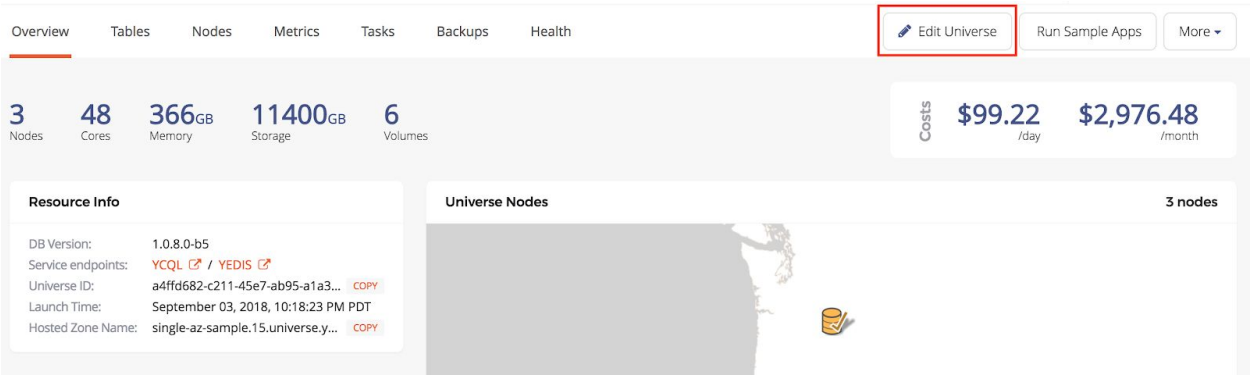
Expected Results

- Read Ops/sec: **~100k**
- Read Latency: **~1.25 ms**
- Write Ops/sec: **~6k**
- Write Latency: **~2.70 ms**
- CPU (User + Sys): **65%**

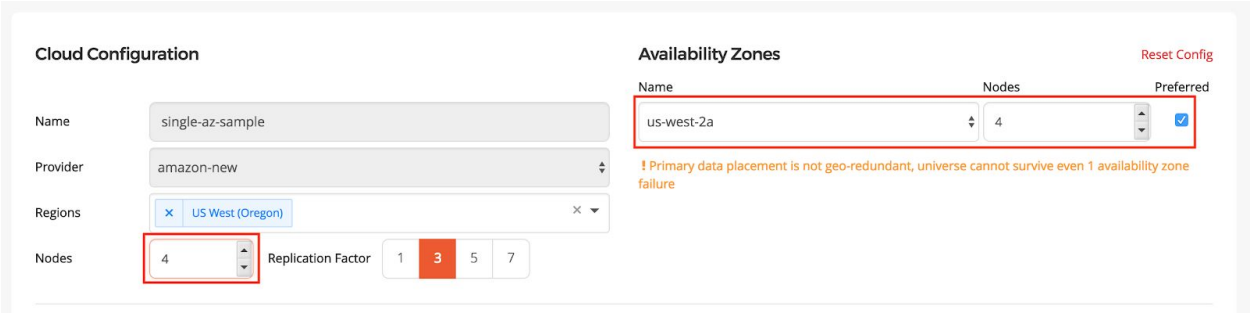
Test Cluster Expansion

Universe Expand Operation

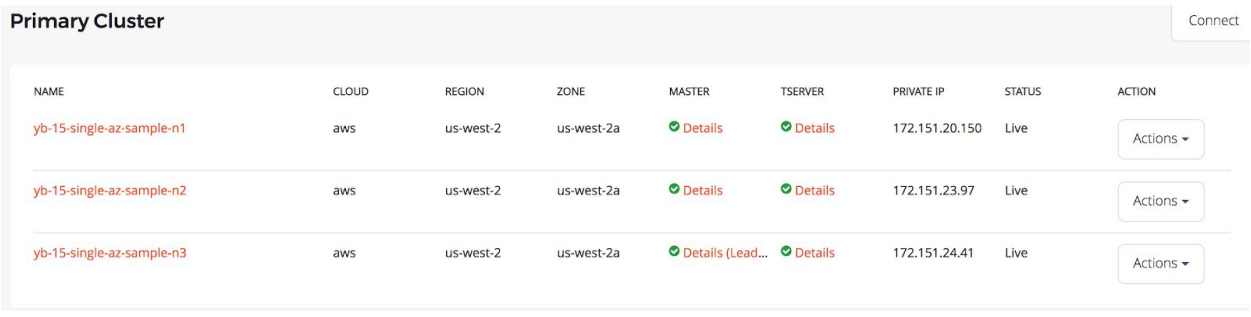
Select "Edit Universe" and increase number of nodes to the desired value. Complete documentation for all aspects of Edit Universe are available in YugaByte DB Docs.



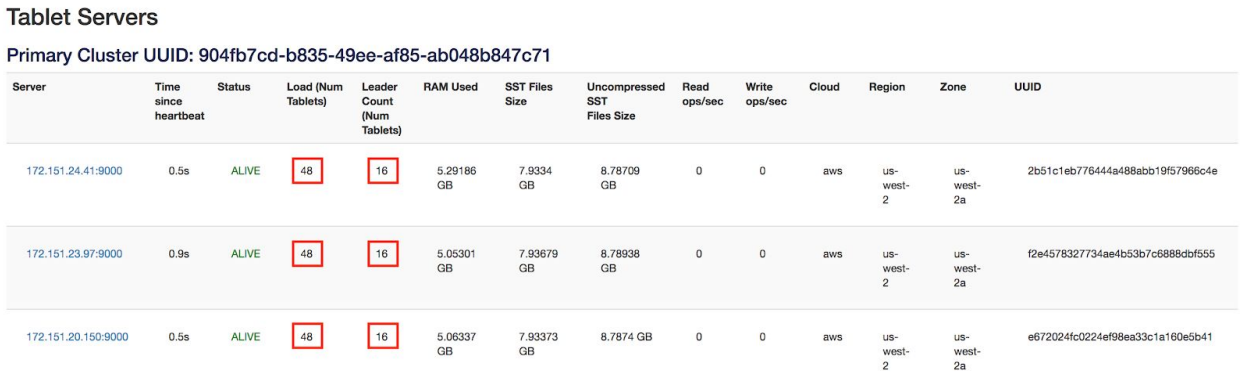
Under the "Cloud Configuration" section, increase the Nodes count as desired.



Double-check the node distribution in the "Availability Zones" section is the expected (for example testing a single-zone Universe).Then, on the Universe page, go to the Nodes tab.



Click on the link for the Master leader [with "Details (Leader)" in the MASTER column], and then on the Tablet Servers link in the master admin UI.



Tablet Servers

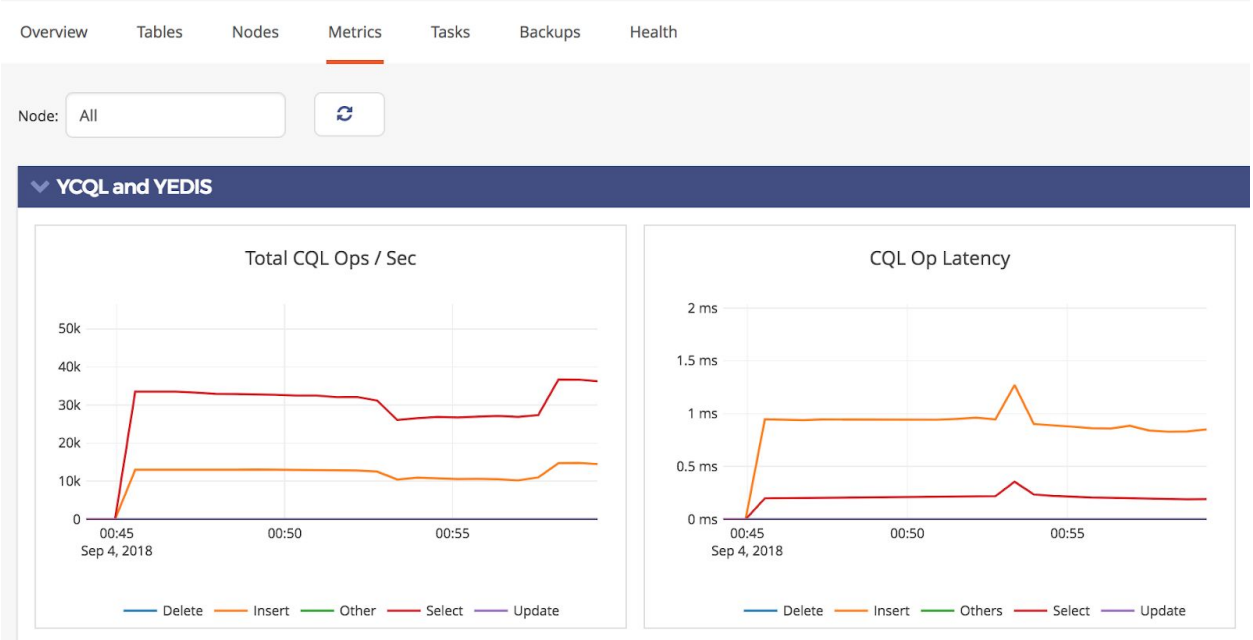
Primary Cluster UUID: 904fb7cd-b835-49ee-af85-ab048b847c71

Server	Time since heartbeat	Status	Load (Num Tablets)	Leader Count (Num Tablets)	RAM Used	SST Files Size	Uncompressed SST Files Size	Read ops/sec	Write ops/sec	Cloud	Region	Zone	UUID
172.151.24.41:9000	0.5s	ALIVE	48	16	5.29186 GB	7.9334 GB	8.78709 GB	0	0	aws	us-west-2	us-west-2a	2b51c1eb776444a488abb19f57966c4e
172.151.23.97:9000	0.9s	ALIVE	48	16	5.05301 GB	7.93679 GB	8.78938 GB	0	0	aws	us-west-2	us-west-2a	f2e4578327734ae4b53b7c6888dbf555
172.151.20.150:9000	0.5s	ALIVE	48	16	5.06337 GB	7.93373 GB	8.7874 GB	0	0	aws	us-west-2	us-west-2a	e672024fc0224ef98ea33c1a160e5b41

The example image above shows a 3-node universe with each TServer having 48 tablets or shards (48 * 3 = 144 tablets). This means 48 tablets (16 of which are leaders) per node. This corresponds to 36 tablets (and 12 leaders) per node in a 4-node universe.

Cluster During Expansion

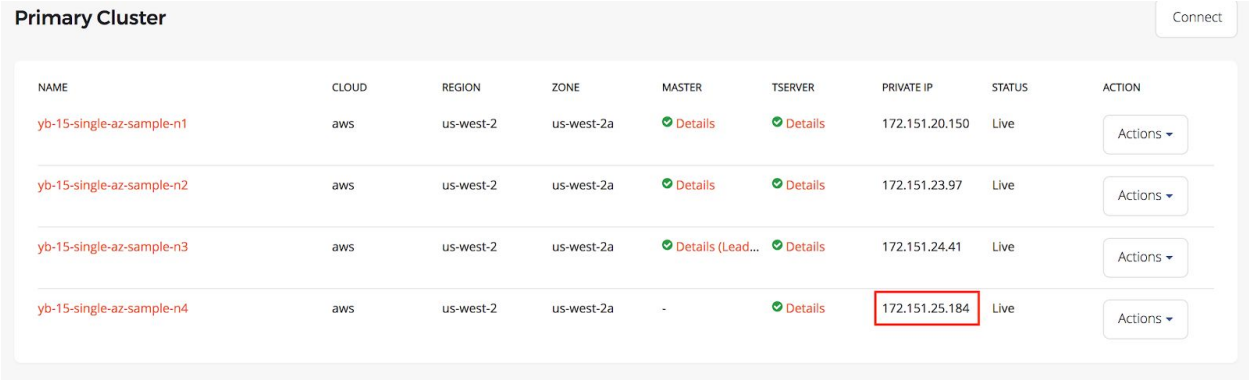
The expand operation should take under 10 minutes; but this is a fully online operation with the cluster still taking traffic. The progress of various steps is reported in the control plane. Once the node is provisioned, and software is installed and configured, the tablets are moved automatically in a throttled and safe manner all in an online manner.



In the image above, you can see a slight dip in ops/sec during the expansion with the number increasing after the expand operation is done.

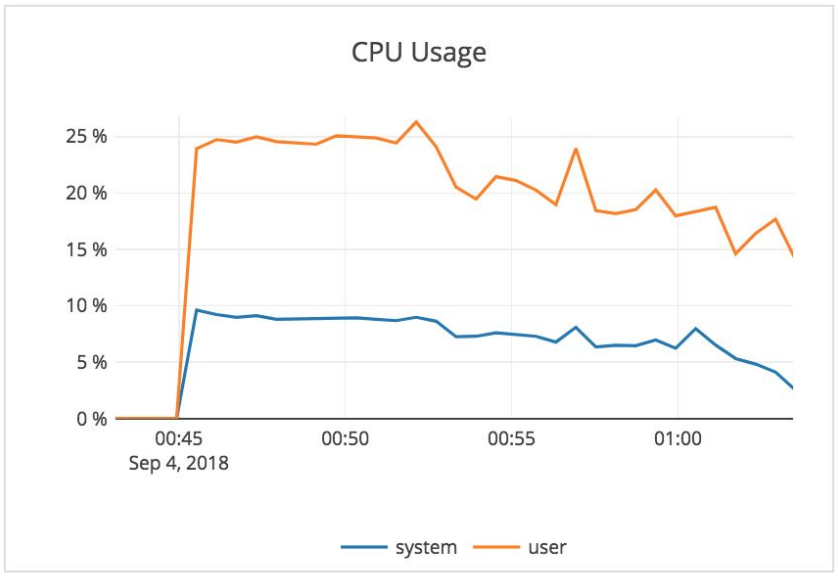
Cluster After Expansion

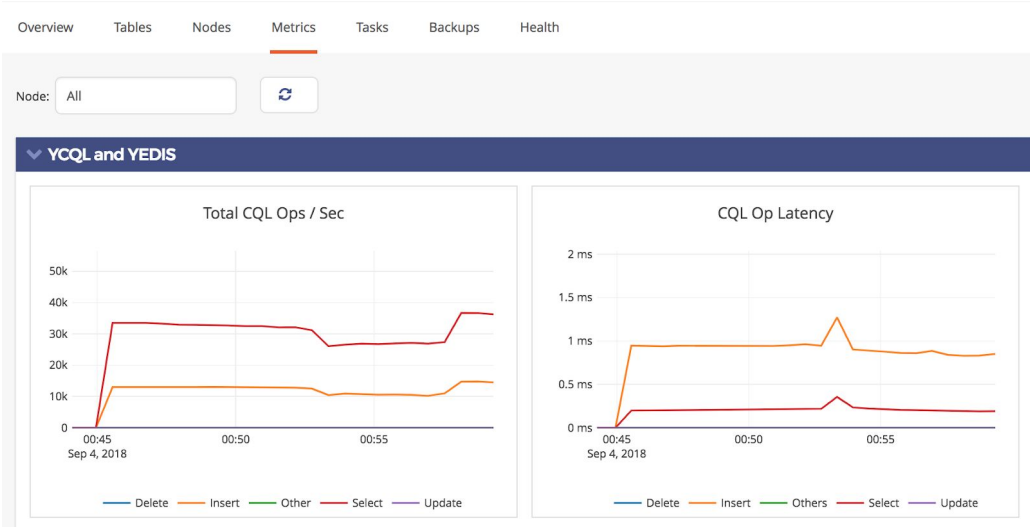
After expansion is done the new node should appear Live on the Nodes tab (note the IP of the new node).



View of tablets (shards) on each tablet server shows that those 144 tablets are now spread evenly across the 4 nodes (so 36 tablets, 12 tablet leaders per node).

As a result of the expand, you can see the drop in CPU-per-node & latency while the IOPS cluster wide increase.





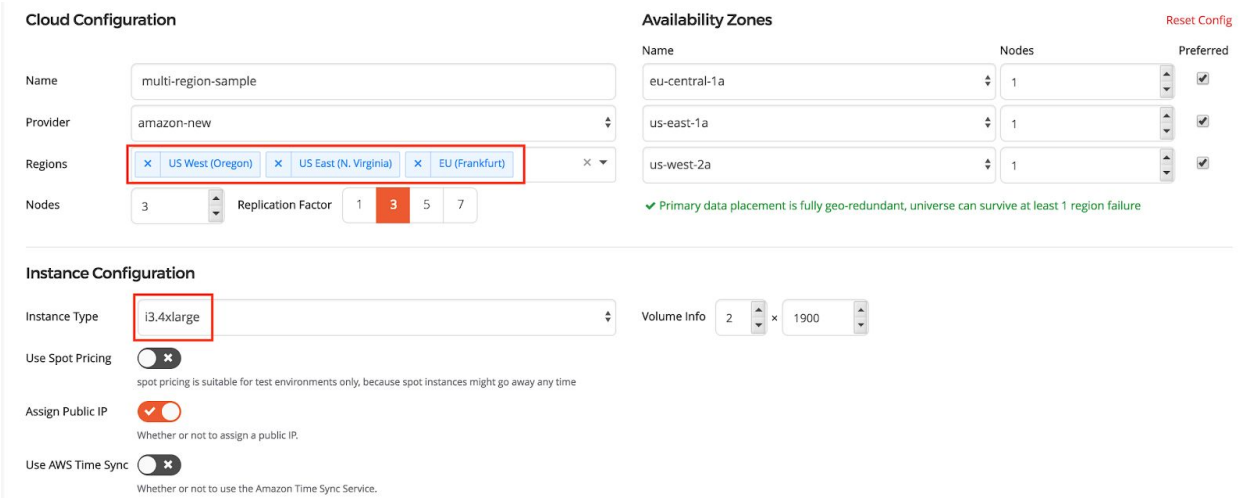
Test Multi-Region Scalability

We will now create a new cluster which has nodes spanning multiple regions. Complete documentation regarding multi-region deployments is available in YugaByte DB Docs.

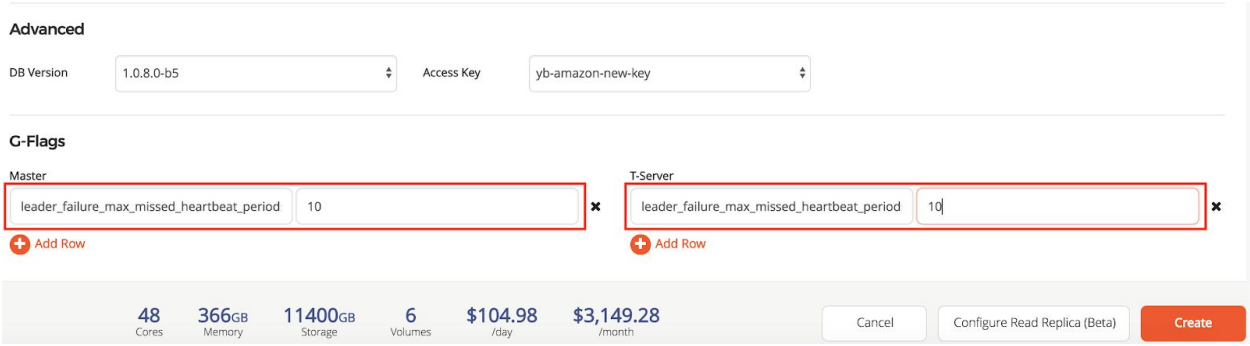
Following is the configuration of the multi-region universe we will create:

- Regions: (US West) Oregon, (US East) N. Virginia, (Europe) Frankfurt
- Nodes: 3
- Machine Type: i3.4xlarge

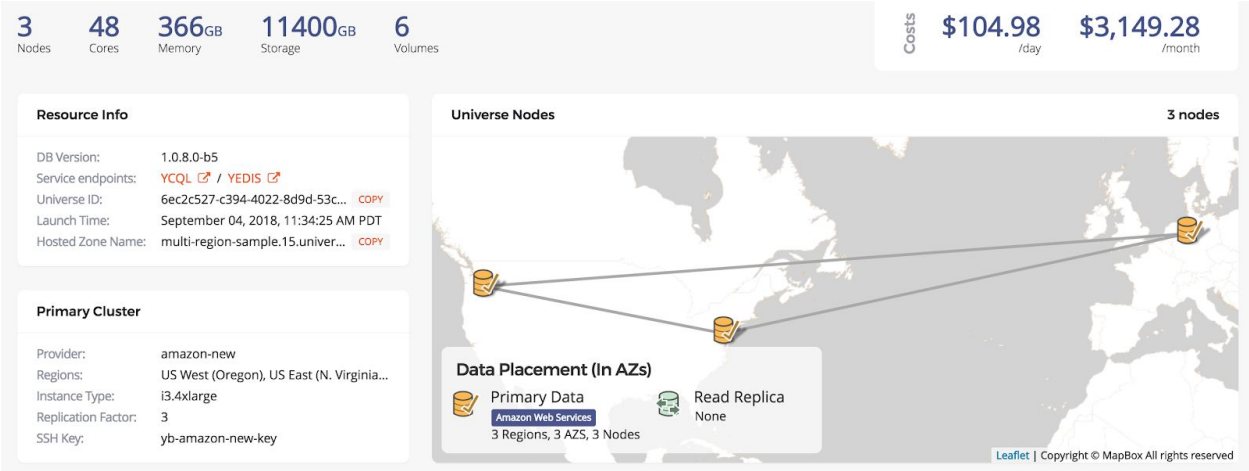
Create a Universe as before but add the 3 regions (US West) Oregon, (US East) N. Virginia, (Europe) Frankfurt in the Regions field under Cloud Configuration. We will use the same node type as before (i3.4xlarge).



Make sure to add the following G-Flag for both YB-Master and YB-TServer: leader_failure_max_missed_heartbeat_periods = 10. Since the the data is globally replicated, RPC latencies are higher. We use this flag to increase the failure detection interval in such a higher RPC latency deployment. See the screenshot below.



Then click Create as usual -- once the universe is created the overview page should look like below:



Workload Description

Running a workload representing users with their profile, modeled as a simple Cassandra KV table. This workload is “read-heavy” (representing users logging in, and hence need read access to the profile; it is fairly light in terms of writes - which correspond to someone updating their profile).

Note: Connect to the load-tester and get the (YCQL) ENDPOINTS exactly as in the single-az case above.

Write-only Workload

Load 1M keys (which will also be used by the read-heavy workloads below).

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS --num_threads_write 256 --num_threads_read 0 --value_size 128 --num_unique_keys 1000000 --nouuid --with_local_dc us-west-2
```

Expected Results

(Write) Ops/Sec: **1219.93**
(Write) Latency: **210.18 ms**
CPU Usage: **< 5%** (Should be bottlenecked by latency)

The high write latencies are expected because of a multi-region setup. Such a setup naturally depends on the round-trip network time between the regions since YugaByte DB uses distributed consensus on writes.

Strongly Consistent Reads Workload

YugaByte DB reads are strongly consistent by default. In this case they, the leaders will be evenly distributed across the 3 nodes so read queries should be evenly distributed accordingly.

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS --num_threads_write 8 --num_threads_read 64 --value_size 128 --num_unique_keys 1000000 --nouuid --with_local_dc us-west-2
```

Expected Results
(Read) Ops/Sec: **851.34**
(Read) Latency: **75.30 ms**
(Write) Ops/Sec: **45.40**
(Write) Latency: **175.94 ms**
CPU Usage: **<5%** (Should be bottlenecked by latency).

The high write latencies are expected because of a multi-region setup. Such a setup naturally depends on the round-trip time between the regions since YugaByte DB uses strongly consistent reads by default and the tablet leaders are distributed evenly over all nodes where we expect one third of the reads to go to each node (see also consistent prefix reads as well as preferred region feature below).

Consistent Prefix Reads Workload

We can set the read consistency to consistent prefix (allowing reads from follower tablets) by using the --local_reads flag. Since this a 3-node RF3 cluster, the local (US West) node will be able to answer all queries so we expect local AZ latency -- since the load tester is in the same AZ.

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS --num_threads_write 8 --num_threads_read 64 --value_size 128 --num_unique_keys 1000000 --nouuid --with_local_dc us-west-2 --local_reads
```

Expected Results

- (Read) Ops/Sec: **~66k**
- (Read) Latency: **0.95 ms**

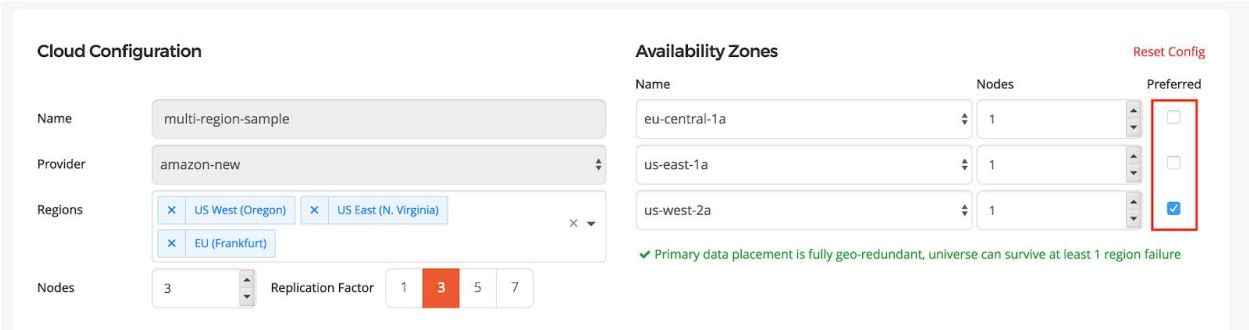
- (Write) Ops/Sec: **~32**
- (Write) Latency: **~240 ms**
- CPU Usage: **~22%**

The high write latencies remain because writes are still strongly consistent (require quorum).

Test Preferred Region Placement & Continuous Balancing

In this workload, we are also using YugaByte “preferred region” capabilities where we can specify that “Oregon” (us-west-2) as the preferred region to keep the tablet leaders for various shards.

See below, how in the “universe specification” we have under the “Preferred” column next to us-west-2a a checkbox indicating that it is the preferred DC for this setup. This is an optional setting - but a nice feature to have if you know that most of the read/write access is going to come from one (or more) particular zones/region(s).



Note: Before clicking Edit do the following two steps to monitor the cluster during the edit operation.

Run a workload

java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS --num_threads_write 8 --num_threads_read 16 --value_size 128 --num_unique_keys 1000000 --nouuid --with_local_dc us-west-2

Note: Connect to the load-tester and get the YCQL ENDPOINTS as before.

On the Master leader web UI (Tablet Servers page) check the tablet leader distribution before and after setting the preferred region.

(Now you can press *Edit* and keep monitoring a. and b. above)

Expected Tablet Distribution (Before → After)

Server	Time since heartbeat	Status	Load (Num Tablets)	Leader Count (Num Tablets)	RAM Used	SST Files Size	Uncompressed SST Files Size	Read ops/sec	Write ops/sec	Cloud	Region	Zone	UUID
172.151.27.87:9000	1.0s	ALIVE	48	48	873.126 MB	0 B	0 B	37346.1	75.1639	aws	us-west-2	us-west-2a	24c01d6186da46e397c4181139c04b22
172.152.21.212:9000	0.3s	ALIVE	48	0	853.362 MB	0 B	0 B	0	0	aws	us-east-1	us-east-1a	01262f00359849e7b75c12db6a51baec
172.158.20.93:9000	0.8s	ALIVE	48	0	851.947 MB	0 B	0 B	0	0	aws	eu-central-1	eu-central-1a	a8a014f6d5b749cea9a1b382ff864319

Primary Cluster UUID: 61cd09e9-d051-4ee6-ad49-7ab612b78671

Server	Time since heartbeat	Status	Load (Num Tablets)	Leader Count (Num Tablets)	RAM Used	SST Files Size	Uncompressed SST Files Size	Read ops/sec	Write ops/sec	Cloud	Region	Zone	UUID
172.151.27.87:9000	0.1s	ALIVE	48	16	872.278 MB	0 B	0 B	67.285	15.3548	aws	us-west-2	us-west-2a	24c01d6186da46e397c4181139c04b22
172.152.21.212:9000	0.6s	ALIVE	48	16	849.355 MB	0 B	0 B	56.9894	13.2362	aws	us-east-1	us-east-1a	01262f00359849e7b75c12db6a51baec
172.158.20.93:9000	0.9s	ALIVE	48	16	852.149 MB	0 B	0 B	71.9674	16.9923	aws	eu-central-1	eu-central-1a	a8a014f6d5b749cea9a1b382ff864319

Expected Results (Before → After)

- (Read) Ops/Sec: **~200 → 37k** (almost 200x increase)
- (Read) Latency: **~85 → 0.5 ms**
- (Write) Ops/Sec: **~44 → 75**
- (Write) Latency: **~180 → 100 ms**
- CPU Usage (User + Sys): **<2% → ~11%**

Note that the write latencies (and ops/sec) also improve because getting a quorum can now be faster; It does not need to involve the farther away Frankfurt node at all -- only the two closer ones Oregon (local) and N. Virginia.

Test Node, Zone and Region Failures

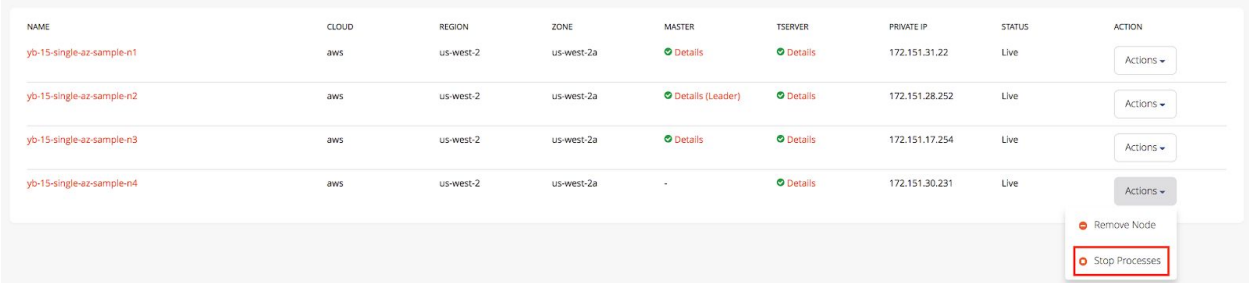
Note: Connect to the load-tester and get the YCQL ENDPOINTS exactly as before. Make sure to use the endpoints of the single-az universe for a) below and the endpoints of the multi-region universe for b) below.

Run a workload with 64 readers and 8 writer threads throughout the failure testing process.

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS --nouuid --value_size 256 --num_threads_read 32 --num_threads_write 32 --num_unique_keys 10000000
```

Testing single node failure in a single region deployment

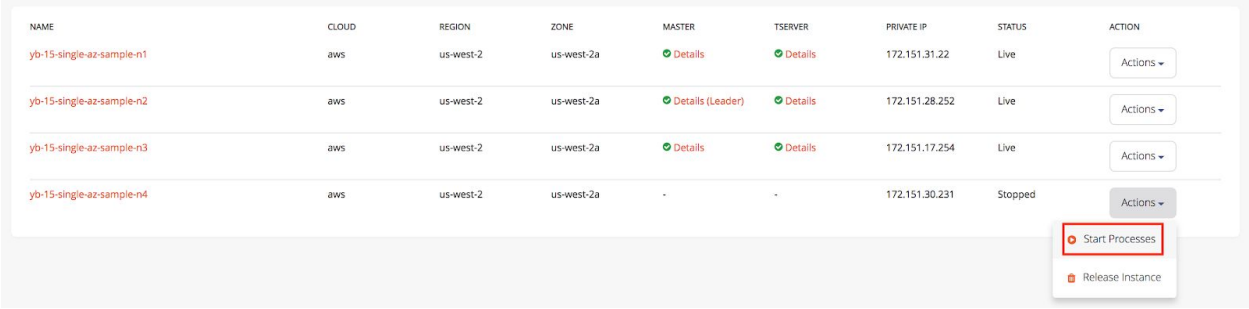
With load running, go to the "Nodes" tab on the universe page. On the "ACTION" column select "Stop processes" for the relevant node.



During the simulated node failure, you can check the following.

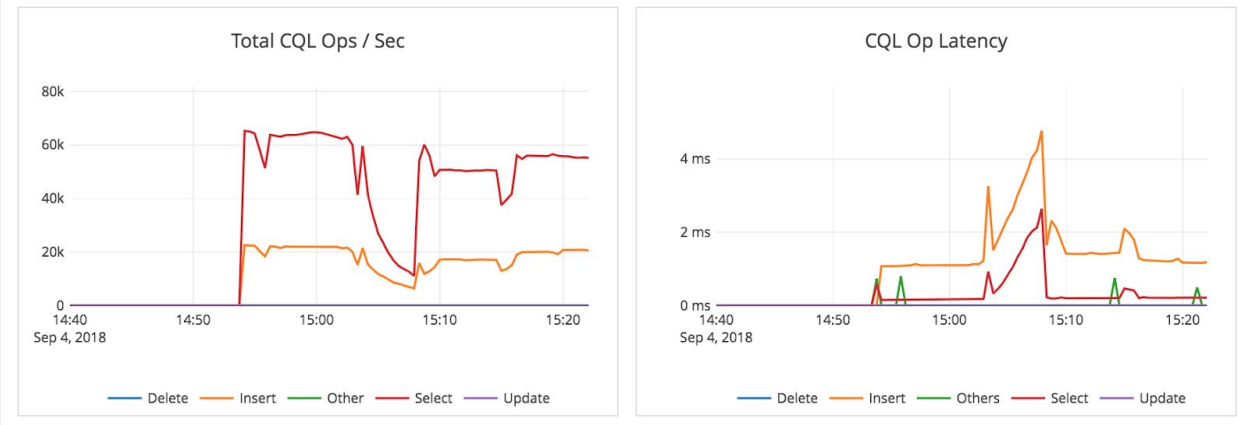
1. On the "Metrics" tab to check the ops/sec, latency, etc.
2. On the Master leader web UI (Tablet Servers page) the tablets/leaders distribution (on the nodes).

Finally, bring the node back up by doing "Start processes" on the "ACTION" column from the "Nodes" tab.



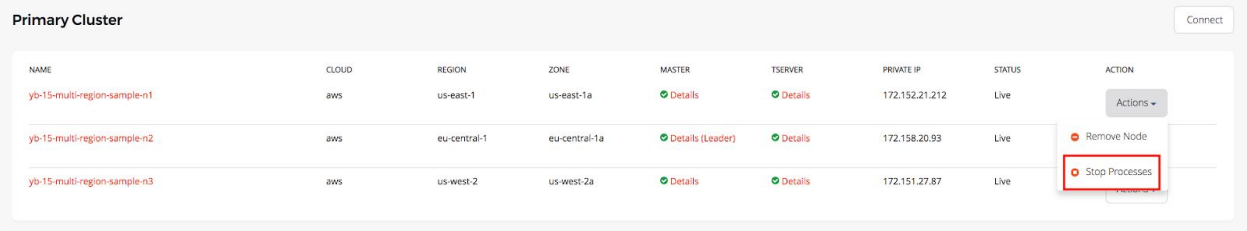
Expected Results

Metrics should slightly lower ops/sec (depending on overall load) while the one node is down but the cluster should still be able to take both reads and writes. Additionally there may be some temporary dips immediately after stopping or starting the processes are stopped/started (seen below at around 15:05 and, respectively, 15:15).



Testing a single region failure in the multi-region deployment

With load running, go to the "Nodes" tab on the universe page. On the "ACTION" column select "Stop processes" for the node(s) in the relevant region.



During the simulated region failure, you can check the following:

1. On the "Metrics" tab to check the ops/sec, latency, etc.
2. On the Master leader web UI (Tablet Servers page) the tablets/leaders distribution (on the nodes).

Finally, bring the node(s) back up by doing "Start processes" on the "ACTION" column from the "Nodes" tab.

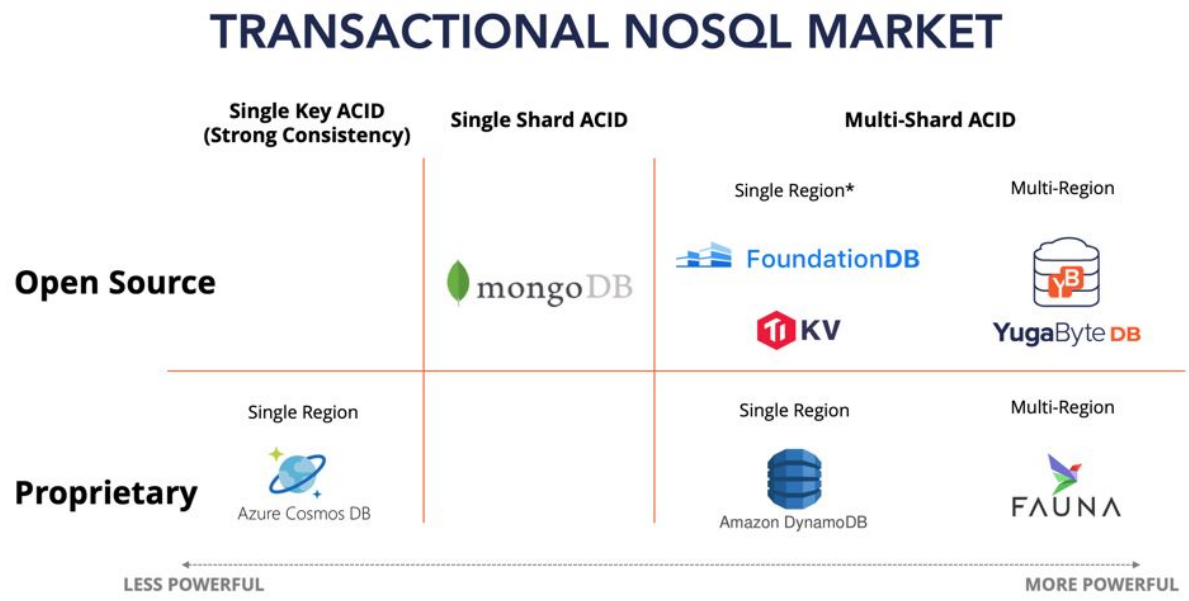
Expected Results

Metrics should report a drop in ops/sec (depending on overall load) while the one node is down but the cluster should still be able to take both reads and writes. The exact values may vary significantly depending on settings (local or remote region is down, if using preferred region, if reads/write are concentrated in one region, etc).

Comparison with Other Databases

By virtue of its multi-model capabilities, YugaByte DB can be thought as a transactional NoSQL as well as a distributed SQL database at the same time.

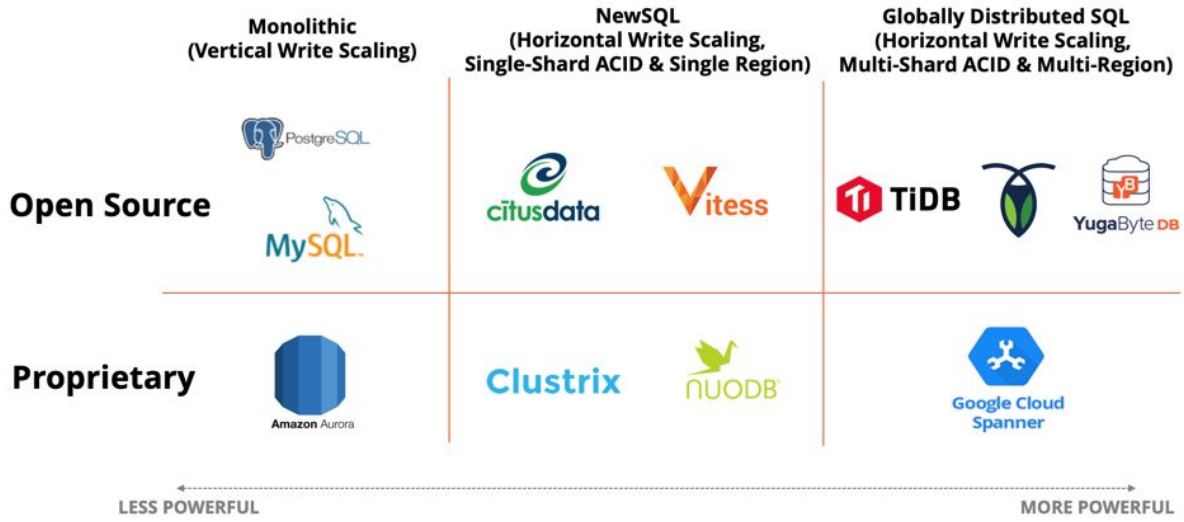
Transaction NoSQL



As highlighted in our [“Why are NoSQL Databases Becoming Transactional?”] post, popular NoSQL databases of the past including Amazon DynamoDB and MongoDB are becoming transactional. These databases are adding transactional features in order to become more attractive to developers in the context of powering business-critical apps that require zero-data-loss even under extreme failure conditions.

Through its multi-shard ACID capabilities that can even be deployed across multiple regions, YugaByte DB leads these transactional NoSQL databases in terms of functional depth. Additionally, YugaByte DB's high-performance distributed document store ensures that performance is not compromised even while serving internet-scale transactional workloads.

Distributed SQL



The distributed SQL market can be summarized as per the figure above. The NewSQL movement was a good attempt at solving some of the core issues in monolithic SQL such as manual sharding. However, the lack of true fault tolerance and multi-shard transactions meant that NewSQL remained a band-aid and never achieved mass adoption. Application architects responsible for launching business-critical microservices using cloud native principles are no longer restricted to monolithic SQL or NewSQL databases when it comes to their data tier. Google Spanner-inspired globally distributed SQL databases such as YugaByte DB are solving every core issue of the past and hence are laying the foundation for highly reliable and highly resilient data infrastructure in the cloud native era.

Conclusion

YugaByte DB is meant to be a system-of-record/authoritative database that distributed applications can rely on for correctness and availability. It allows applications to easily scale up and scale down across multiple regions in the public cloud, on-premises datacenters or across hybrid environments without creating operational complexity or increasing the risk of outages. With its cloud native operational experience and

enterprise-grade features, YugaByte DB Enterprise Edition ensures that enterprises can develop, deploy and operate mission-critical clusters with unparalleled ease. This guide provides an outline for evaluating YugaByte DB Enterprise across various real-world scenarios.

What's Next?

YugaByte DB Documentation, Downloads & Guides

- YugaByte DB Documentation
- YugaByte DB Downloads and Quick Start Guide

Have Questions or Need Help?

- YugaByte DB on Stack Overflow
- YugaByte DB on GitHub
- Contact Us

Appendix

Yahoo Cloud Serving Benchmark (YCSB)

Setup YCSB

Clone the YCSB repository
```
cd $HOME
git clone https://github.com/brianfrankcooper/YCSB.git
cd YCSB
```

Use the Java driver for the Cassandra-compatible YCQL.
1. In pom.xml change the line:
```
<cassandra.cql.version>3.0.0</cassandra.cql.version>
```
to the latest version of the driver:
```
<cassandra.cql.version>3.2.0-yb-18</cassandra.cql.version>
```

Note: You can (and probably should) always check Maven to find the latest version.

2. In cassandra/pom.xml change the line:
```
<groupId>com.datastax.cassandra</groupId>
```
to:
```
<groupId>com.yugabyte</groupId>
```

3. Build YCSB and the YCQL binds
```
mvn -pl com.yahoo.ycsb:cassandra-binding -am clean package -DskipTests
```

Setup YugaByte DB's fork of the cqlsh shell

```
cd $HOME
git clone https://github.com/YugaByte/cqlsh
```

Run YCSB Tests

1. Create an executable file in the YCSB folder:

```
cd $HOME/YCSB
touch run-yb-ycsb.sh
chmod a+x run-yb-ycsb.sh
```

2. Copy the following contents in run-yb-ycsb.sh:

Note: You may want to change the highlighted values below with the correct/intended ones for your setup.

```
#!/bin/bash


# YB-CQL host (any of the yb-tserver hosts)
# (The other nodes should get automatically discovered by the driver).
hosts=127.0.0.1
ycsb=$HOME/YCSB/bin/ycsb
cqlsh=$HOME/cqlsh/bin/cqlsh <ip-addr>
ycsb_setup_script=$HOME/YCSB/cassandra/src/test/resources/ycsb.cql
keyspace=ycsb
table=usertable
# See https://github.com/brianfrankcooper/YCSB/wiki/Core-Properties for param descriptions
params="-p recordcount=1000000 -p operationcount=10000000"

setup() {
  $cqlsh <<EOF
create keyspace $keyspace with replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
EOF
  $cqlsh -k $keyspace -f $ycsb_setup_script
}

cleanup() {
  $cqlsh <<EOF
drop table $keyspace.$table;
drop keyspace $keyspace;
EOF
}

delete_data() {
  $cqlsh -k $keyspace <<EOF
drop table $table;
EOF
  $cqlsh -k $keyspace -f $ycsb_setup_script
}


run_workload() {
  local workload=$1----------------
  echo ========================== $workload ==========================
  $ycsb load cassandra-cql -p hosts=$hosts -P workloads/$workload $params \
   -p threadcount=40 | tee $workload-load.dat
  $ycsb run cassandra-cql -p hosts=$hosts -P workloads/$workload $params \
   -p cassandra.readconsistencylevel=QUORUM -p cassandra.writeconsistencylevel=QUORUM \
   -p threadcount=256 -p maxexecutiontime=180 | tee $workload-transaction.dat
  delete_data
}

setup
```

```
load_data workloada
run_workload workloada
run_workload workloadb
run_workload workloadc
run_workload workloadf
run_workload workloadd

truncate_table
load_data workloade
run_workload workloade

cleanup
```

Note that YugaByte DB's defaults of strongly consistent reads and writes correspond to the QUORUM read/write consistency level in Apache Cassandra as shown in the command above.