MongoDB on VMware vSAN 6.6 All-Flash

Table of contents

MongoDB on VMware vSAN 6.6 All-Flash 4
Executive Summary 4
Business Case 4
Solution Overview 4
Key Results 4
Introduction 5
Purpose 5
Scope 5
Audience 5
Technology Overview 6
VMware vSphere 6.5 6
VMware vSAN 6.6 6
MongoDB 3.4 6
Solution Configuration 8
Solution Architecture 8
Hardware Resources 9
Software Resources 10
VM and Database Configuration 11
MongoDB Storage Engine 11
Performance Test Tool and Settings 11
Solution Validation 14
Overview 14
Evaluate the Impact of Different Client Threads 15
Evaluate the Impact of Different Operation Count 16
Parameter Settings in the Baseline Testing 17
Evaluate the Impact of Different Virtual CPU Cores and Memory Configurations 18
Evaluate the Impact of Different Database Size 19
Evaluate the Impact of MongoDB Cluster Scaling out 20
Evaluate the Impact of MongoDB Replica Set Setting 22
Evaluate the Impact of Different vSAN Object Stripe Width 23
Evaluate the Impact of Different vSAN Failure Tolerance Method 24
Evaluate the Impact of Different Durability Options 25
Evaluate the Impact of vSAN Object Checksum 26
Evaluate the Impact of vSAN Cache Tier SSD (SATA or NVMe) 27
Failure Testing and Failback Validation 28

Best Practices 40
Conclusion 41
About the Author 42

MongoDB on VMware vSAN 6.6 All-Flash

Executive Summary

This section covers the business case, solution overview, and key results of MongoDB on VMware vSAN 6.6 All-Flash.

Business Case

Due to the changing enterprise data landscape, unstructured databases, such as NoSQL, are needed in lieu of traditional relational databases for many applications. NoSQL databases provide advantages over relational databases regarding performance, scalability, and suitability for cloud environments. Especially, MongoDB, the popular NoSQL document store database used by thousands of enterprises. More and more customers from across the globe and diverse industries have already adopted MongoDB.

Managing storage is a distinct requirement for database applications. In traditional MongoDB deployments, MongoDB nodes use servers' local hard disks to provide data locality. Traditional shared storage systems are not treated as proper storage for MongoDB. However, this deployment approach increases the complexity of management and it cannot take advantage of the advanced features in a virtualized environment like VMware vSphere®.

VMware vSAN™, by aggregating servers' local disks into a shared storage pool, provides both high performance and data persistence. By using high-performance disks as a cache tier, vSAN ensures the performance achieved is at a desirable level. By using the Fault to Tolerance (FTT) method in the Storage Policy Based Management (SPBM) system, vSAN provides a solid data persistence method. vSAN is also integrated into the kernel of vSphere, so it becomes the best-fit storage system when deploying MongoDB in a vSphere environment.

This solution paper is intended to validate that vSAN is a HyperConverged Infrastructure for MongoDB. Users who run MongoDB in vSphere with vSAN will have the benefits of easy management, easy data protection method and a consistent level of performance.

Solution Overview

This reference architecture is a showcase of using vSAN as the storage tier for deploying MongoDB in a vSphere environment:

- We demonstrated the architecture of the deployment.
- We validated the base performance level of running MongoDB servers on vSAN.
- We compared the impact of different parameter settings on the performance level.
- We validated the resiliency tests and concluded the recovery methods.
- We provided the best practices based on performance comparison.

Key Results

The reference architecture in this paper:

- Designs the architecture of deploying MongoDB in a vSAN cluster.
- Validates predictable performance when running MongoDB in a vSAN cluster.
- Demonstrates the data protection method in a vSAN cluster to ease the management of MongoDB deployment.
- Compares the performance results of different parameter settings.
- Drives host failure and disk failure tests.
- Provides best practices guidance.

Introduction

This section provides the purpose, scope, and audience of this document.

Purpose

This reference architecture verifies the supportability and performance of running MongoDB in a vSAN cluster.

Scope

The reference architecture covers the following scenarios:

- The architecture of deploying MongoDB in a vSAN cluster
- Tuning parameters of both MongoDB and vSAN
- Performance validation of various configurations
- Failover validation of various hardware failures

Audience

This reference architecture is intended for MongoDB and other NoSQL database administrators and storage architects involved in planning, designing, or administering NoSQL databases on vSAN.

Technology Overview

This section provides an overview of the technologies used in this solution: - VMware vSphere 6.5 - VMware vSAN 6.6 - MongoDB

VMware vSphere 6.5

VMware vSphere 6.5 is the next-generation infrastructure for next-generation applications. It provides a powerful, flexible, and secure foundation for business agility that accelerates the digital transformation to cloud computing and promotes success in the digital economy.

vSphere 6.5 supports both existing and next-generation applications through its:

- Simplified customer experience for automation and management at scale
- Comprehensive built-in security for protecting data, infrastructure, and access
- Universal application platform for running any application anywhere

With vSphere 6.5, customers can run, manage, connect, and secure their applications in a common operating environment, across clouds and devices.

VMware vSAN 6.6

VMware vSAN is the industry-leading software powering HyperConverged Infrastructure (HCI) solutions. HCI, or HyperConverged Infrastructure, converges traditional IT infrastructure silos onto industry-standard servers and virtualizes physical infrastructure to help customers easily evolve their infrastructure easily without risk, improve TCO over traditional resource silos, and scale to tomorrow with support for new hardware, applications, and cloud strategies. HCI originally included just virtual compute and virtual storage but can now be extended with virtualized network resources for a fully software-defined data center.

The industry's first native HyperConverged Infrastructure (HCI) encryption solution and a highly available control plane is delivered in vSAN 6.6 to help customers evolve without risk and without sacrificing flash storage efficiencies. Operational costs are reduced with 1-click firmware and driver updates. vSAN 6.6 significant enhancements enable customers to scale to tomorrow's IT demands. See VMware vSAN 6.6 Technical Overview for details.

The performance increase benefits this solution. vSAN has a performance advantage thanks to its native, vSphere architecture. vSAN 6.6 introduces further optimizations to deliver up to 50% higher flash performance, enabling over 150K IOPS per host. This means you can run both traditional enterprise workloads more efficiently and with greater consolidation while also having the confidence to deploy new workloads like big data. Specifically, some of the performance enhancements include reduced overhead of checksum; improved dedupe and compress; destaging optimizations; object and management improvements.

MongoDB 3.4

MongoDB is a document-oriented database. The data structure is composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

The advantages of using documents are:

- Documents (for example, objects) correspond to native data types in many programming languages.
- Embedded documents and arrays reduce the need for expensive joins.
- Dynamic schema supports fluent polymorphism.

For more key features of MongoDB, refer to Introduction to MongoDB.

Replica Set

A replica set in MongoDB is a group of processes that maintain the same data set. Replica sets provide redundancy and high availability and are the basis for all production deployments.

Sharding

Sharding in MongoDB is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

Mongos

Mongos for "MongoDB Shard," is a routing service for MongoDB shard configurations that processes queries from the application layer, and determines the location of this data in the sharded cluster, in order to complete these operations. From the perspective of the application, a Mongos instance behaves identically to any other MongoDB instance.

Mongod

Mongod is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations.

Solution Configuration

This section introduces the resources and configurations: - Solution architecture - Hardware resources - Software resources - VM and database configuration - Performance test tool and settings

Solution Architecture

We created a 4-node vSphere and vSAN cluster and then deployed on the nodes MongoDB services with sharding enabled. Replica sets are either enabled or deactivated in various test configurations. The configurations and performance results are discussed in the solution validation section.

We deployed four virtual machines as Mongos servers and four virtual machines as Mongod servers as the basic architecture and performance baseline. In a production system, users can scale out both Mongos servers and Mongod servers horizontally to meet specific demands. On top of the Mongos servers is an external load balancer. We do not specify any load balancer requirements; you may choose the one you want for the client workload balancing.

In the architecture, the external load balancer is an optional component and can be omitted. In this situation, each test client should connect to one Mongos server directly.

We enabled Journal in the MongoDB configuration so that any database update can be acknowledged after the write is flushed to the disk.

In Figure 1, *ConfiDB* stands for the configuration database for MongoDB cluster's internal use. It is a mandatory component for a sharding MongoDB cluster. It stores the sharding information, Mongos states, and other information.

To eliminate the additional impact from the clients and also avoid a bottleneck, we used four test clients to operate on the database simultaneously. For example, if we require 128 threads to operate on the database simultaneously, each test client will use 32 threads, and we will aggregate the result of all the four test clients.

Figure 1 shows the solution architecture.

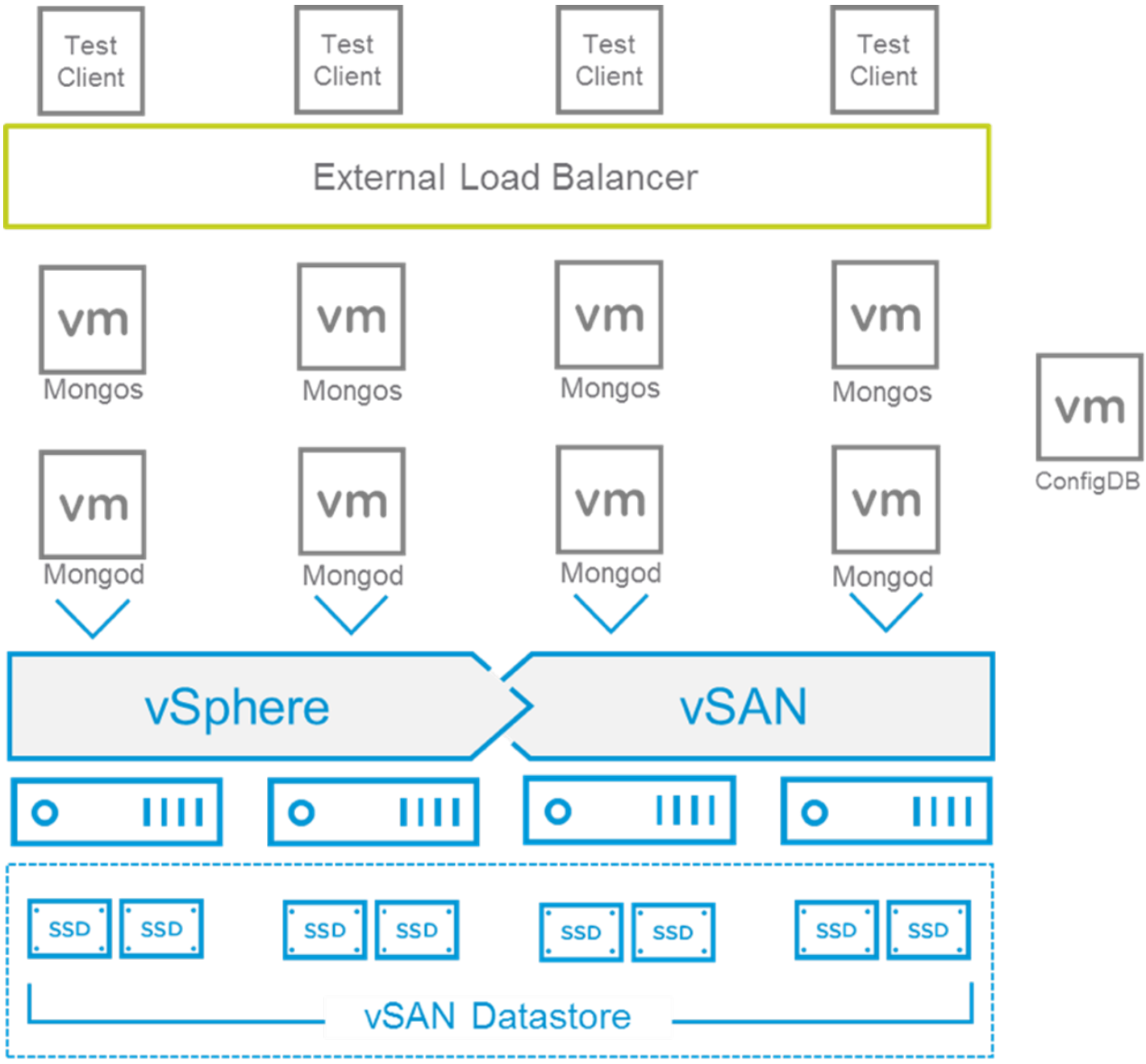

Figure 1. Solution Architecture

Hardware Resources

We used direct-attached SSDs on VMware ESXi™ servers to provide a vSAN Datastore. Each ESXi server has two disk groups each consisting of one cache tier SSD and three capacity tier SSDs.

Each ESXi Server in the vSAN Cluster has the following configuration as shown in Table 1.

Table 1. Hardware Resources per ESXi Server

Property	SPECIFICATION
Server	Supermicro SSG-2027R-AR24NV
CPU cores	2 sockets, 10 cores each of 3.0GHz with hyperthreading enabled
RAM	512GB DDR4 RDIMM
Network adapter	2 x Intel 10 Gigabit X540-AT2, + I350 1Gb Ethernet
Storage adapter	2 x 12Gbps SAS PCI-Express
Disks	SSD: 2 x 3,000GB NVMe drive as cache SSD SSD: 8 x 400GB SATA drive as capacity SSD

Software Resources

Table 2 shows the software resources used in this solution.

Table 2. Software Resources

Software	Version	Purpose
Software	Version	Purpose
VMware vCenter ® Server and ESXi	6.5.0d (vSAN 6.6 is included)	ESXi Cluster to host virtual machines and provide vSAN Cluster. VMware vCenter Server provides a centralized platform for managing VMware vSphere environments
VMware vSAN	6.6	Software-defined storage solution for hyperconverged infrastructure
CentOS	7.3	We use CentOS 7.3 as the guest operating system of all the virtual machines.

Software	Version	Purpose
MongoDB	3.4	MongoDB is an open source database that uses a documentoriented data model. MongoDB is built on an architecture of collections and documents. Documents comprise sets of key-value pairs and are the basic unit of data in MongoDB. We use the community version of MongoDB 3.4. Follow the installation instructions here.
Yahoo Cloud Serving Benchmark (YCSB)	0.12.0	YSCB is a framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores.

VM and Database Configuration

We use the virtual machine settings as the base configuration as shown in Table 3.

Table 3. VM and Database Configuration

ROLE	CPU CORES	MEMORY	OS DISK	DATA DISK
ConfigDB	8	32 GB	32 GB	200GB
Mongos	8	64 GB	32 GB	None
Mongod	8	64 GB	32 GB	200GB

The configuration in each row is for each virtual machine. The rule is that the aggregated CPU cores and memory should not exceed the physical resources. If they are overcommitted, there may be contention, which we should avoid. When calculating physical resources, we should count the physical cores before Hyper-Threading is taken into consideration.

For the ConfigDB role, MongoDB always recommends that you configure it in the replica set mode. The ConfigDB role is important as it stores the MongoDB cluster's sharding information and member's information. We configure three ConfigDB virtual machines to form a replica set.

The Mongos servers are acting as the internal routing service for MongoDB. They do not store any persistent data, so we do not configure data disks to MongoDB servers. The default OS disk is enough for configuring the Monogs service. For a Mongos server, it is CPU cores and memory bound since the routing service requires considerable CPU cores cycles. Meanwhile, a Mongos server is stateless because all stateful information is stored in the ConfigDB server. So for Mongos servers, we should properly set the CPU cores and memory capacity; however, disk capacity has no impact on performance so the setting does not matter.

In the base test configuration, we use a 100 million records database since 100 million is a medium sized database and is suitable for most common cases. In other NoSQL benchmarking references, we noticed that the configured database sizes vary from 10 million to 500 million. With taking that into consideration, we think that setting the base size to 100 million records is both realistic and reasonable for a benchmarking. In the test stage, we found that 100 million records occupy around 128GB capacity. So we configured four Mongod servers with 200GB data disk each, which should be able to hold the whole database.

MongoDB Storage Engine

The storage engine in MongoDB is responsible for managing how data is stored in memory and on disk. MongoDB supports multiple storage engines such as WiredTiger and MMAPv1. WiredTiger is the default storage engine since MongoDB version 3.2.

We also use WiredTiger as the storage engine in this reference architecture.

Performance Test Tool and Settings

YCSB is a popular Java open-source specification, and program suite developed at Yahoo! to compare the relative performance of various **NoSQL** databases. Its workloads are used in various comparative studies of NoSQL databases. We use YCSB to test the performance of MongoDB.

We used YCSB workload A and B as summarized below:

- Workload A (Update heavy workload): 50/50% mix of reads/writes
- Workload B (Read mostly workload): 95/5% mix of reads/writes

How to Run Load

According to the real world case requirements, choose a proper record count (YCSB Property 'recordcount') to insert into the database. Choose an appropriate number of threads to generate the desired workload. For the impact of operation count, the performance will eventually go into a steady state after a certain amount of time. So we should properly choose a minimal

operation count to achieve the best performance while in a steady state. The parameters can all vary due to the testbed hardware or virtual machine configurations or database sizes. The solution validation section will try to find the impact of the various parameters such as operation count or threads number.

Steps to prepare and run YCSB on test client:

1. Download the latest release of YCSB:

```
curl -O --location
https://github.com/brianfrankcooper/YCSB/releases/download/0.12.0/ycsb-0.12.0.tar.gz
tar xfvz ycsb-0.12.0.tar.gz
cd ycsb-0.12.0
```

2. Set up a database to benchmark. There is a README file under each binding directory.

3. Run the YCSB command (take workload A as an example).

```
./bin/ycsb.sh load basic -P workloads/workloada
./bin/ycsb.sh run basic -P workloads/workloada
```

Running Parameters

The following examples are the examples to run the actual workloads on MongoDB.

```
./bin/ycsb run MongoDB -s -P workloads/workloadb –p recordcount=100000000 -p operationcount=100000000 -threads 64 -p
MongoDB.url="MongoDB://ip_address_of_node:27017/test?w=1&j=1" -p readallfields=true -p requestdistribution=zipfian
```

- **thread:** the total threads that each client used to operate on the MongoDB database.
- **readallfields:** set to true to read all fields.
- **requestdistribution:** which distribution should be used to select the records to operate on – uniform, zipfian or latest (default: uniform). In our test, we choose zipfian as the distribution parameter. This setting means items in the database are selected according to popularity irrespective of insertion time; this represents social media applications where popular users have many connections, regardless of the duration of their membership.

Besides the parameters above and connection string to MongoDB router (Mongos), we use the default parameters of YCSB.

Durability Settings for the Database

In these tests, we define durability as the write being successfully written to persistent storage (disk). MongoDB uses a write-ahead log (WAL) to record operations against the database. Writes are first committed to the WAL in memory, and then the WAL is written to disk to ensure durability. We tested the following kinds of durability:

- Throughput Optimized

  In this configuration, a write operation does not require any acknowledgment. Though we can achieve best throughput with this setting, data may get lost as much as the memory size. We recognize this as a not realistic configuration in a production system.

- Balanced

  Client's write acknowledged when written to WAL in RAM, WAL is written to disk more frequently than data files. This approach allows the database to optimize writing the data files to disk, while still ensuring durability. The window of possible data loss is a function of how frequently the WAL is flushed to disk. Journal is flushed to disk every 100MB, which means we may lose up to 100MB of data.

- Durability Optimized

  The client waits for WAL or data to be written to disk. MongoDB flushes journal to disk, and there is no possibility of any data loss.

Write concern describes the level of acknowledgment requested from MongoDB for write operations to a standalone Mongod or replica sets or sharded clusters. In sharded clusters, Mongos instances will pass the write concern on to the shards. The write concern-parameters used in the test client (YCSB) are as follows:

- w: 0

Requests no acknowledgment of the write operation. However, w: 0 may return information about socket exceptions and networking errors to the application.

- w: 1

  Requests acknowledgment that the write operation has propagated to the standalone MongoDB or the primary in a replica set. w:1 is the default write concern for MongoDB.

- w: majority

  Requests acknowledgment that write operations have propagated to the majority of voting nodes, including the primary. After the write operation returns with a w: "majority" acknowledgment to the client, the client can read the result of that write with a "majority" readConcern.

- j: true

  Requests acknowledgment that the Mongod instances, as specified in the w: <value>, have written to the on-disk journal. j: true does not by itself guarantee that the write operation will not be rolled back due to replica set primary failover.

- j: false

  Requests no write acknowledgment to the on-disk journal. j:false with w:1 or w: majority means that the write operation is acknowledged by memory.

For throughput optimized durability, we use w:0 and j: false to indicate that it does not require any acknowledgment of writes. Though we can achieve best throughput with these settings, data may get lost as much as the memory size.

For balanced durability, we use w:1 and j: false to make sure write operation is acknowledged in the RAM, and we use w: majority to make sure write operation is acknowledged in memory in majority vote (primary and secondary in our designed replica set). In this setting, the worst-case data loss in case of server crash is up to 100MB.

For durability optimized durability, we use w:1 and j: true, or w: majority and j: true to make sure every write is hardened in the Journal, so no possible data loss happened in case of a server crash.

The various write concern combinations are summarized in Table 4.

Table 4. Summary of Various Write Concern Combinations

w	j	WRITE CONCERN CATEGORY	WRITE ACKNOWLEDGEMENT TYPE

Solution Validation

In this validation section, we tested the parameters impacts on the performance of MongoDB on vSAN with analyzed results.

Overview

For each test configuration settings, we always test both YCSB workload A and workload B to show the performance of different workloads.

The results were collected and analyzed with the following procedures:

- We show the operations per second (ops/sec) and the average latency for each test.
- We use four test clients to run the workloads simultaneously across all the tests. So for each test result, the threads are the aggregates threads of all the four clients. The total ops/sec is the aggregated ops/sec of all four clients and the latency is the average latency of the four test clients (add the latencies and then divide them by 4).
- If there are any abnormal results, we use vSAN Performance Service to monitor each level of performance in a top-down approach of the vSAN stack.

Baseline Performance Methodology

We tested the impacts of various performance tool settings to find the optimal settings for a comparison baseline.

The procedures included:

1. We gradually increased the threads number and looked for the best performance. At the point where increasing the number of threads degrades performance is the ideal number of threads for testing. This test is considered as a preliminary test to eliminate the side effect of the client side and focus the performance on the server side. This is a preparing stage for the actual performance test.
2. After a test is started, the performance will eventually go into a steady state. So we changed the operation count (YCSB parameter 'operationcount') to find out the best operation count for our testing. At the point where increasing the operation count does not change the performance results, we consider it steady state. This is also considered as another preparing stage for the actual performance test.
3. When the best client threads number and best operation count are found, we set the performance tests with those values as the baseline performance. Later performance tests will compare with this baseline performance result.

Performance Testing Methodology

We tested the parameter impacts on the performance of MongoDB on vSAN against the baseline performance.

We performed the following procedures:

1. We changed the CPU cores and memory of the virtual machines to show the impact of CPU cores and memory on the performance of a MongoDB cluster running on vSphere and vSAN.
2. We changed the database size to show its impact if we fix the CPU cores and memory of the virtual machines.
3. When business grows and database size expands, the MongoDB cluster should scale out to expand the storage capacity. We evaluated the performance when MongoDB data nodes grows from 2-nodes to 4-nodes to 8-nodes.
4. In the baseline testing, we set vSAN's FTT value to 1. With FTT=1, we consider that vSAN provides the data protection for the MongoDB cluster, so we do not enable replica set on MongoDB's data nodes. However, some customers still want MongoDB's application level replication. So in this test case, we enabled MongoDB's replica set on the data nodes to compare with the baseline test. In this situation, MongoDB's replica set provides application-level data protection and vSAN provides storage level data protection.
5. We changed the vSAN object stripe width in SPBM to see the impact on performance with different stripe width.
6. We changed the vSAN failure tolerance method and failure number (RAID 1 or RAID 5) in SPBM to see the impact on performance with different failure tolerance method.
7. We changed the durability settings to measure the impact on performance with different 'w' and 'j' option combinations.
8. We enabled or deactivated the vSAN checksum in SPBM to see the impact on performance with checksum configuration.

Note: vSAN checksum is end-to-end software feature to avoid data integrity issues arising due to potential problems on the underlying storage media. We highly recommend using the default setting which is always enabled. The only reason to deactivate it is that the application already has this functionality included.

9. We switched the vSAN cache tier disks from NVMe SSD to SATA SSD to show the impact of different hardware equipment.

Evaluate the Impact of Different Client Threads

Test Overview

MongoDB performance test used MongoDB v3.4 running on CentOS virtual machines, which were stressed by YCSB workload A and workload B. The MongoDB's virtual machines were deployed on the 4-node all-flash vSAN cluster. In the baseline test, there were four Monogs servers and four Mongod servers. To maintain a balanced workload, each physical server holds a Mongos server and a Monogd server. MongoDB's ConfigDB virtual machines were randomly placed in the cluster.

For the baseline test, we deactivated checksum in the applied storage policy to the virtual disks for the database. In later tests, we will evaluate the impact of checksum.

The 100GB MongoDB database was split into four shards, with each node having one shard.

For most enterprise mission-critical applications, we think that data is important and in most cases, data should not get lost when failures occur. Therefore, we set the durability level as "Durability Optimized". In this configuration, the 'w' option is set to 1 and the 'j' option is set to 1, so the ops/sec is relatively low and the latency is relatively high. We think this is a reasonable tradeoff as the data is protected at the highest level. Data is always acknowledged to disk, so a node failure like power shortage will not cause a data loss.

We used four clients and set the aggregated threads of the YCSB starting from 32, then 64, 96, 128, 160, 192, 224 and up to 256 on every test client to increase the workloads gradually. This test helped us find the best threads number suitable for this testbed and this baseline configuration.

Test Results

For both workload A and workload B, we collected the ops/sec value, average read latency, and average update latency.

Workload A

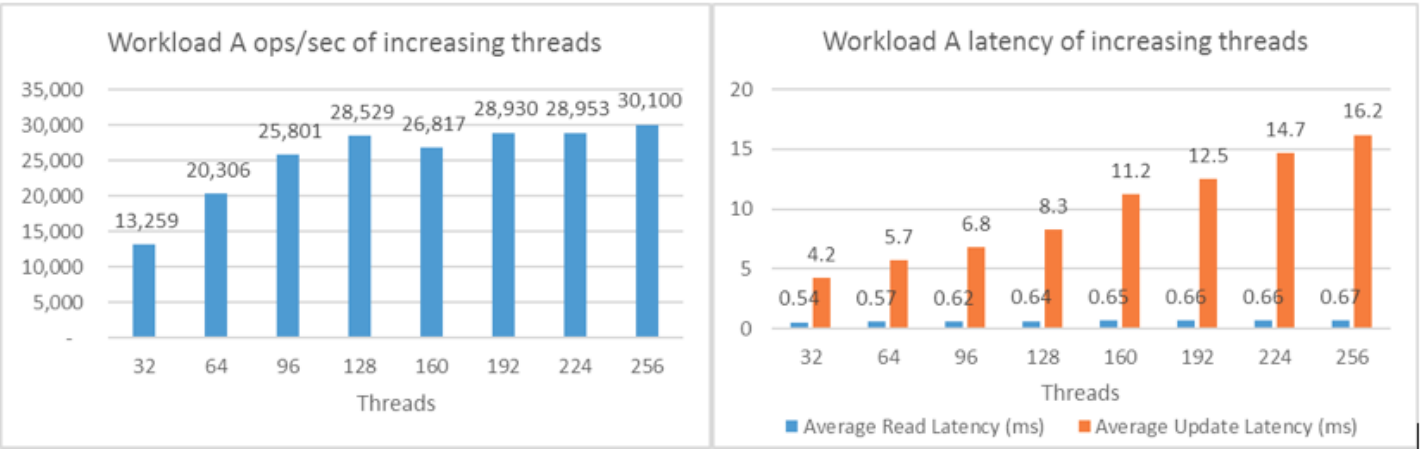

Figure 2. Workload A Test Results with Different Client Threads

For workload A, both the ops/sec and latency increased as the client threads number increased. When the threads number was under 128, the ops/sec increased rapidly. But after the threads number exceeded 128, the ops/sec value went into a steady state. There were some fluctuations in performance but the performance no longer increased dramatically. So we think in this testbed and with the above database configuration, using 128 client threads leads to a maximum performance and keeping the latency lower than that with higher threads.

Workload B

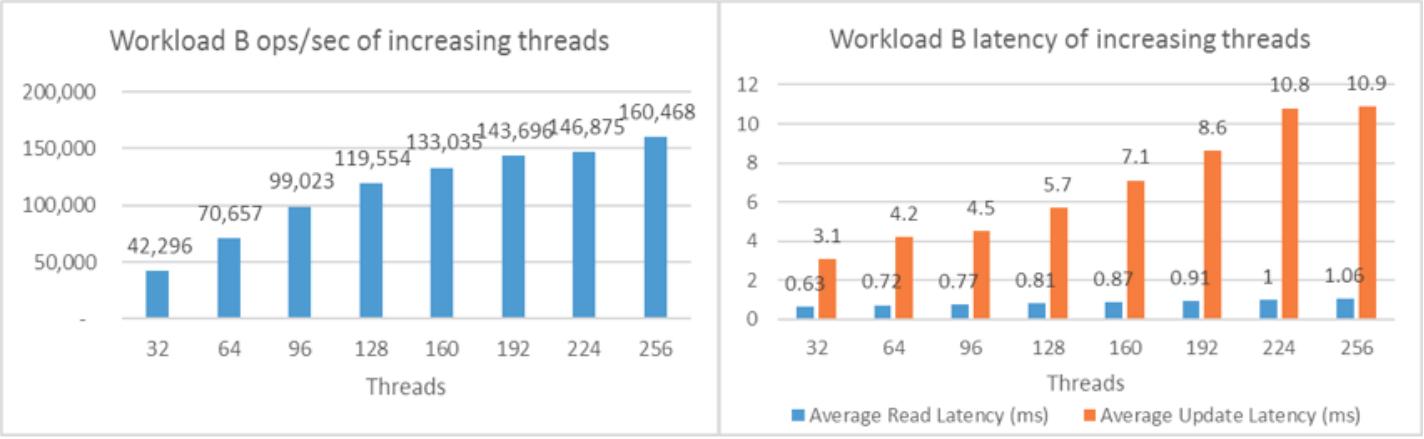

Figure 3. Workload B Test Results with Different Client Threads

For workload B, the observation was slightly different. Using 128 client threads achieved an ops/sec of 119,554. Read latency was 0.81ms and update latency was 5.7ms. As client threads grew, both ops/sec and latency increased. Update latency exceeded 10ms after client threads exceeded 192. Comparing the results of 128 and 192 client threads, ops/sec increased from 119,554 to 143,696, a 20.1% increase. Meanwhile, update latency increased from 5.7ms to 8.6ms, a 50.1% increase. So in order to keep latency relatively low and to keep consistent with workload A, we also used 128 threads to test against workload B for the rest of the tests.

Based on these test results, we used 128 client threads across the remaining tests for both workload A and workload B for consistency.

Evaluate the Impact of Different Operation Count

Test Overview

Following the tests above for the best-suited threads with maximum threads and a relatively low latency, we also wanted to find the best-suited operation count (YCSB parameter 'operationcount') to eliminate the effect of client-side operation count on performance and keep it consistent across the remaining tests.

The principle behind the scene is that when a test lasts for a long time, the performance will eventually enter a steady state. Ops/sec and average latency will go into a steady level and will not vary very much. If the performance already goes into a steady state, further operations will not change the overall performance for the benchmarking purpose.

We used 128 threads as explained before and kept the other parameters unchanged.

Test Results

For both workload A and workload B, we collected the ops/sec value, average read latency, and average update latency.

In Figure 4 and Figure 5, 'M' stands for 'million," so 5M means 5 million.

Workload A

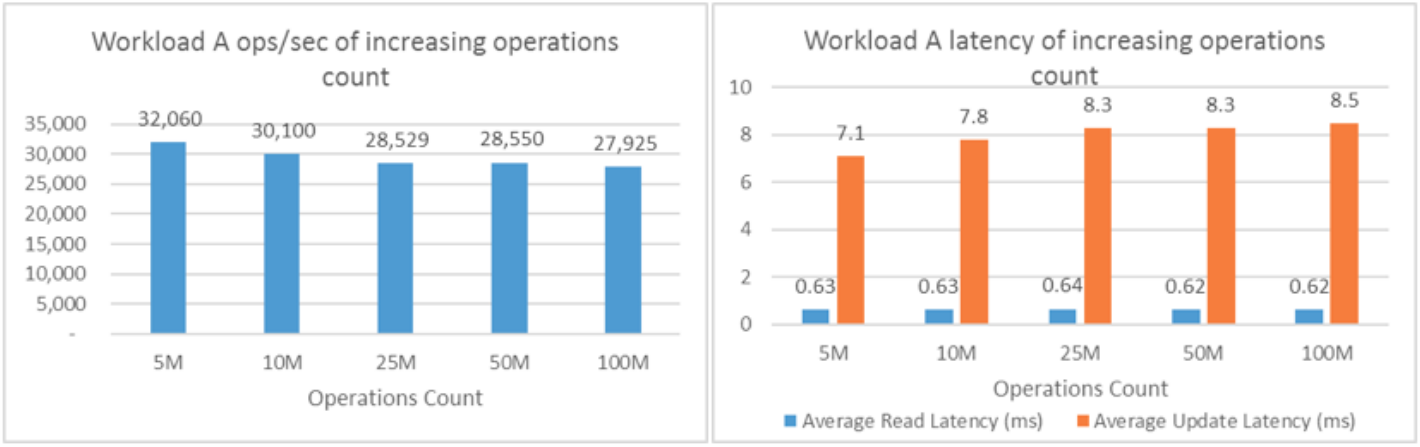

Figure 4. Workload A Test Results with Different Operation Count

For workload A, we observed there was just a slight variation in ops/sec and latency. The result revealed that when the operation

count increased from 5M to 25M, the ops/sec dropped and the latency increased. Any increase in operation count over 25M did not affect performance further. The ops/sec and latency values were both steady after operation count exceeded 25M. So we conclude that the performance will go into a steady state when the operation count is around 25M for workload A.

Workload B

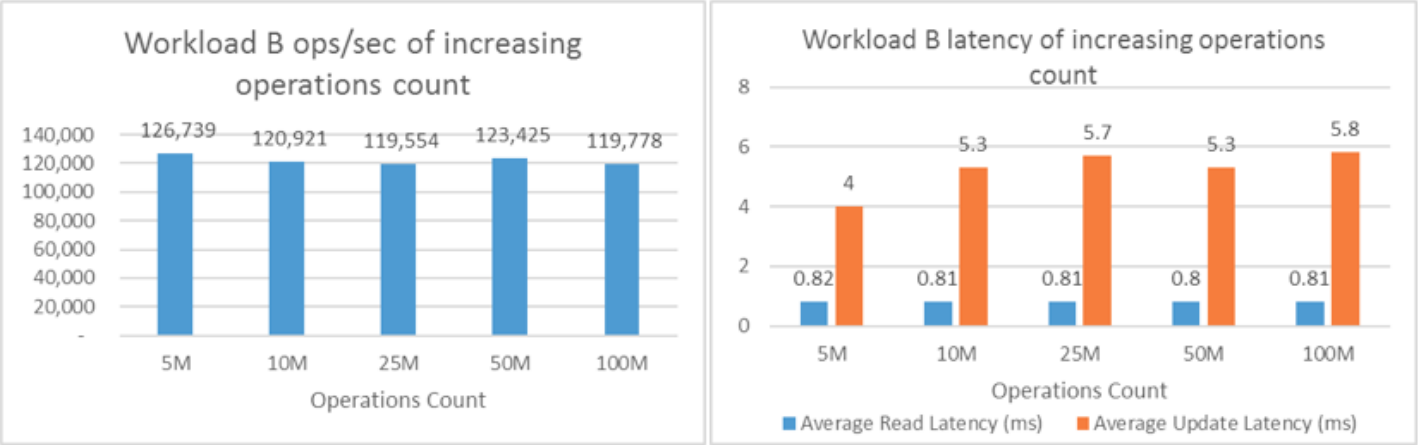

Figure 5. Workload B Test Results with Different Operation Count

For workload B, we observed a similar pattern with workload A. As the operation count increased, the performance dropped slightly and then it remained at a steady level. For both ops/sec and latency, they remained steady when operation count exceeded 10M. To keep consistent with workload A, we also set the operation count to 25M for workload B.

By combining the above results of workload A and workload B, we used 25M as the operation count in the following tests to achieve the best results and keep consistent.

Parameter Settings in the Baseline Testing

In combination with the above threads, operation counts, and overall architecture settings, we summarized the detailed parameter settings as shown in Table 5.

Table 5. Parameter Settings in the Baseline Testing

PARAMETER	VALUE
Monogd data server number	4
Mongod data disk size	200GB
Mongod CPU cores number	8
Mongod memory size	64GB
Mongos CPU cores number	8
Mongos memory size	64GB
Enable Mongod replica set?	No
vSAN stripe width setting	1

PARAMETER	VALUE
vSAN FTT	1
vSAN object checksum	Disable d
YCSB client threads	128
Database entry size	100 million
Operation count	25 million
MongoDB durability 'w' option	1 (true)
MongoDB durability 'j' option	1 (true)

The following tests used these settings as the baseline configuration. We modified these parameters in the subsequent tests.

In the baseline testing:

- Workload A, the ops/sec value was 28,529 with average read latency 0.64ms and average update latency 8.3ms.
- Workload B, the ops/sec value was 119,554 with average read latency 0.81ms and average update latency 5.7ms.

NOTE: This is not a best-case configuration. It is just a baseline configuration, so we have performance result that we use to compare with. There is no optimization in this baseline configuration.

Evaluate the Impact of Different Virtual CPU Cores and Memory Configurations

Test Overview

With the above tests, we used 8 CPU cores and 64GB memory for the MongoDB servers. For a specific database size, more CPU cores means it is faster to serve the data and more memory means more data is cached in memory. So both CPU cores and memory settings can affect the performance. In this test, we changed the CPU cores and memory size for the Mongod data servers to evaluate their impacts.

In Figure 6, '8C+32GB' means it is configured with 8 CPU cores and 32GB memory for each Mongod data server.

Workload A

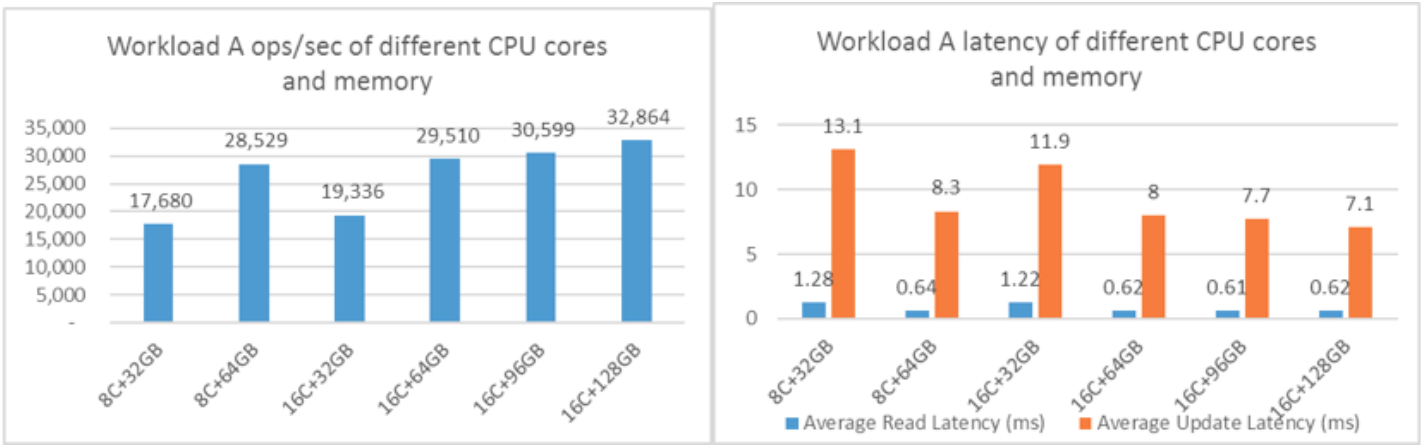

Figure 6. Workload A Test Results with Different CPU Cores and Memory

For workload A, we fixed the memory with 32GB and observed the results with CPU increasing from 8 to 16. Ops/sec increased from 17,680 to 19,336, which was a 9.3% increase. With 64GB memory and CPU increasing, ops/sec increased from 28,529 to 29,510, a 3.4% increase.

Then we compared the results when CPU was fixed and memory size increased. With 8 CPU cores and memory increased from 32GB to 64GB, ops/sec increased from 17,680 to 28,529, which was a 61.3% increase. With 16 CPU cores and memory increased from 32GB to 64GB to 96GB to 128GB, ops/sec increased from 19,336 to 29,510 to 30,599 to 32,864, which was 52.6%, 58.2% and 69.9% increase respectively.

For the above tests, average read and update latency decreased when ops/sec increased as shown in the right part of Figure 6.

Workload B

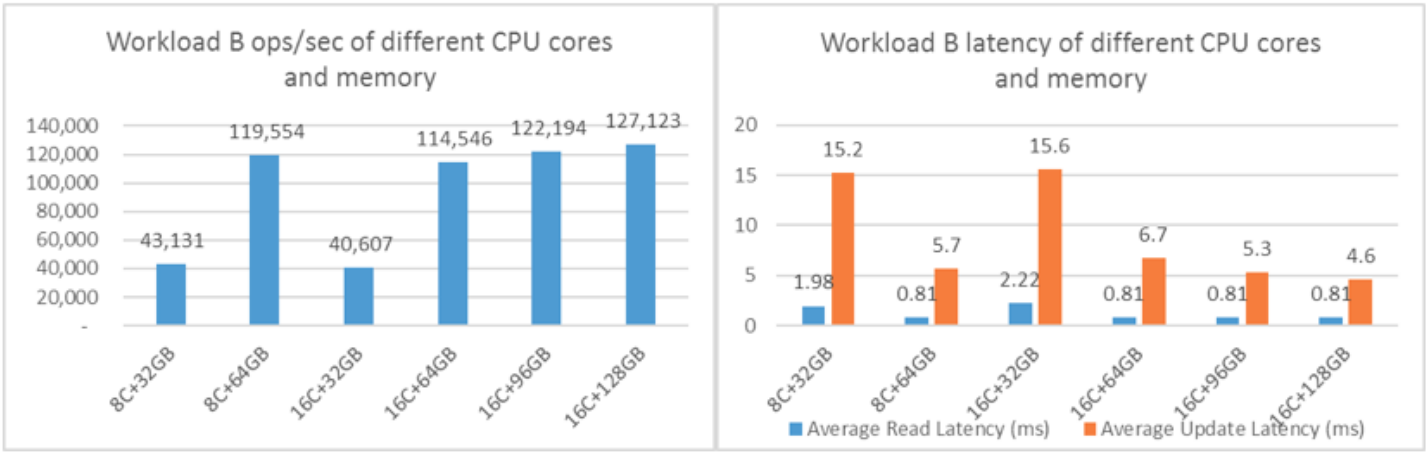

Figure 7. Workload B Test Results with Different CPU Cores and Memory

For workload B, we fixed the memory with 32GB and observed the results with CPU increasing from 8 to 16. Ops/sec decreased from 43,131 to 40,607, which was a 5.8% decrease. With 64GB memory and CPU increasing, ops/sec increased from 119,554 to 114,546, a 4.2% decrease.

Then we compared the results when CPU was fixed and memory size increased. With 8 CPU cores and memory increased from 32GB to 64GB, ops/sec increased from 43,131 to 119,554, which was a 177% increase. With the increase of 16 CPU cores and memory from 32GB to 64GB to 96GB to 128GB, ops/sec increased from 40,607 to 114,546 to 122,194 to 127,123, which was 182%, 200% and 213% increase respectively. From the right part of Figure 7, we observed that the latency was not affected with the increase of CPU and it was reduced with the increase of memory size. Thus, for workload A and workload B, increasing memory size helped increase performance more effectively than increasing CPU.

So increasing both CPU and memory can improve performance. Increasing memory is a more effective way to improve performance. In a production system, we recommend using a proper memory size and number of CPU cores for users' specific database size to achieve better performance.

Evaluate the Impact of Different Database Size

Test Overview

With the above tests, we used 8 CPU cores and 64GB memory for the MongoDB servers. We already observed the impact of different CPU cores and memory on performance. In this test, we fixed the CPU cores and memory as in the baseline testing and evaluated the impact of different database size.

We used 100 million entries in the baseline test. In this test, we used 50 million, 200 million, and 500 million respectively to show the system was either underutilized or overloaded.

In Figure 8 and Figure 9, 'M' stands for 'million' so 50M means 50 million.

Workload A

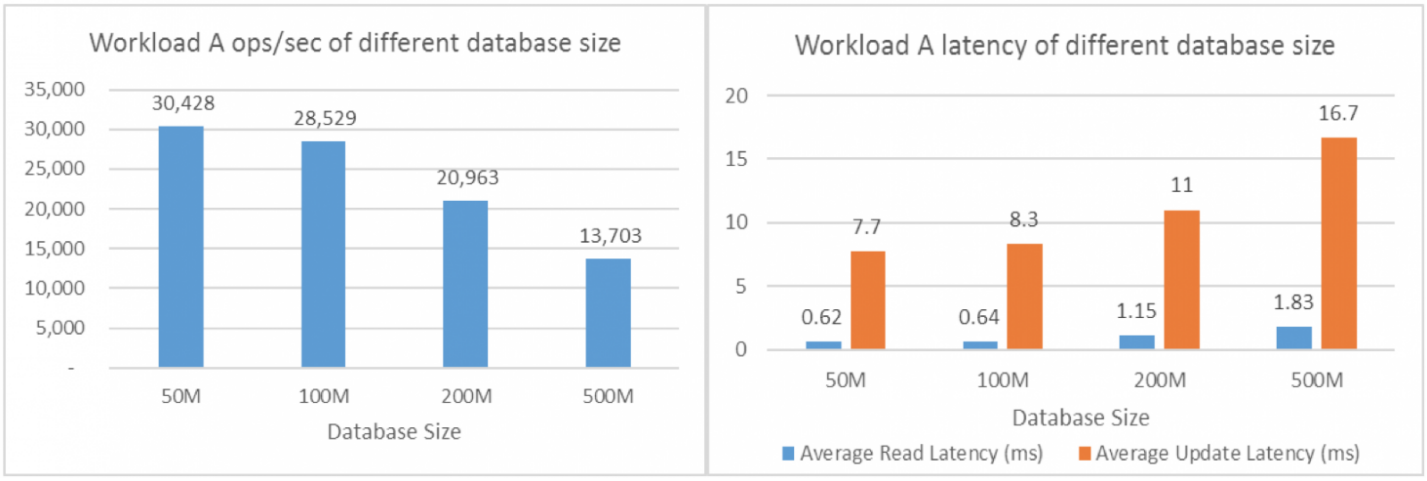

Figure 8. Workload A Test Results with Different Database Size

For workload A, the ops/sec decreased as database size increased. The latency increased as database size increased. We used a fixed size of CPU and memory, so when database size grew, the memory used grew and cache would eventually not be enough. More and more data cannot be cached in memory and must be flushed to disk. So ops/sec decreased and latency increased.

In this test, 8 CPU cores and 64GB memory can hold up to a database size of 100 million entries. If the database is larger than 100 million, we recommend increasing memory to achieve better performance.

Workload B

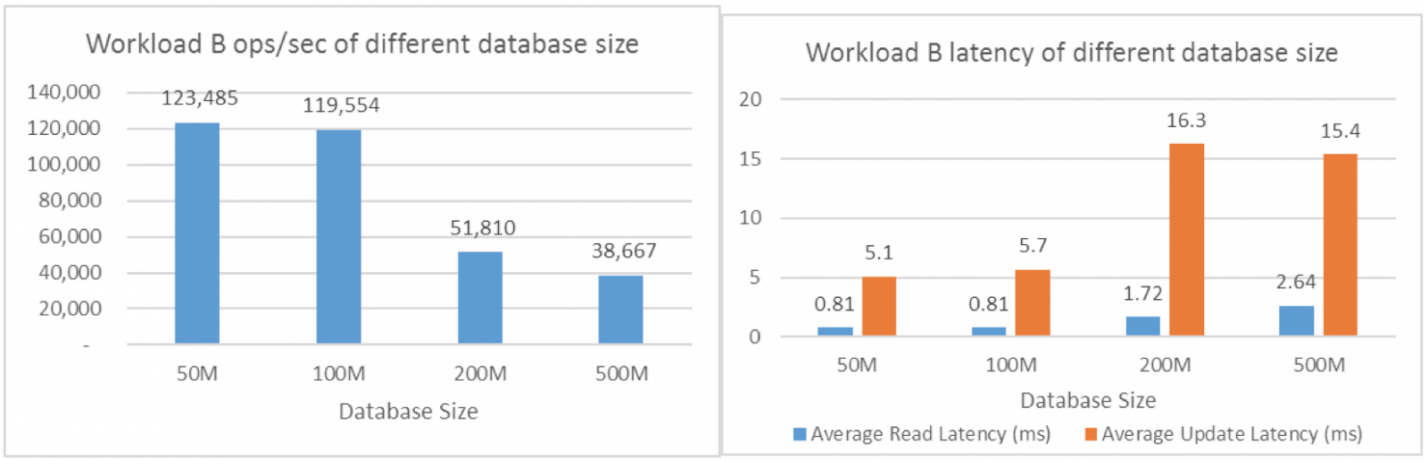

Figure 9. Workload A Test Results with Different Database Size

For workload B, we observed a similar pattern. The ops/sec dramatically decreased when database size was over 100 million and the latency dramatically increased. The reason was the same as that for workload A. In this test, 8 CPU cores and 64GB memory could hold up to a database size of 100 million entries. If the database is larger than 100 million, we recommend increasing CPU and memory to avoid a performance penalty.

So in a MongoDB deployment, we should properly calculate the database size, CPU cores and memory for better throughput and avoid high latency.

Evaluate the Impact of MongoDB Cluster Scaling out

Test Overview

When a business grows, customers might need to scale out a MongoDB cluster horizontally. Scaling out means more than one aspect:

1. Firstly, the number of mongod nodes scales out. That means more computation power and more storage capacity is added to the cluster.
2. Secondly, when a MongoDB cluster scales out the database size should scale out accordingly.
3. Thirdly, when a MongoDB cluster scales out, it should support more clients' connection.

Based on the demands, we expect that the throughput should increase while keeping the latency at a relatively steady level when the MongoDB cluster scales out.

We used the following nodes number, database size, and client threads configuration as shown in Table 6.

Table 6. Database Size and Client Threads Configuration for Different Number of Nodes

NUMBER OF NODES	DATABASE SIZE	CLIENT THREADS
2	50 million entries	64
4	100 million entries	128
8	200 million entries	256

In Table 6, 2-node means that there were two virtual machines acting as the mongod servers. We kept mongos servers and ConfigDB servers unchanged to focus on capacity scaling out.

Each node has the same CPU and memory capacity. So as nodes scale out, CPU and memory expand accordingly.

Workload A

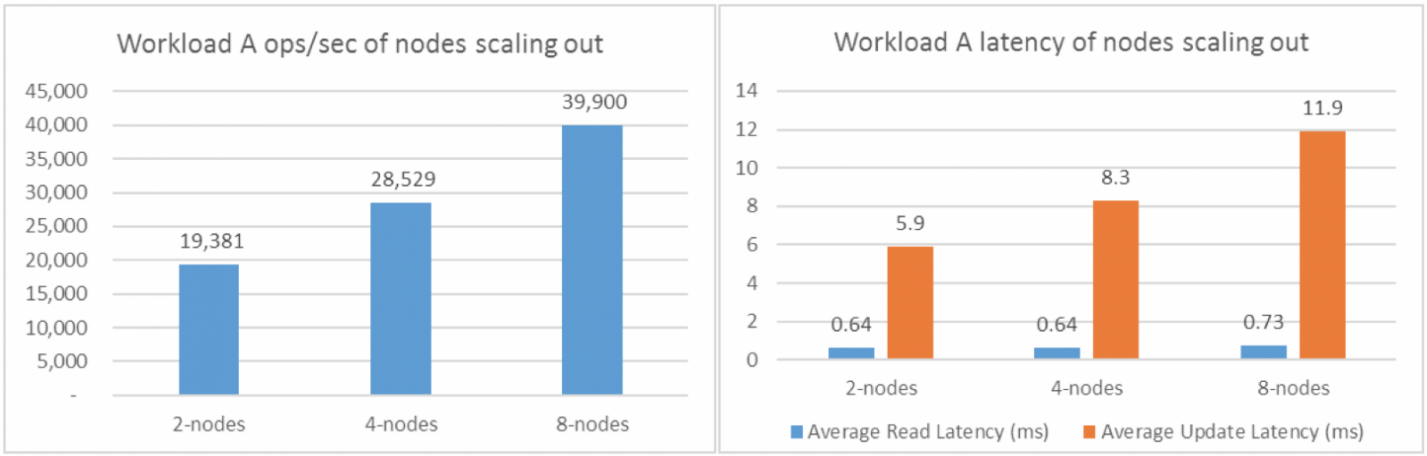

Figure 10. Workload A Test Results with MongoDB Nodes Scaling out

Workload B

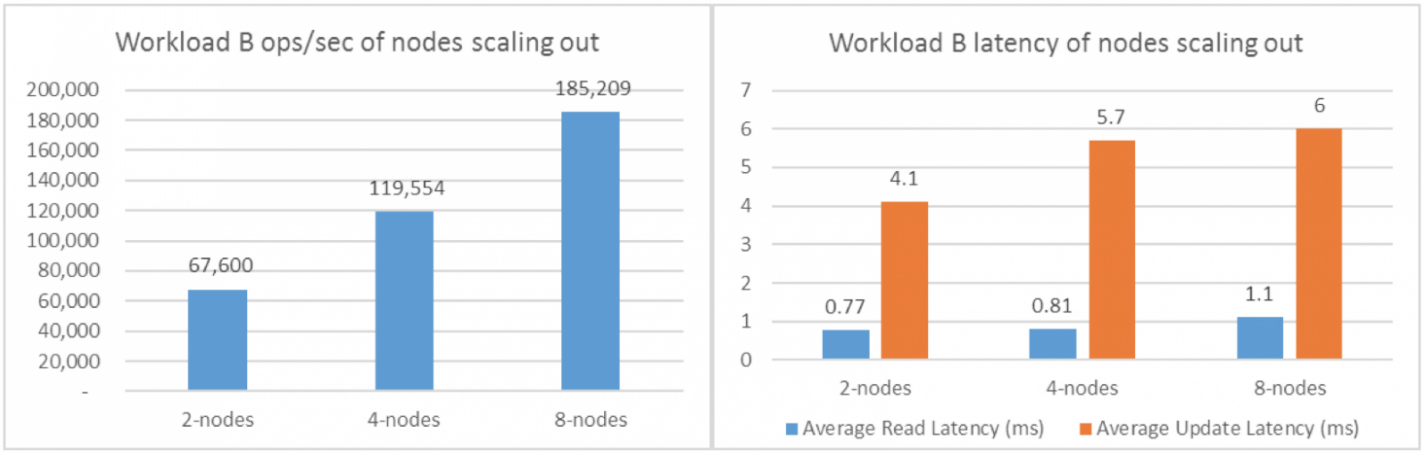

Figure 11. Workload B Test Results with MongoDB Nodes Scaling out

For both workload A and workload B, ops/sec increased as MongoDB nodes scaled out. Both read latency and update latency increased while update latency increased more sharply. For workload A, from 2-nodes to 8-nodes, the update latency increased from 5.9ms to 11.9ms. For workload B, from 2 nodes to 8 nodes, the update latency increased slightly from 4.1ms to 6ms. Comparing workload A with workload B, the update latency increased more than that with workload B. So scaling out MongoDB can

improve the throughput with acceptable latency impact. Meanwhile, scaling out MongoDB favors 'read' more.

Evaluate the Impact of MongoDB Replica Set Setting

Test Overview

Without a storage level fault tolerance mechanism, MongoDB recommends using replica sets for data nodes to recover when a node fails. In our baseline testing, we used only one virtual machine in a replica set due to the fact vSAN provides storage level protection. The application level replication and storage level protection complement each other and they have different benefits, such as different recovery time. We provide the performance evaluation when a replica set contains two data nodes and one arbiter node to test the scenario of enabling both application level replication and storage level protection.

We use the term 'rs=1' to indicate that there is no application level replication. In each replica set, there is just one virtual machine acting as the data node. In this situation, we rely on vSAN to provide data protection.

We use the term 'rs=3' to indicate that there is an application level replication. In each replica set, there are two virtual machines acting as data nodes and one virtual machine acting as the arbiter. Data will be replicated from the primary data node to the secondary data node. In this situation, vSAN provides storage level data protection and the replica set provides application level data replication.

With 'rs=3', there are two write duration options. When the 'w' option is set to 1, write is acknowledged by the primary data node and is replicated to the secondary data node asynchronously. When the 'w' option is set to 'majority', the majority of the data nodes acknowledges write. In our case, 'w=majority' equals 'w=2'. Both of the data nodes should acknowledge write, so update latency is relatively high in this case. 'w=majority' provides the highest level of data protection while performance might be affected.

Figure 12 and Figure 13 show the performance differences of the different setting combinations: rs=1; rs=3 and w=1; rs=3 and w=majority.

Workload A

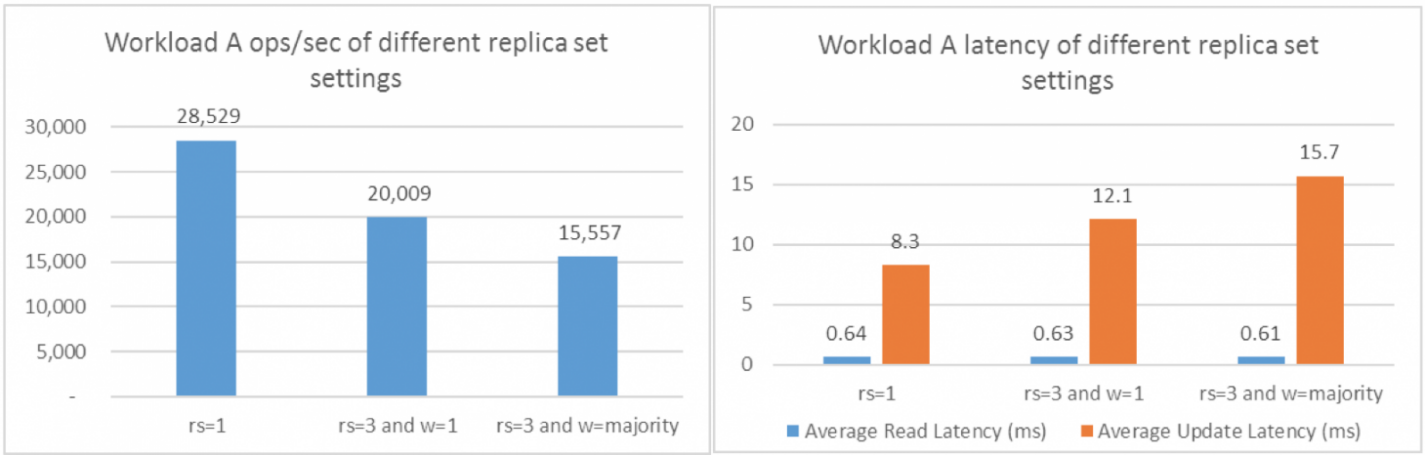

Figure 12. Workload A Test Results with Different MongoDB Replica Set Setting

Workload B

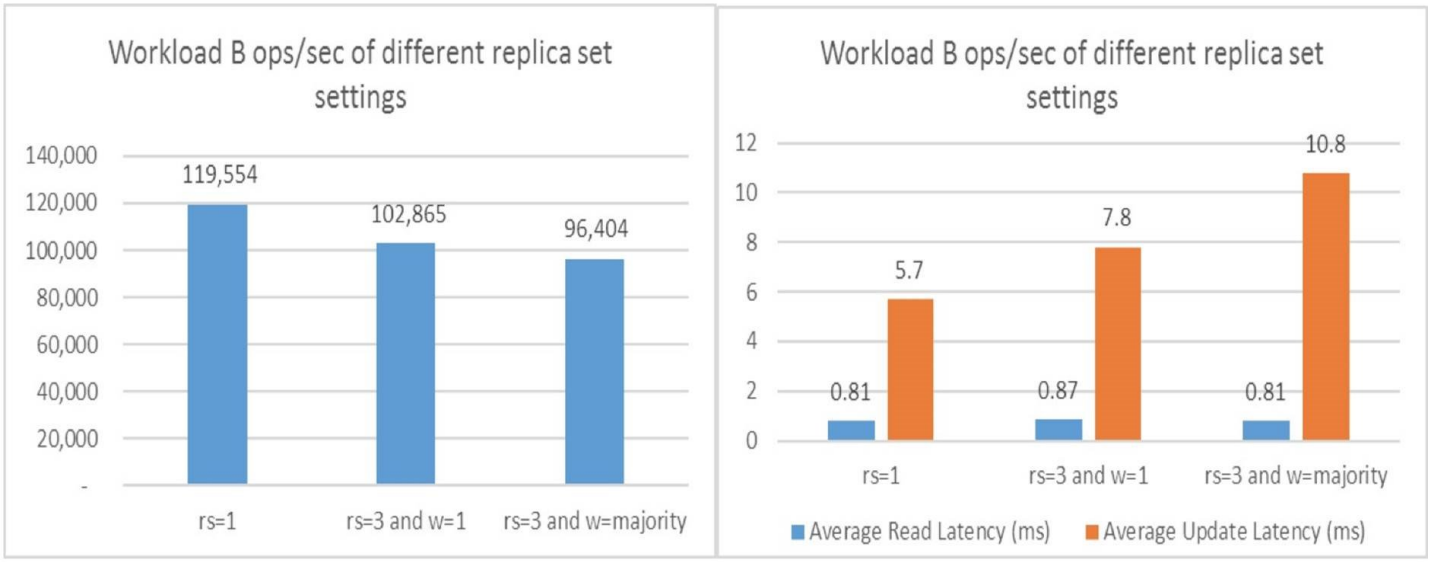

Figure 13. Workload B Test Results with Different MongoDB Replica Set Setting

For both workload A and workload B, ops/sec decreased and latency increased as MongoDB replica set changed from 1 to 3. Although the latency increased, it was still within an acceptable level of around 10ms. That was an expected result, as more nodes in a replica set require more write acknowledgments, the ops/sec value decreased. Besides, the update latency was higher with 'w=majority' than that with 'w=1'. This was also expected as 'w=majority' requires two write acknowledgments.

Evaluate the Impact of Different vSAN Object Stripe Width

Test Overview

vSAN leverages SPBM to provision and manage the objects on a vSAN datastore. By increasing the stripe width on an object, the object is divided into more components. These components will be guaranteed to distribute on different capacity disks. Thus an object will leverage more disk to serve the read operation and potentially use more cache disks to improve performance.

In the baseline testing, we set the stripe width of all the Mongod data servers to 1. In this testing, we changed the stripe width from 1 to 2, 4, 6, 8, and 12 to show the impact of different vSAN object stripe width.

Workload A

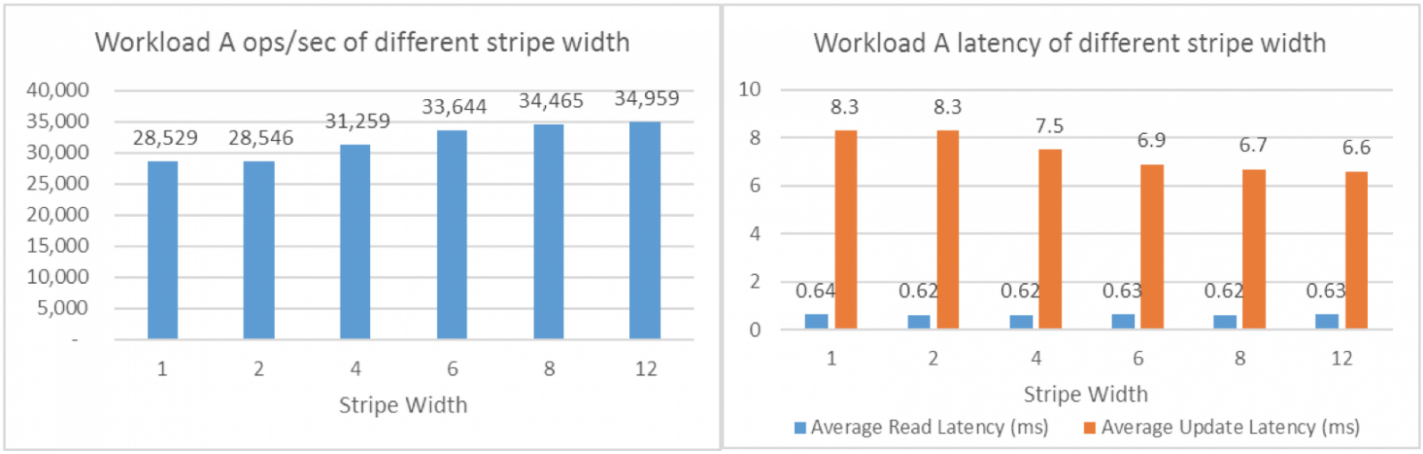

Figure 14. Workload A Test Results with Different Stripe Width

For workload A, the ops/sec increased with stripe width increasing until 12. It achieved the best performance with the stripe width setting to 12. The latency showed a similar trend.

Workload B

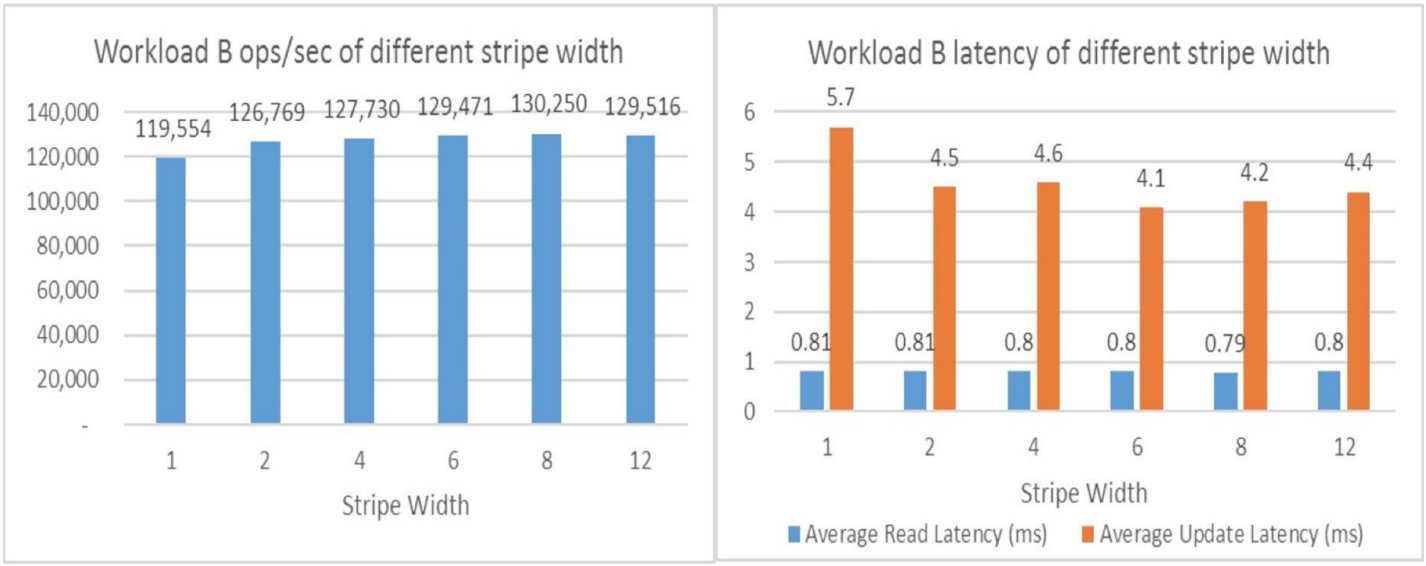

Figure 15. Workload B Test Results with Different Stripe Width

For workload B, we observed different results as workload A. The ops/sec increased with stripe width increasing until 8. It achieved the best performance with the stripe width setting to 8. The ops/sec dropped when the stripe width further increased to 12. The latency showed a similar trend. It dropped as stripe width increased to 8 and increased when the stripe width further increased to 12. So increasing the vSAN object stripe width will potentially increase performance. Users who deploy MongoDB servers on vSAN should try to increase the stripe width if they observe a relatively low performance.

NOTE: In this test, we achieved a maximum performance with stripe width setting to 12 for workload A and 8 for workload B. However, this does not mean we should set it to 8 or 12 in every MongoDB running on a vSAN environment. This is testbed and software configuration dependent. For each environment, there should be a best-suited stripe width number.

Evaluate the Impact of Different vSAN Failure Tolerance Method

Test Overview

vSAN leverages SPBM to provision and manage the objects on a vSAN datastore. In SPBM settings, there is a configuration item called FTT. Users can also set the failure tolerance method, either RAID 1 or RAID 5/6. RAID 1 is also known as mirrored. Each data is stored identically on two hosts. RAID 1 occupies two times storage while provides better performance. RAID 5 uses 1.33 times storage and RAID 6 uses 1.5 times storage. Both RAID 5 and RAID 6 can save some storage space but receive a relatively low performance compared to RAID 1. RAID 5 requires at least four hosts and RAID 6 requires at least six hosts.

In the baseline testing, we used the FTT method of RAID 1. In this test, we changed the FTT method of the mongod data nodes to RAID 5. The purpose of this test is to evaluate the tradeoff between performance and storage efficiency.

Workload A

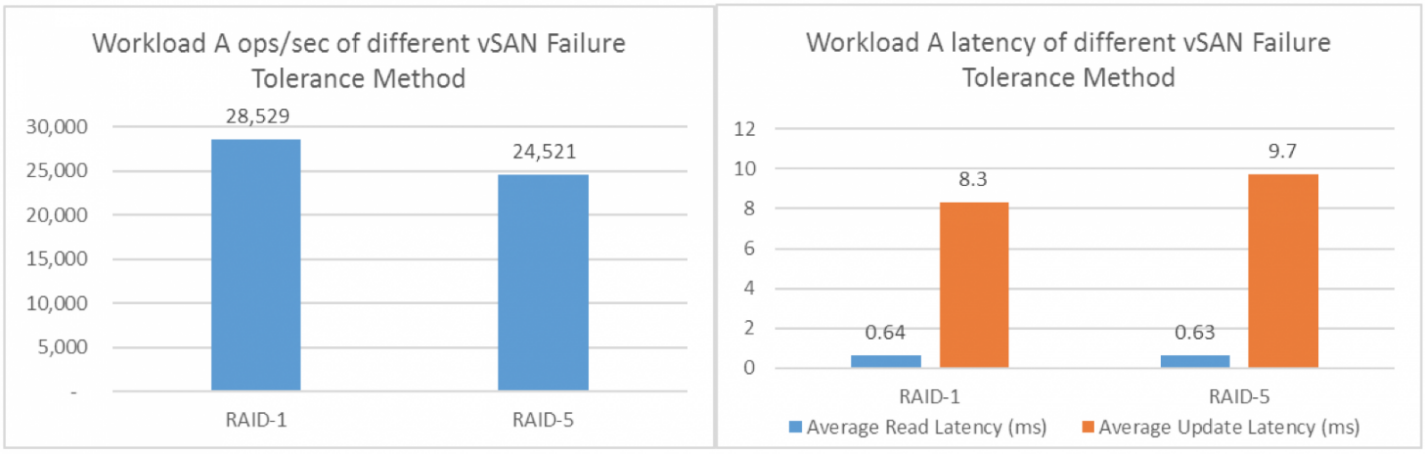

Figure 16. Workload A Test Results with Different Fault Tolerance Methods

For workload A, the ops/sec decreased as the failure tolerance method changed from RAID 1 to RAID 5 as expected. Read latency decreased while update latency increased.

Workload B

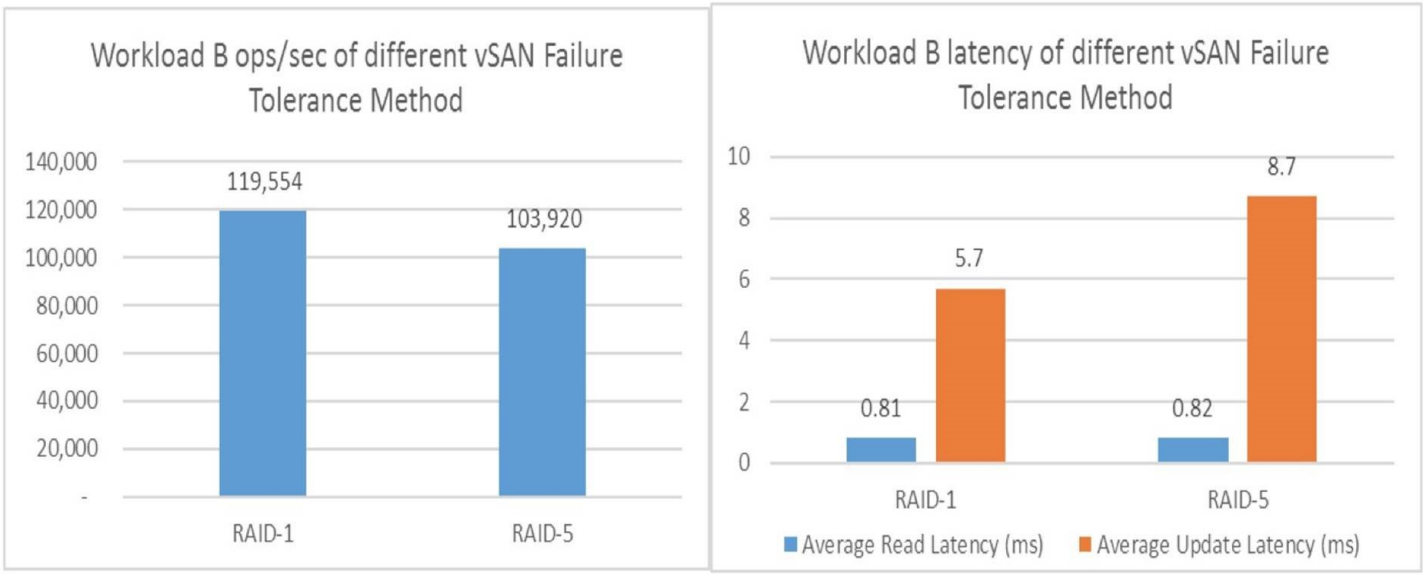

Figure 17. Workload B Test Results with Different Fault Tolerance Methods

For workload B, we observed a similar pattern as workload A.

The test results showed that using RAID 5 could save storage space but also would lead to a lower throughput and higher latency. Users should consider the tradeoff between storage space and performance.

Evaluate the Impact of Different Durability Options

Test Overview

In MongoDB's data persistent model, there are different 'w' option and 'j' option combinations that form different durability levels. If 'w' and 'j' are both set to 1, which is the baseline in this reference architecture and data is acknowledged by the journal on disk so no data would be lost in case of a failure. Use this configuration if you do not want to lose any data.

However, in some cases, users may want higher performance. Thus, they can accept the loss of some latest written data. If we set 'w' to 1 and set 'j' to 0, all written data is first acknowledged in memory and MongoDB will flush to data to journal on disk in a periodic manner. The period is roughly by 100MB data size. So if a failure occurs, 100MB data might get lost. If we set 'w' to '0' and 'j' to 0, there is no acknowledgment and all data in memory can be lost. In this configuration, the potential data loss size is based on the memory size.

All the configurations will have influence on performance. There is a tradeoff between performance and potential data loss level is based on user requirements.

The 'w' and 'j' options are specified in the MongoDB connection URL. For detailed information, see MongoDB document Connection String URI Format.

We used three combinations: 'w=1 and j=1'; 'w=1 and j=0'; and 'w=0 and j=0' to evaluate the impact of different durability options.

Workload A

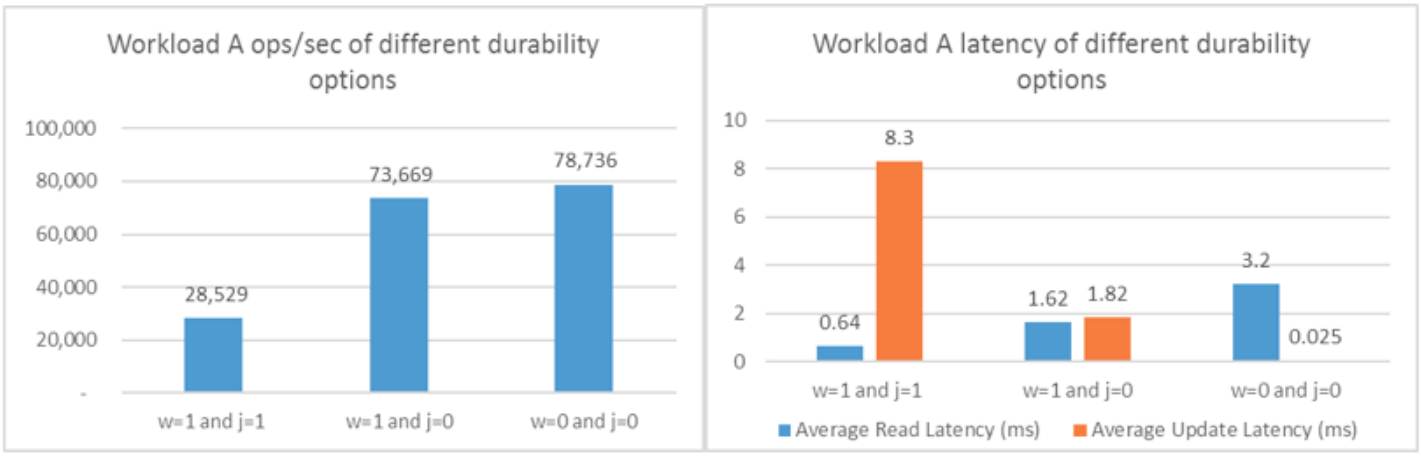

Figure 18. Workload A Test Results with Different Durability Combinations

As we can see, with 'w' and 'j' options changing from 'w=1 and j=1' to 'w=1 and j=0', and 'w=0 and j=0', the ops/sec increased. The read latency increased as there was more written data stored in memory waiting to flush to disk. So the read cache in memory would be less so that some data was fetched from disk. The update latency decreased as the data was not written to the disk but was just written to the memory, so the update latency decreased. Specifically, for 'w=0 and j=0', data was also written to memory but no acknowledgement was needed, so the update latency was nearly negligible 0.025ms.

Workload B

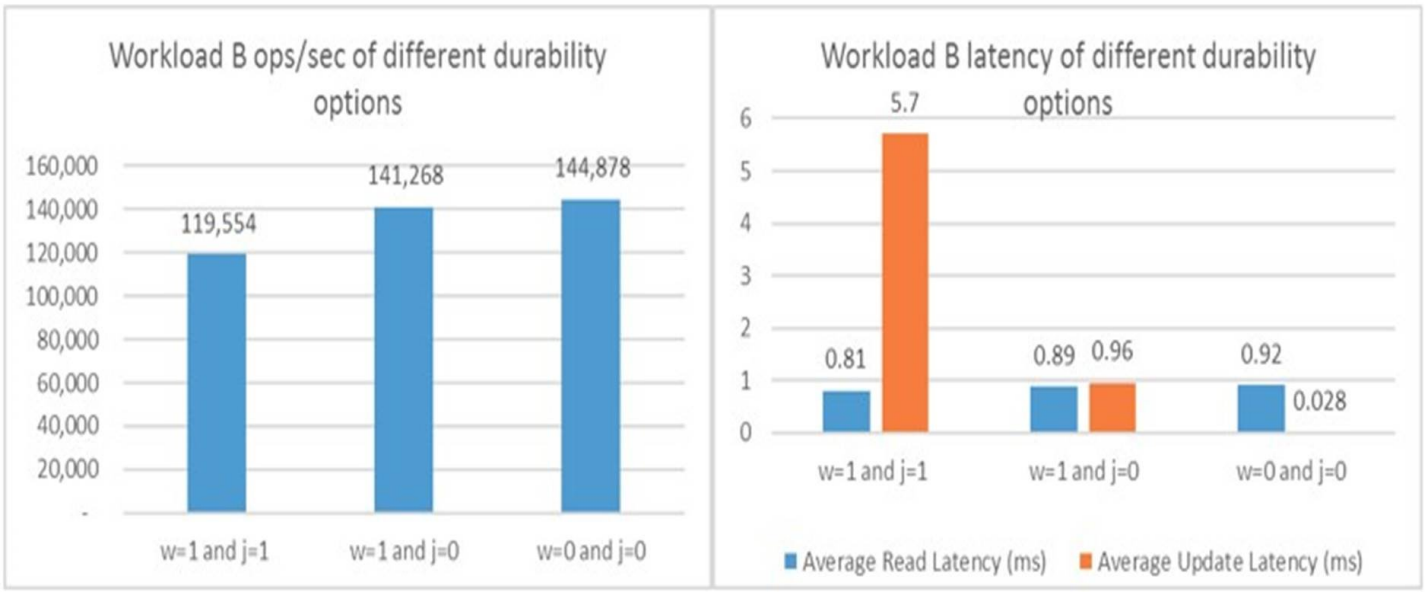

Figure 19. Workload B Test Results with Different Durability Combinations

For workload B, we observed the same trend as workload A. The ops/sec and read latency increased while the update latency decreased. However, the ops/sec in workload B test did not increase the same percentage as that in workload A. That was because workload A was 50% read and 50% write while workload B was 95% read and 5% write. The durability options were related to write acknowledgment so they mainly had impacts on the throughput of writes. They had more impacts on workload A than workload B.

Again, users should consider the tradeoffs between performance and data durability and choose the proper 'w' and 'j' options on demand.

Evaluate the Impact of vSAN Object Checksum

Test Overview

vSAN leverages SPBM to provision and manage the objects on a vSAN datastore. There is a field in a policy called object checksum. By enabling checksum, vSAN calculates the checksum of each I/O block. This ensures that there is no bit flip on the disk or any other hardware failure that might corrupt a block of data.

In the baseline testing, we set the 'checksum' field of all the Mongod data servers to 'Disabled'. In this testing, we enabled

checksum to show the impact of vSAN's checksum setting on MongoDB's performance.

Workload A

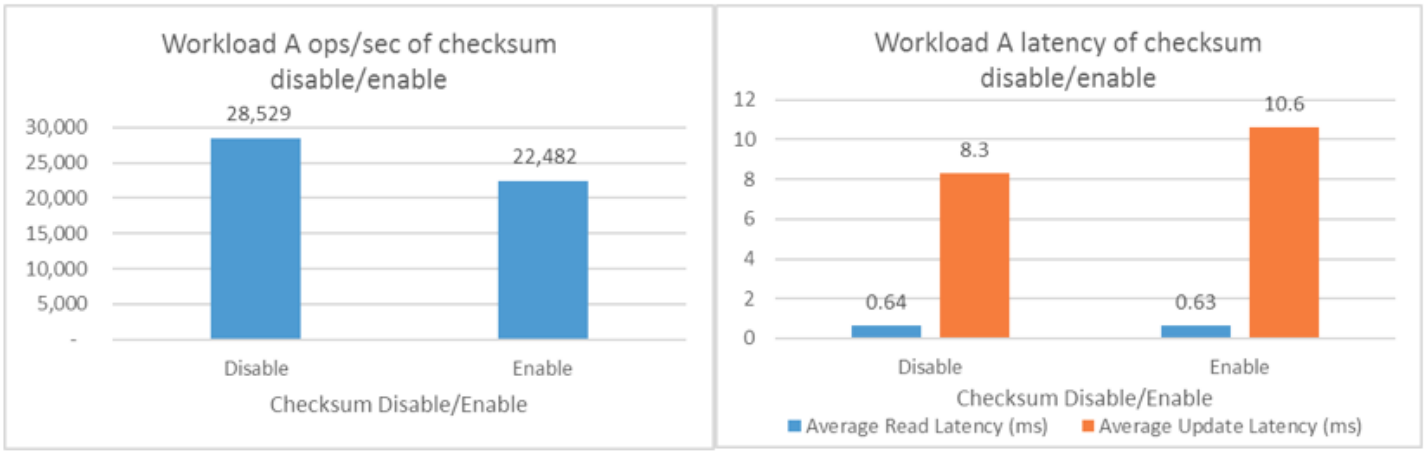

Figure 20. Workload A Test Results with Disable/Enable Checksum

For workload A, after enabling checksum, the ops/sec dropped from 28,529 to 22,482, which was a 21.2% decrease. The update latency increased from 8.3ms to 10.6ms while the variation for read latency was negligible.

Workload B

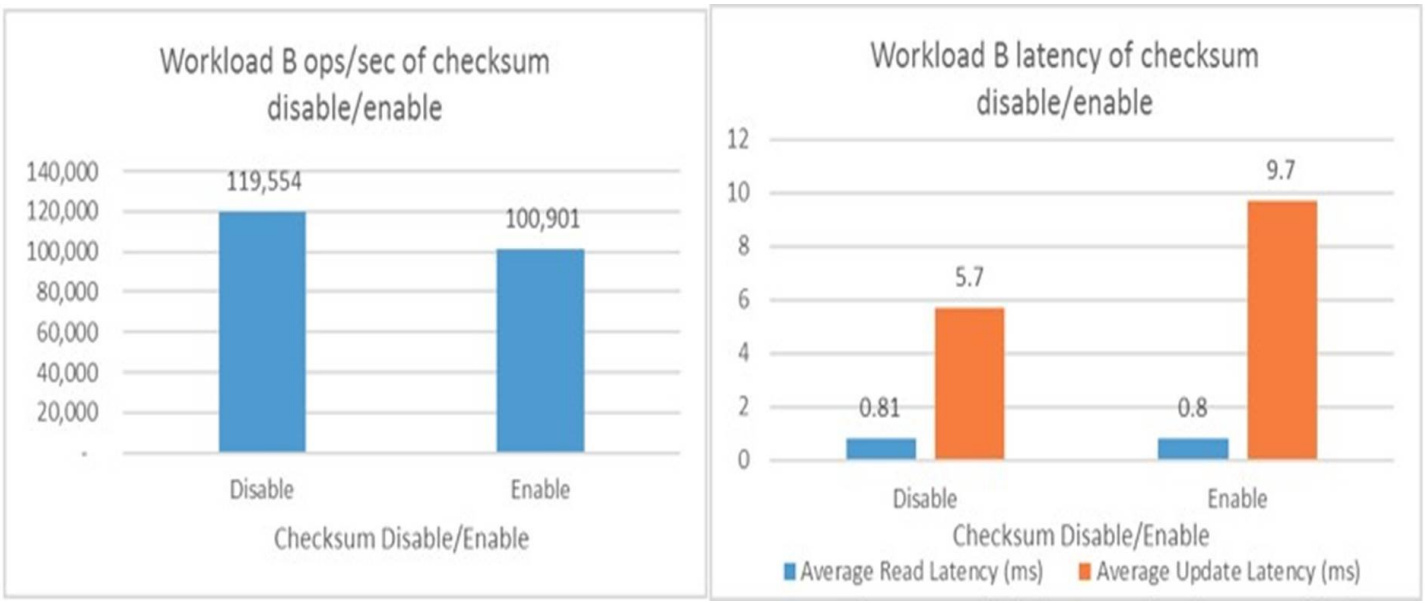

Figure 21. Workload B Test Results with Different Stripe Width

For workload B, after enabling checksum, the ops/sec dropped from 119,554 to 100,901, which was a 15.6% decrease. The update latency increased from 5.7ms to 9.7ms and the variation for read latency was also negligible.

We can see that enabling checksum can ensure data integrity but does have some penalty on performance.

Note: vSAN checksum is end-to-end software feature to avoid data integrity issues arising due to potential problems on the underlying storage media. We highly recommend using the default setting which is always enabled. The only reason to deactivate it is that the application already has this functionality included.

Evaluate the Impact of vSAN Cache Tier SSD (SATA or NVMe)

Test Overview

vSAN is tiered storage that consists of a cache and capacity tier. For an all-flash vSAN cluster, the cache tier is dedicated to write buffer and the capacity tier is used for providing high capacity.

In our baseline testing, we used NVMe SSDs as the cache tier. In some cases, users may want cheaper disks but with a relatively lower-level performance. So we replaced the NVMe SSDs in the cache tier with SATA SSDs to evaluate the impact of different

cache tier SSDs. The SATA SSD is 400GB in capacity, which is another differentiator between NVMe SSD and SATA SSD.

Workload A

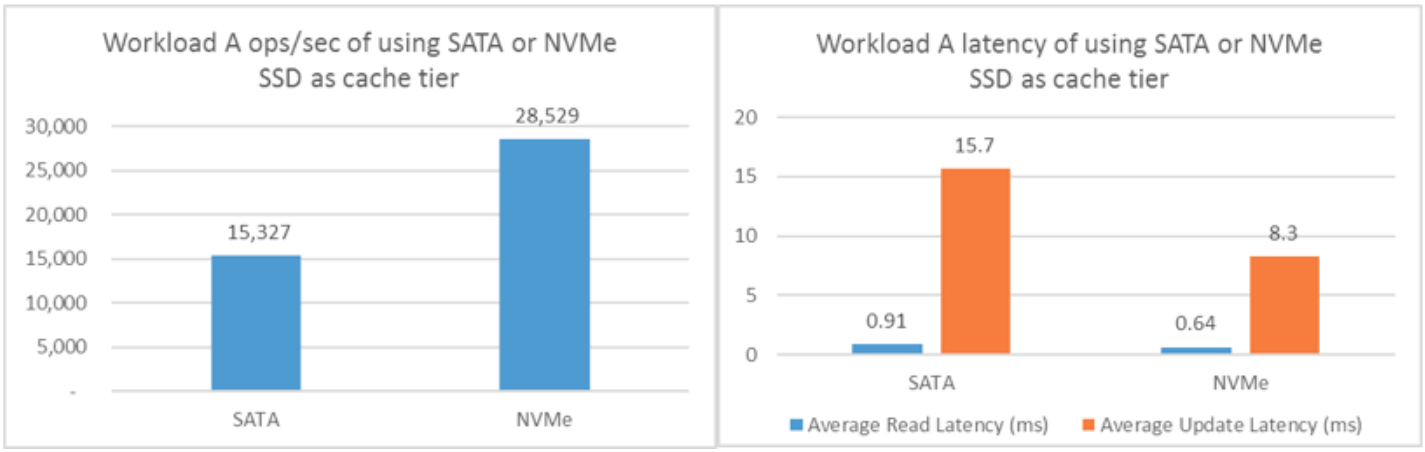

Figure 22. Workload A Test Results with SATA vs NVMe SSDs

For workload A, comparing SATA SSDs and NVMe SSDs as the cache tier, the ops/sec increased from 15,327 to 28.529, a 74.7% increase. The read latency decreased from 0.91ms to 0.64ms and the update latency decreased from 15.7ms to 8.7ms.

Workload B

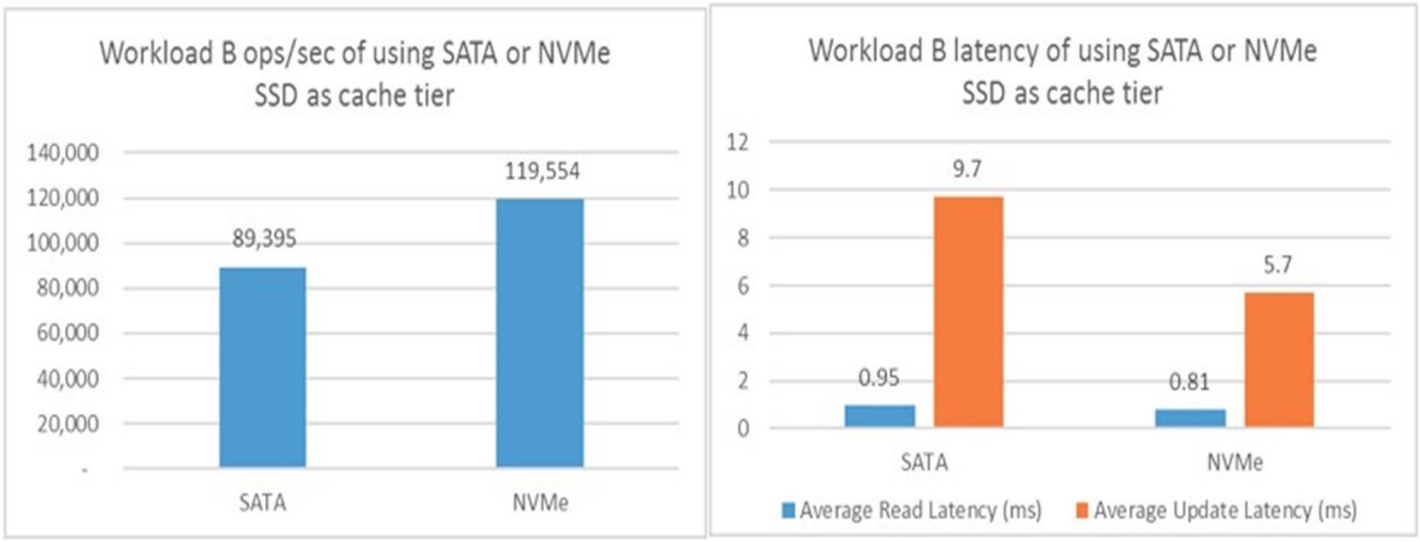

Figure 23. Workload B Test Results with SATA vs NVMe SSDs

For workload A, comparing SATA SSDs and NVMe SSDs as the cache tier, the ops/sec increased from 89,395 to 119,554, a 33.7% increase. The read latency decreased from 0.95ms to 0.81ms and the update latency decreased from 9.7ms to 5.7ms.

The results show that a more powerful hardware platform brings higher performance results. Users should carefully plan the hardware tier to meet the performance requirements.

Failure Testing and Failback Validation

Test Overview

In this failure testing section, we used the same parameter settings as the baseline testing.

From the perspective of MongoDB's replica set setting, the test is divided into two parts:

- Enable MongoDB's replica set which means there are three virtual machines in a MongoDB replica set, and thus we use 'rs=3' as the short term.
- Disable MongoDB's replica set which means there is only one virtual machine in a MongoDB replica set, and thus we use 'rs=1' for the short term.

From the perspective of failure, we conducted two types of failure:

- A physical host failure which will power off all the running virtual machines residing on it. When a host fails, VMware vSphere High Availability will restart the impacted virtual machines on another host. This is the backend feasibility of setting 'rs=1' while keeping a low service downtime.
- A physical disk failure in a vSAN datastore which will cause a vSAN object to enter a degraded state. With the storage policy set with FTT=1, the object can still survive and serve I/O. Thus from the virtual machines' perspective, there is no interruption of service.

We tested the combinations of the MongoDB replica set settings and the two failure modes.

rs=1 and Host Failure Testing

In the first test, we configured 'rs=1' and a physical host failure. We simulated a host failure by powering off a physical ESXi host. The following procedure was used:

1. We chose a host, 'esx-008', and checked the virtual machines running on it. The 'Mongo-s4' is one of the 'Mongos' servers and 'Mongo-shard4-1' is one of the shard servers. See Figure 24.
2. We powered off 'esx-008' from the host's remote management console.
3. We verified the virtual machines were restarted by vSphere HA. See Figure 25 and 26.
4. We verified that from vSAN's perspective, the component residing on the ESXi host 'esx-008' was marked as absent. As long as the host comes online within one hour, vSAN should not rebuild the component but resync that component instead. See Figure 27 and 28.
5. We verified the MongoDB service restarted. See Figure 29 and 30.

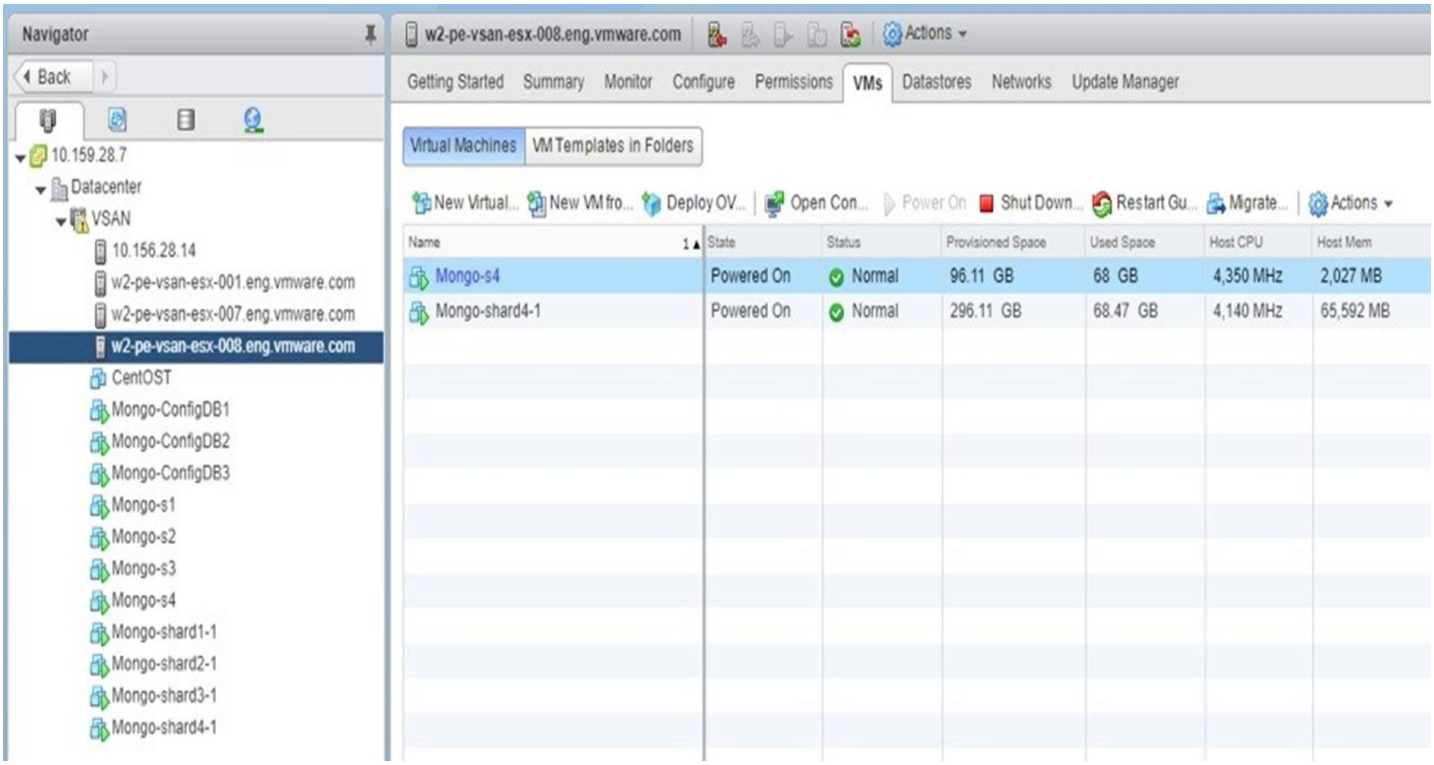

Figure 24. Check the Virtual Machines Impacted by a Host Failure

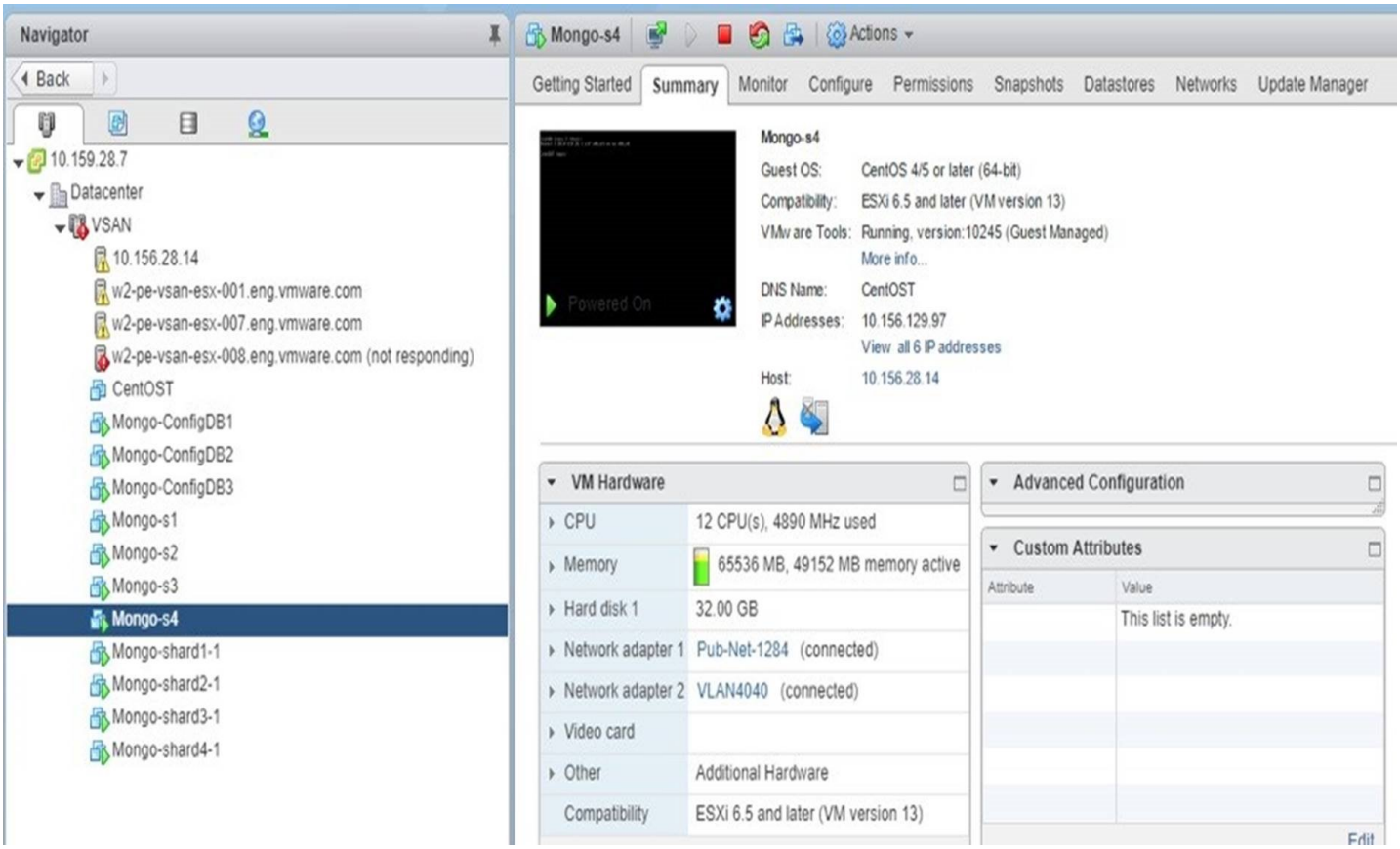

Figure 25. Check that 'Mongo-s4' was Restarted by vSphere HA

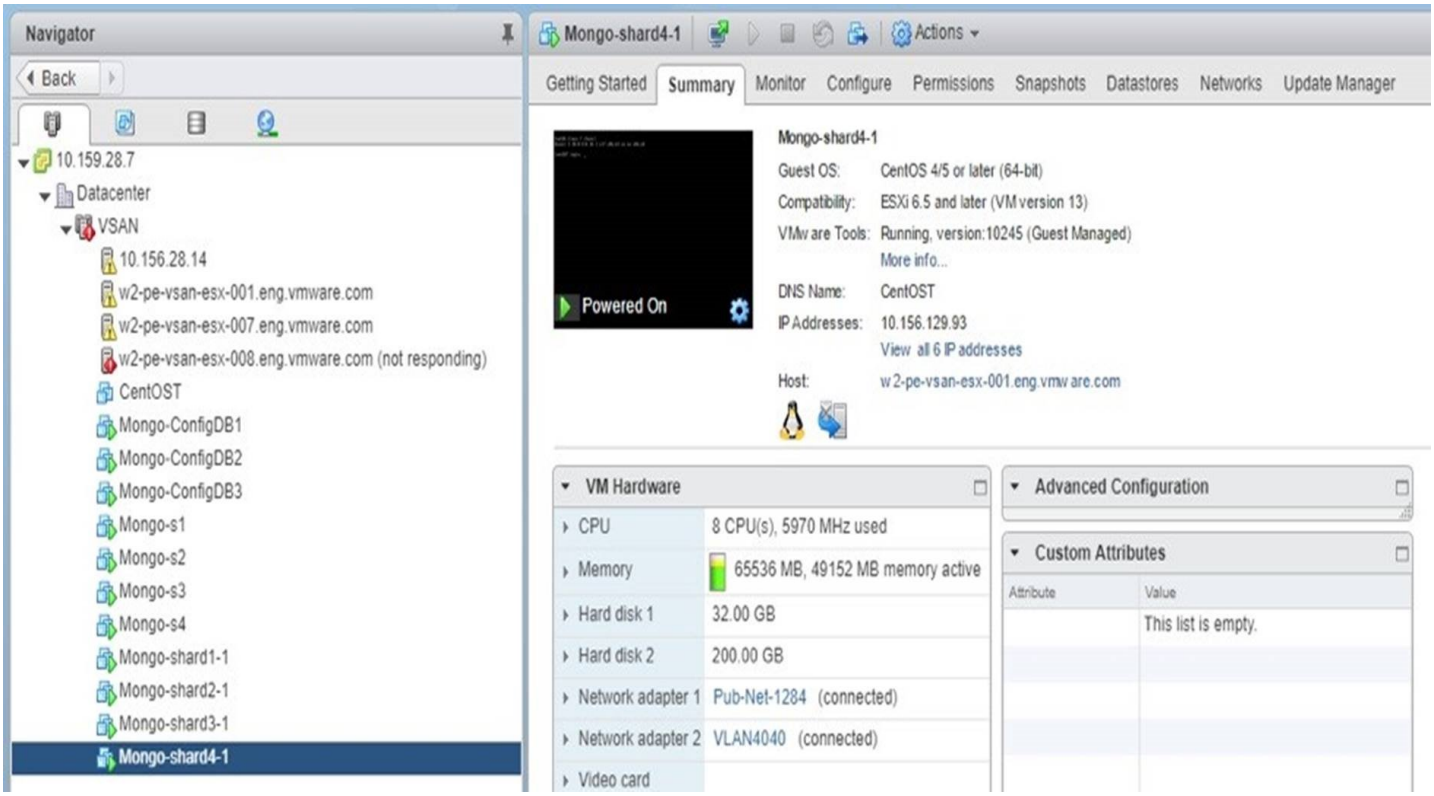

Figure 26. Check that 'Mongo-shard4-1' was Restarted by vSphere HA

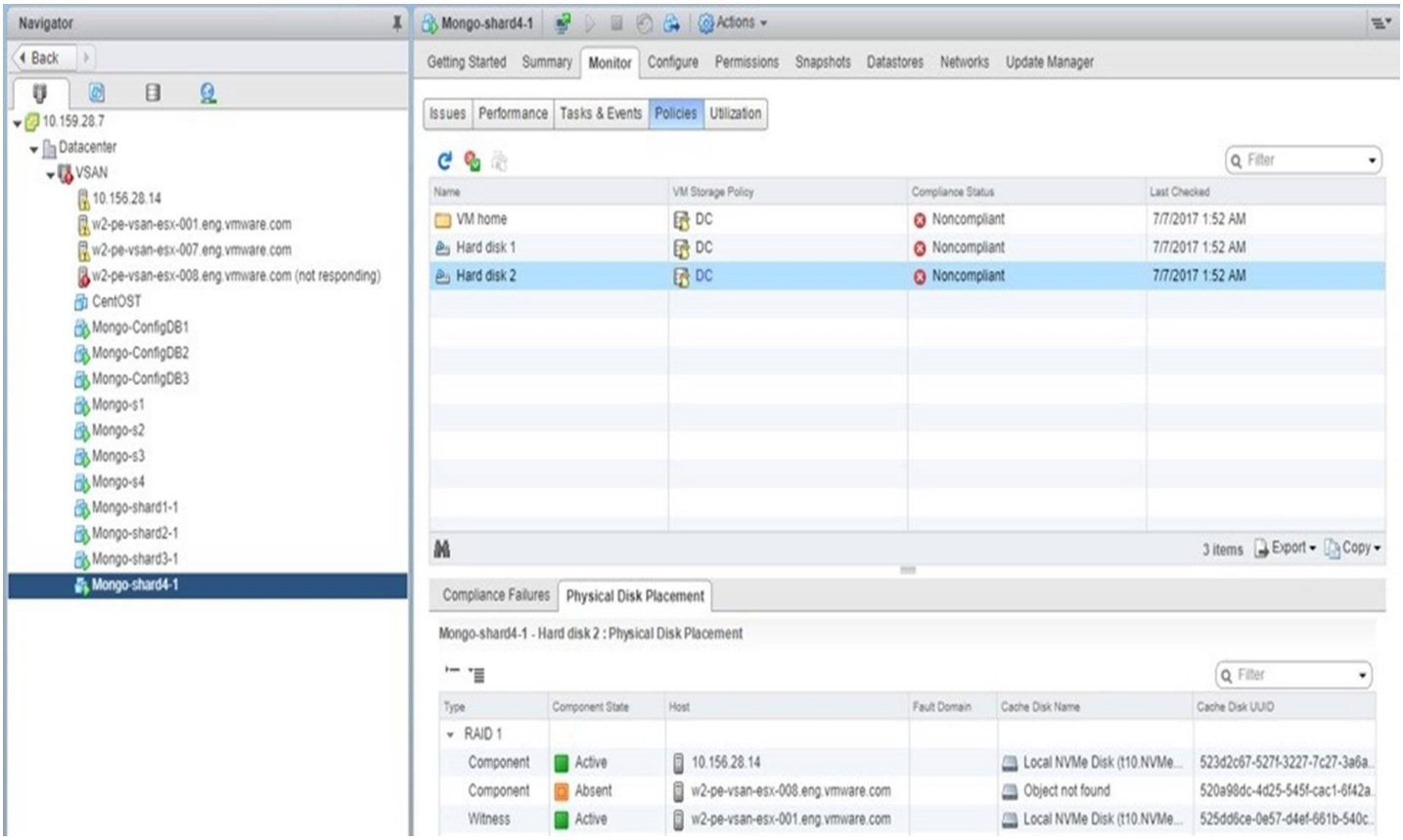

Figure 27. Check the Absent Components Impacted by a Host Failure

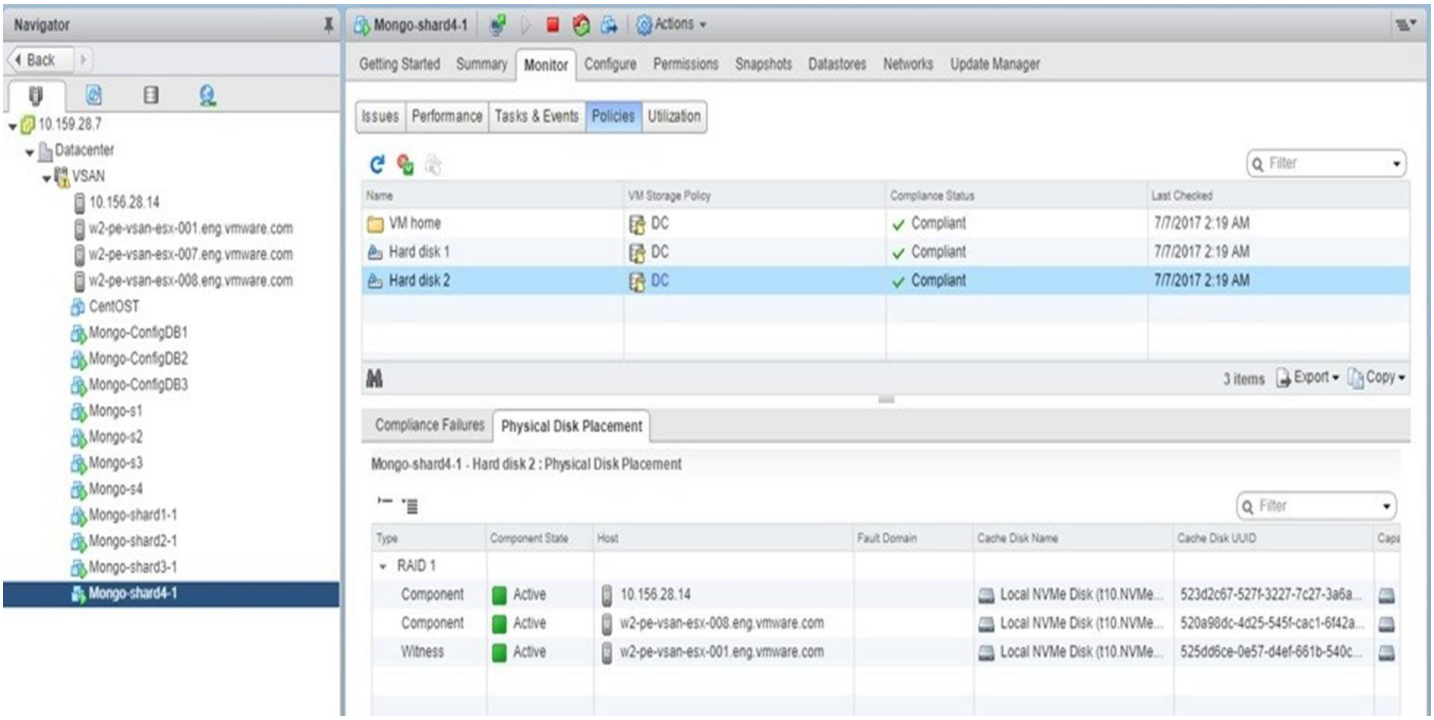

Figure 28. Host Came Online within One Hour and the vSAN Component Resynced

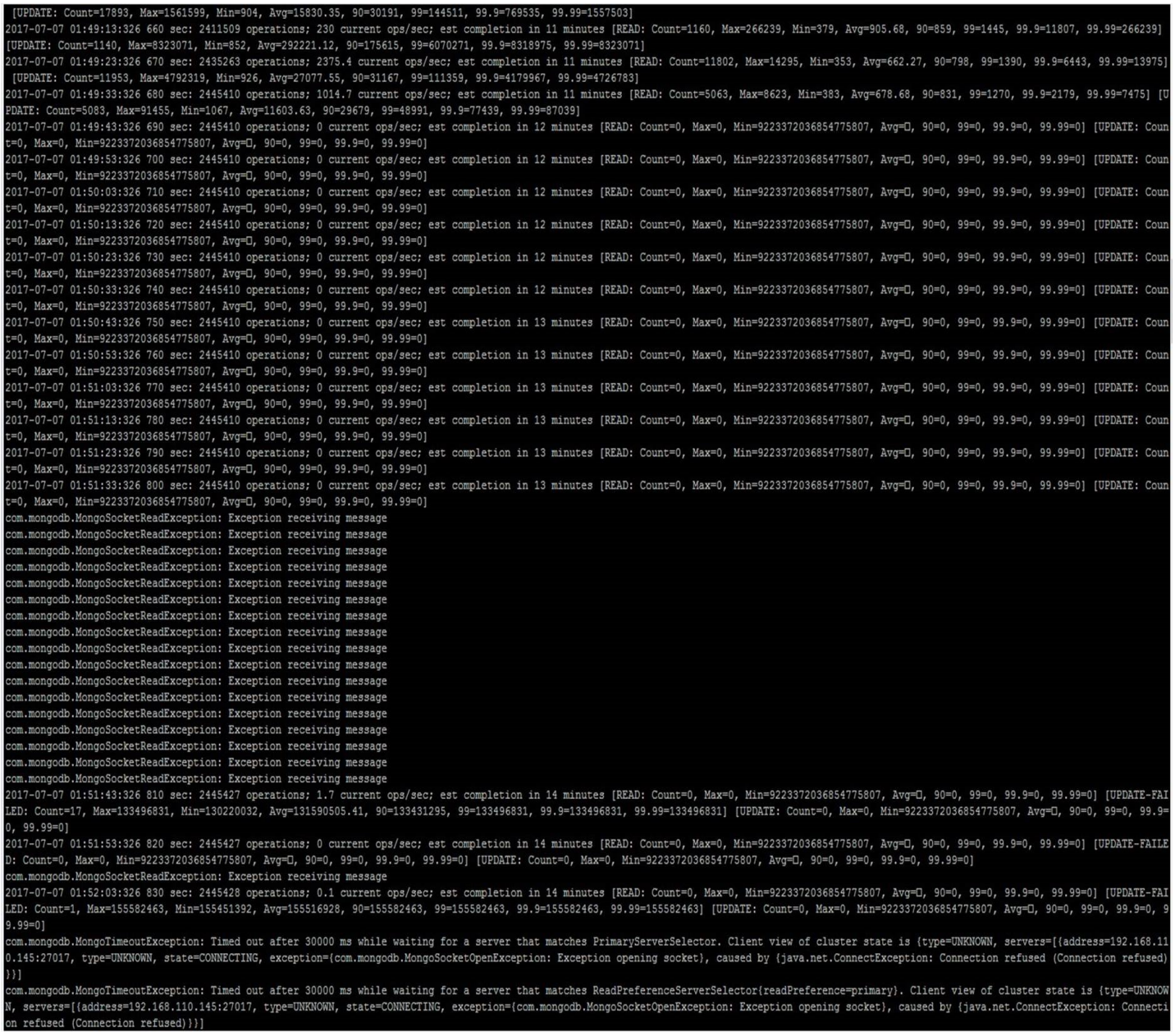

Figure 29. YCSB Client Reported Exceptions and Timeouts until vSphere HA Restarted the Virtual Machines

```
0.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connection refused (Connection refused)
}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches PrimaryServerSelector. Client view of cluster state is {type=UNKNOWN, servers=[{address=192.168.11
0.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connection refused (Connection refused)
}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches PrimaryServerSelector. Client view of cluster state is {type=UNKNOWN, servers=[{address=192.168.11
0.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connection refused (Connection refused)
}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches PrimaryServerSelector. Client view of cluster state is {type=UNKNOWN, servers=[{address=192.168.11
0.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connection refused (Connection refused)
}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches ReadPreferenceServerSelector{readPreference=primary}. Client view of cluster state is {type=UNKNOW
N, servers=[{address=192.168.110.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connecti
on refused (Connection refused)}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches PrimaryServerSelector. Client view of cluster state is {type=UNKNOWN, servers=[{address=192.168.11
0.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connection refused (Connection refused)
}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches ReadPreferenceServerSelector{readPreference=primary}. Client view of cluster state is {type=UNKNOW
N, servers=[{address=192.168.110.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connecti
on refused (Connection refused)}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches PrimaryServerSelector. Client view of cluster state is {type=UNKNOWN, servers=[{address=192.168.11
0.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connection refused (Connection refused)
}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches ReadPreferenceServerSelector{readPreference=primary}. Client view of cluster state is {type=UNKNOW
N, servers=[{address=192.168.110.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connecti
on refused (Connection refused)}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches ReadPreferenceServerSelector{readPreference=primary}. Client view of cluster state is {type=UNKNOW
N, servers=[{address=192.168.110.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connecti
on refused (Connection refused)}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches ReadPreferenceServerSelector{readPreference=primary}. Client view of cluster state is {type=UNKNOW
N, servers=[{address=192.168.110.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connecti
on refused (Connection refused)}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches ReadPreferenceServerSelector{readPreference=primary}. Client view of cluster state is {type=UNKNOW
N, servers=[{address=192.168.110.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connecti
on refused (Connection refused)}}]
com.mongodb.MongoTimeoutException: Timed out after 30000 ms while waiting for a server that matches ReadPreferenceServerSelector{readPreference=primary}. Client view of cluster state is {type=UNKNOW
N, servers=[{address=192.168.110.145:27017, type=UNKNOWN, state=CONNECTING, exception={com.mongodb.MongoSocketOpenException: Exception opening socket}, caused by {java.net.ConnectException: Connecti
on refused (Connection refused)}}]
2017-07-07 01:52:13:326 840 sec: 2445445 operations; 1.7 current ops/sec; est completion in 14 minutes [READ: Count=0, Max=0, Min=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0] [UPDATE-FAI
LED: Count=7, Max=30015487, Min=29999104, Avg=30007296, 90=30015487, 99=30015487, 99.9=30015487, 99.99=30015487] [UPDATE: Count=0, Max=0, Min=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0]
 [READ-FAILED: Count=10, Max=30015487, Min=29999104, Avg=30007296, 90=30015487, 99=30015487, 99.9=30015487, 99.99=30015487]
2017-07-07 01:52:23:326 850 sec: 2445445 operations; 0 current ops/sec; est completion in 14 minutes [READ: Count=0, Max=0, Min=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0] [UPDATE-FAILE
D: Count=0, Max=0, Min=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0] [UPDATE: Count=0, Max=0, Min=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0] [READ-FAILED: Count=0, Max=0, Mi
n=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0]
2017-07-07 01:52:33:326 860 sec: 2477204 operations; 3175.9 current ops/sec; est completion in 14 minutes [READ: Count=15925, Max=27410431, Min=356, Avg=13242.33, 90=1506, 99=10487, 99.9=18015, 99.9
9=22626303] [UPDATE-FAILED: Count=0, Max=0, Min=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0] [UPDATE: Count=15843, Max=22626303, Min=961, Avg=17122.02, 90=7123, 99=14335, 99.9=30175, 99.
99=22052863] [READ-FAILED: Count=0, Max=0, Min=9223372036854775807, Avg=□, 90=0, 99=0, 99.9=0, 99.99=0]
2017-07-07 01:52:43:326 870 sec: 2543695 operations; 6649.1 current ops/sec; est completion in 14 minutes [READ: Count=33295, Max=19807, Min=348, Avg=785.66, 90=1238, 99=3191, 99.9=9191, 99.99=18591
```

Figure 30. MongoDB Service Resumed and YCSB Resumed to a Normal Testing State

From YCSB’s report, we observed that the service was resumed in around 120 seconds.

rs=1 and Physical Disk Failure Testing

In the second test, we configured ‘rs=1’ and a physical disk failure. We simulated a physical disk failure by injecting a disk failure to a capacity SSD drive. The following procedure was used:

1. We chose a capacity disk and verified there were impacted vSAN components on it. We can see from Figure 31 that a data disk of ‘Mongo-shard4-1’ would be impacted if that disk failed.
2. We used the Disk Fault Injection script described in the Failure Testing document to inject a permanent disk error to the SSD disk with the impacted component, ‘naa.55cd2e404c08ef68’.
3. We verified the expected results:

- From MongoDB’s perspective, just one of the two vSAN components failed so the I/O should not be interrupted. Thus no MongoDB servers should fail and YCSB clients should all report normal results.
- From vSAN’s perspective, one of the components should be marked as degraded thus a ‘reconfiguring’ of the component should happen immediately. See Figure 32.

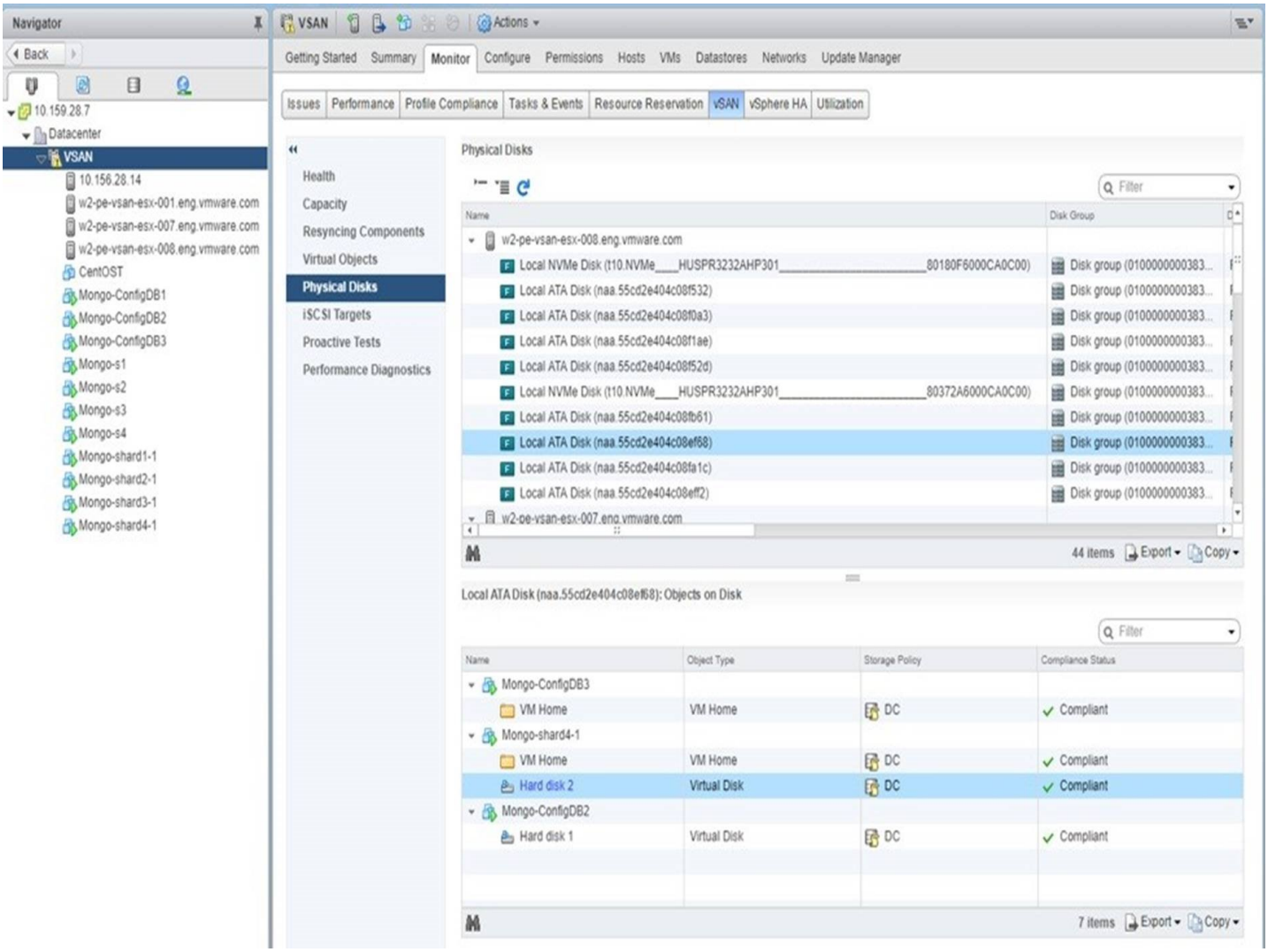

Figure 31. Check the Impacted Components if a Capacity SSD Disk Fails

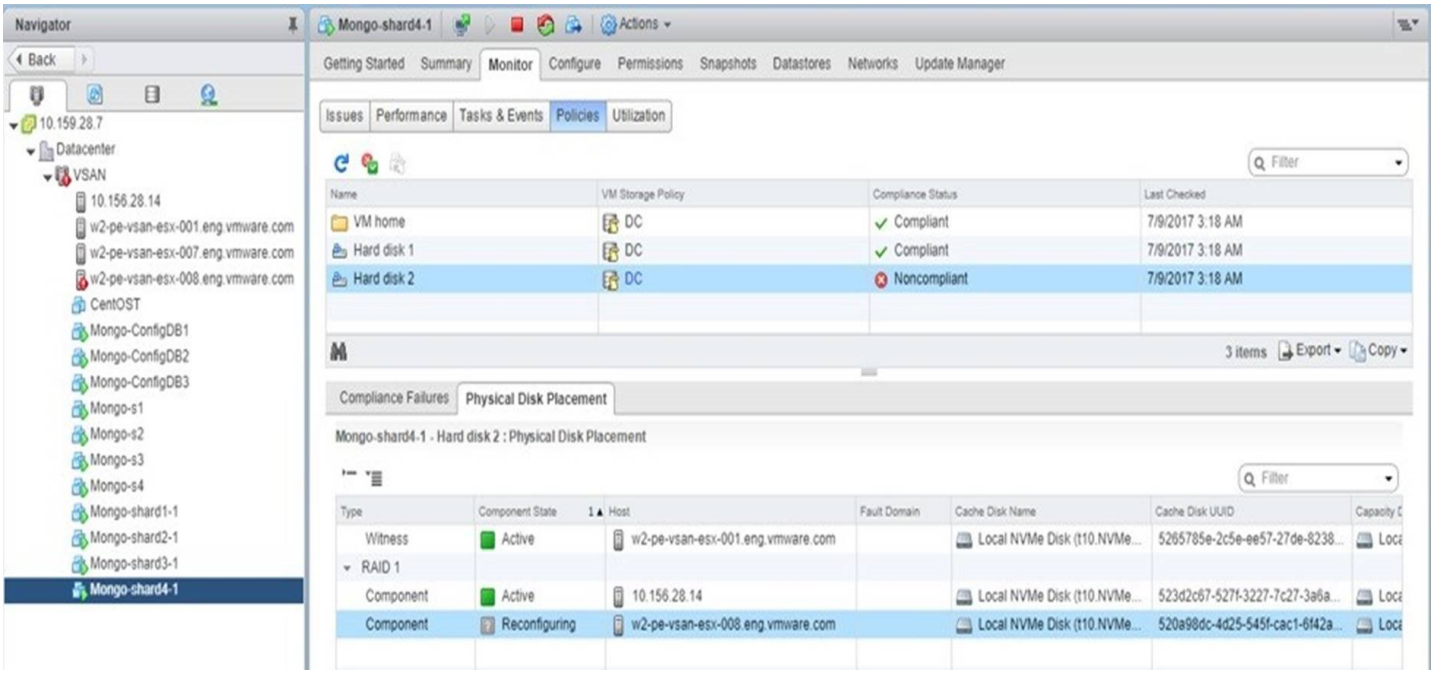

Figure 32. vSAN Started to Reconfigure the Impacted Component

After a few minutes, the impacted component was successfully rebuilt on another capacity SSD disk. See Figure 33. So in this case, vSAN handled the disk failure in the background and MongoDB service were not interrupted at all.

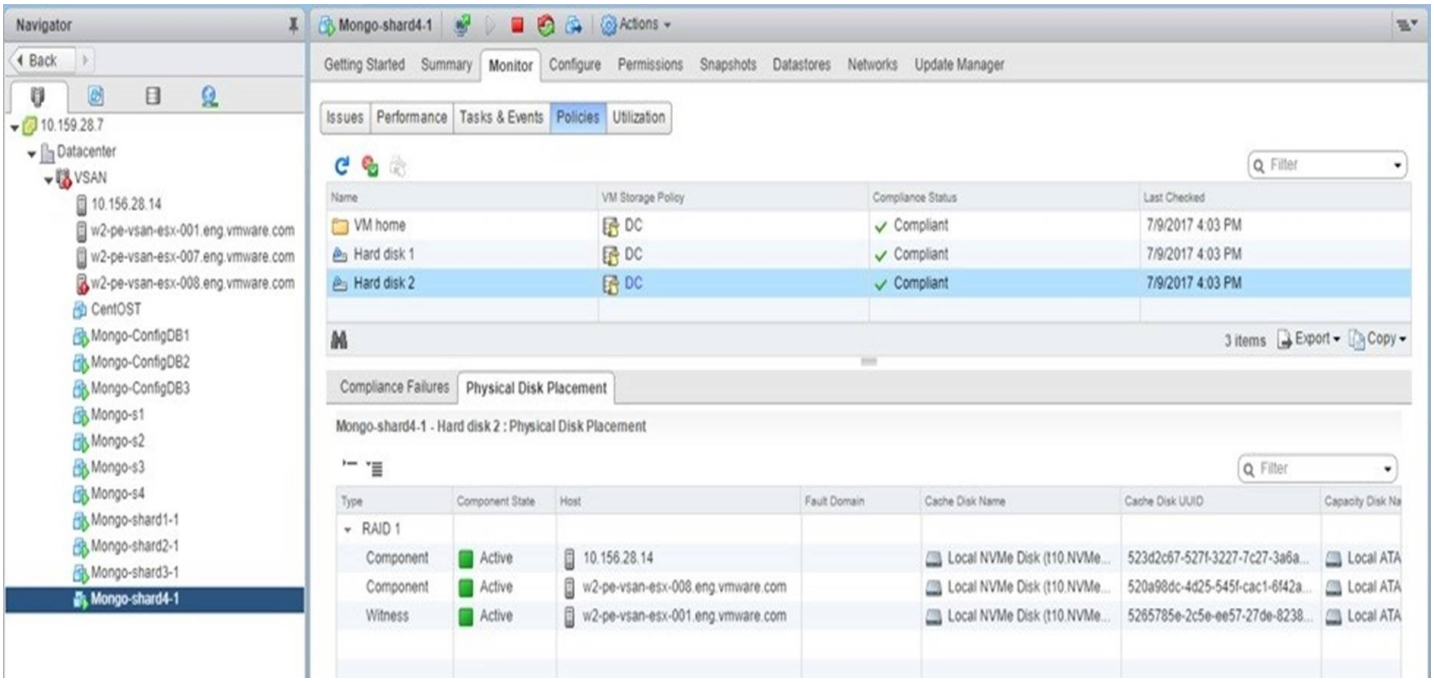

Figure 33. The Impacted Component was Successfully Rebuilt on Another Capacity SSD Disk

rs=3 and Host Failure Testing

In the third test, we configured 'rs=3' and a physical host failure. For this case, both MongoDB's application level replication and vSphere HA were configured. We simulated a host failure by powering off a physical ESXi host. The following procedure was used:

1. We chose a host, 'esx-008', and checked the virtual machines running on it. The 'Mongo-s4' is one of the 'Mongos' servers and 'Mongo-shard4-1' is one of the shard servers. See Figure 34.
2. We powered off 'esx-008' from the host's remote management console.
3. We verified that the MongoDB primary node fails over to the secondary node and the impacted virtual machine restarts on another host.

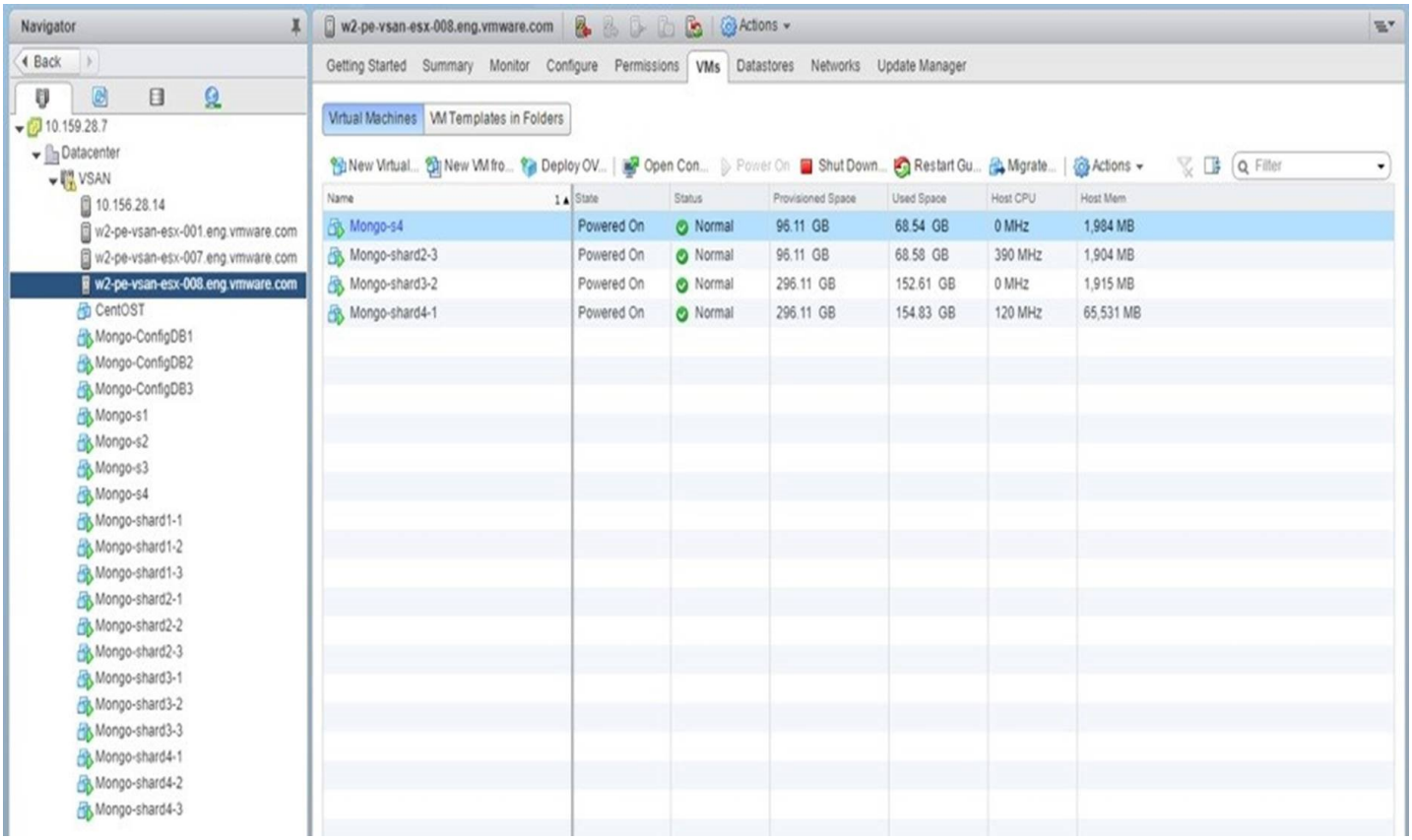

Figure 34. Check the Impacted Virtual Machines Impacted by a Host Failure

The following actions resulted:

- For the impacted virtual machine ‘Mongo-shard4-1’, the primary node in a replica set, the MongoDB failed over to the secondary node in the replica set. The recovery time was determined by MongoDB and we observed it was less than 10 seconds. Meanwhile, vSphere HA restarted the virtual machine ‘Mongo-shard4-1’ on another host. However, when ‘Mongo-shard4-1’ was restarted, it became the secondary node in the replica set. See Figure 35.
- For the impacted virtual machine ‘Mongo-shard3-2’, the secondary node in a replica set, MongoDB did not failover as the primary node still survived in this replica set. However, vSphere HA did restart the virtual machine ‘Mongo-shard3-2’ on another host. When it was restarted, it was activated as the secondary node in the replica set. For this shard, service was not interrupted. See Figure 35.
- Similar actions happened to other impacted virtual machines.

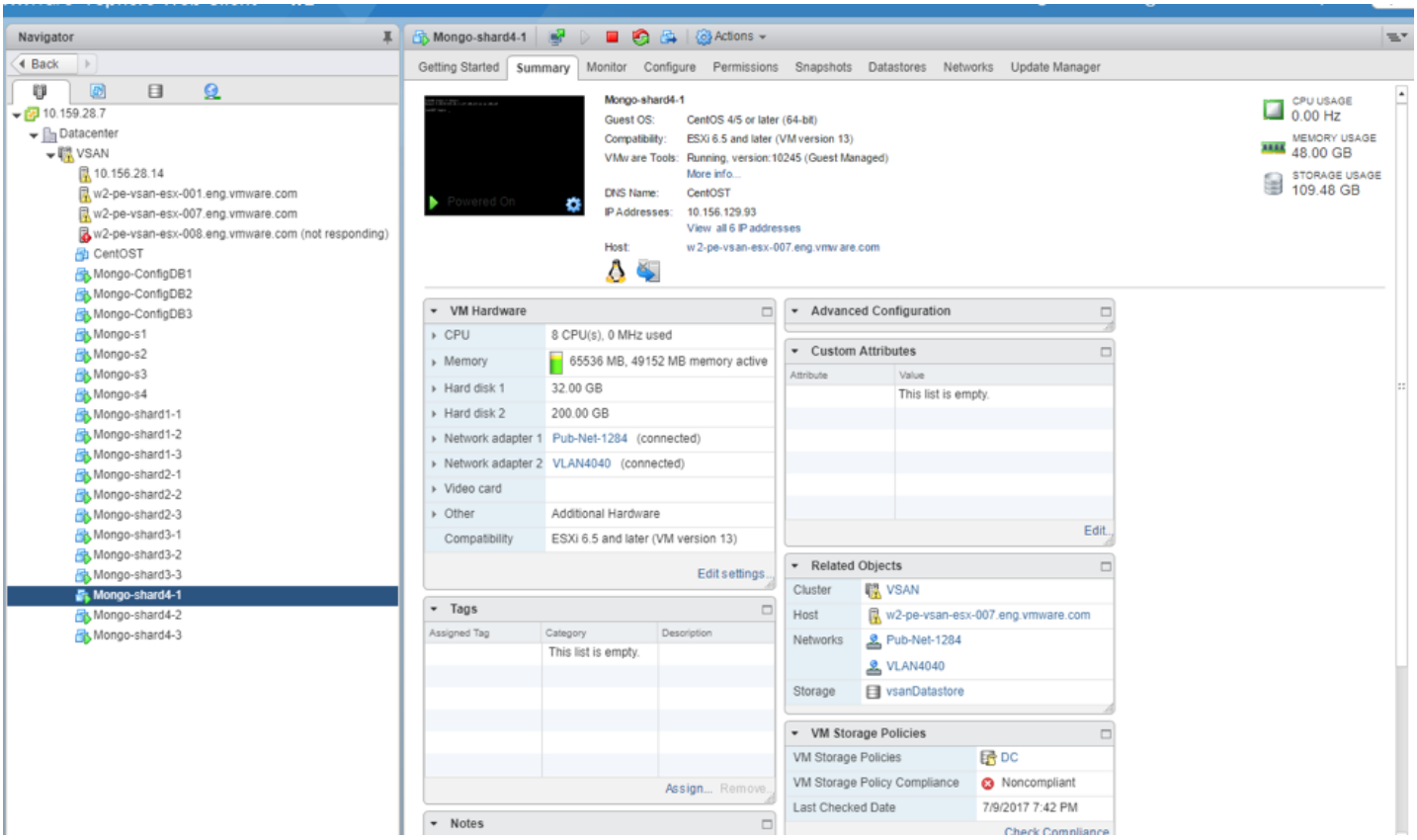

Figure 35. Check the Impacted Virtual Machine 'Mongo-shard4-1' Restarted by vSphere HA on Another Host

So only a primary node in a replica set residing on a failed host will cause service interruption. And the recovery time is less than 10 seconds due to MongoDB's failover. For other shard replica sets, there is no MongoDB failover thus no service interruption.

rs=3 and Physical Disk Failure Testing

In the fourth test, we configured 'rs=3' and a physical disk failure. We simulated a physical disk failure by injecting a disk failure to a capacity SSD drive. The following procedure was used:

We chose a capacity disk and verified there were impacted vSAN components on it. We can see from Figure 36 that a data disk of 'Mongo-shard2-2' and a data disk of 'Mongo-shard4-2' would be impacted if that disk failed.

We used the Disk Fault Injection script described in the Failure Testing document to inject a permanent disk error to the SSD disk with the impacted components, 'naa.55cd2e404c08ef68'.

We verified the expected results:

From MongoDB's perspective, just one of the two vSAN components failed so the I/O should not be interrupted. Thus no MongoDB servers should fail and YCSB clients should all report normal results.

From vSAN's perspective, one of the components should be marked as degraded thus a 'reconfiguring' of the component should happen immediately.

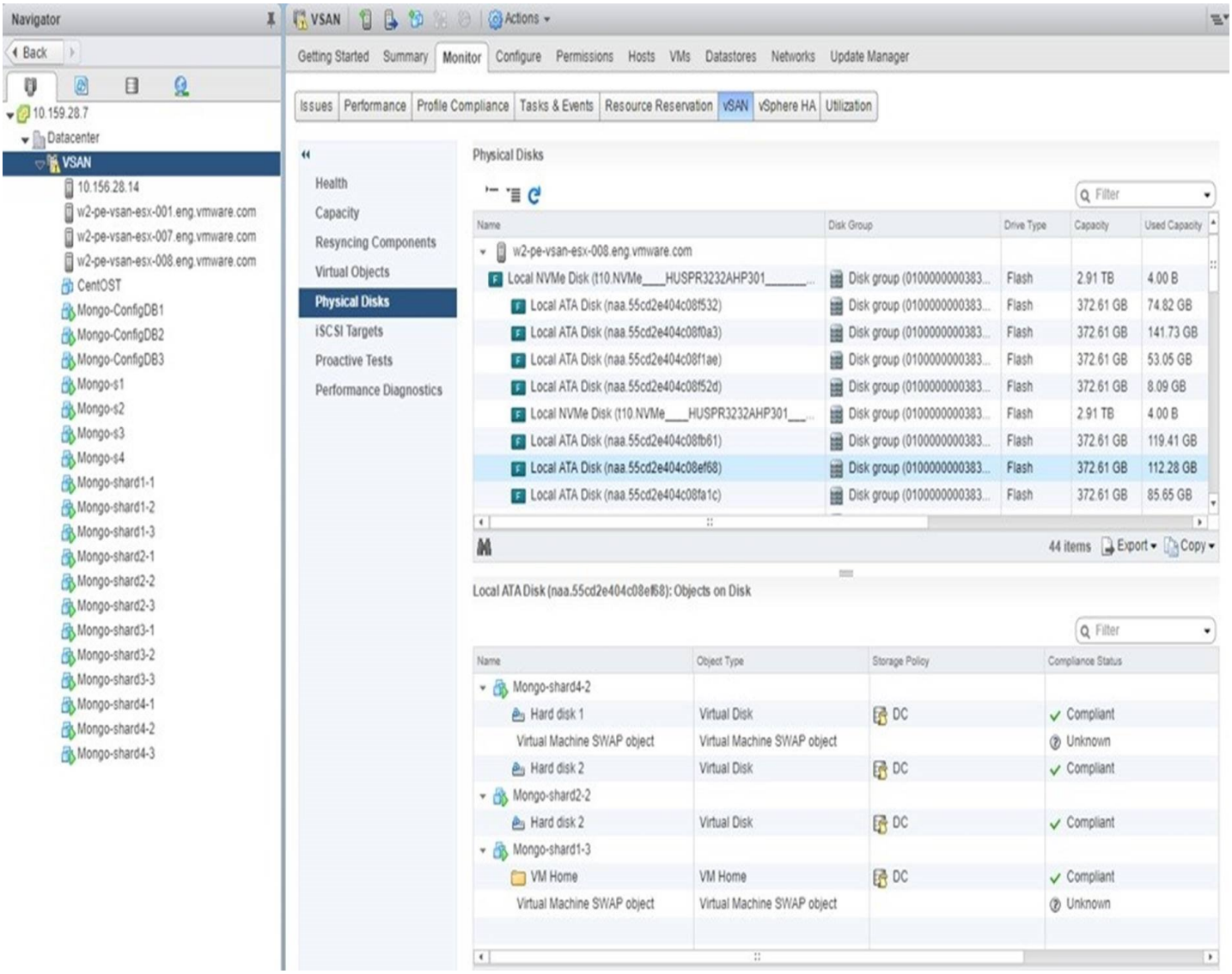

Figure 36. Check the Impacted Components if a Capacity SSD Disk Fails

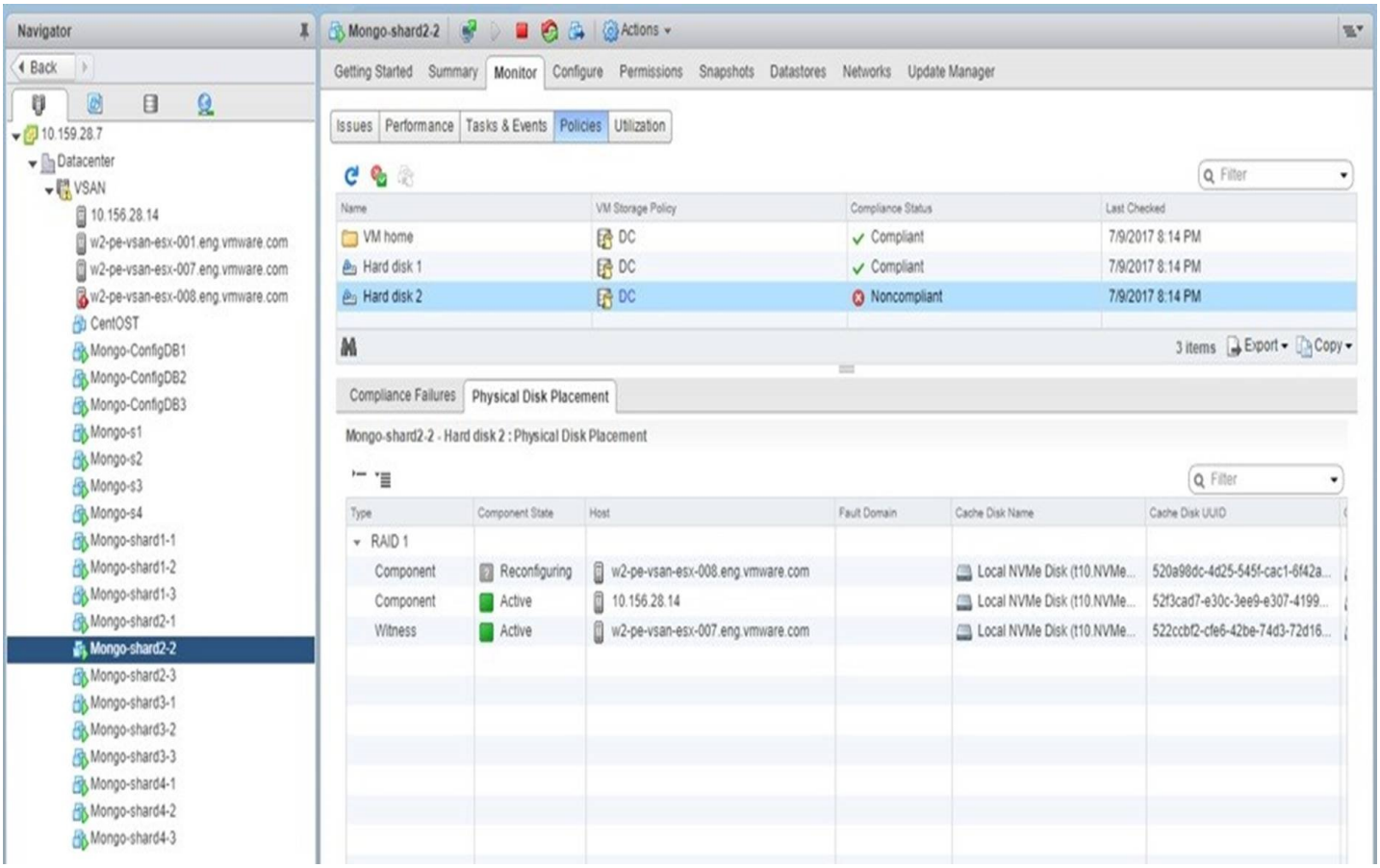

Figure 37. vSAN Started to Reconfigure the Impacted Component

In this case, both MongoDB's replication and vSphere HA are not aware of the underlying disk failure. vSAN will handle the disk failure in the background and MongoDB service is not interrupted at all.

Failure Testing Summary

The failure testing results were summarized in Table 7.

We used the same parameter settings as in the baseline testing. So the service interruption and resume time may vary with the change of the underlying testbed or some parameters.

Table 7. Failure Testing Summary

REPLICA SET CONFIGURATION	FAILURE TYPE	SERVICE INTERRUPTION TIME	RECOVERY METHOD
rs=1	Host Failure	Around 120 seconds	vSphere HA restarted the failed virtual machines.
rs=1	Disk Failure	No interruption	vSAN rebuilt the failed components.
rs=3	Host Failure	Around 10 seconds	1. MongoDB's replica set failed over from the primary node to the secondary node. 2. vSphere HArestarted the failed virtual machines.
rs=3	Disk Failure	No interruption	vSAN rebuilt the failed components.

Best Practices

This section lists the best practices of configuring MongoDB in a vSAN Cluster.

When configuring MongoDB in a vSAN cluster, consider the following best practices:

- Before the deployment, plan the database size and required performance level. This will ensure the hardware is properly deployed and software settings are best suited to serve the requests.
- Always use MongoDB's sharding mechanism to distribute workloads across multiple virtual machines.
- We can turn off MongoDB's replica set mechanism because vSphere HA and vSAN's FTT will provide data protection (it is an option, if you do not need application level replication).
- When configuring the virtual CPU and memory for the virtual machines used by MongoDB, choose an appropriate CPU and memory number to best suit the users' requirements.
- Choose a proper data durability option to meet the data durability requirements. Set an appropriate write acknowledgment value taking into consideration the tradeoff between performance and potential data loss.
- For vSAN policy settings, use a larger stripe width to achieve better performance.
- vSAN checksum is end-to-end software feature to avoid data integrity issues arising due to potential problems on the underlying storage media. We highly recommend using the default setting which is always enabled. The only reason to deactivate it is that the application already has this functionality included.
- Use a proper client number to connect to a MongoDB cluster because more client threads lead to higher latency.
- Follow the MongoDB Production Notes.

Conclusion

This section includes the summary of this solution reference architecture.

In this reference architecture, we describe the architecture, performance, and deployment considerations when deploying MongoDB in a vSAN cluster.

We introduce the components of MongoDB and the method of deploying a MongoDB cluster in a vSphere environment with vSAN enabled. Due to the natural feature of vSphere's Distributed computing and vSAN's distributed storage, we use MongoDB's sharding approach to deploy distributed workloads evenly in a vSphere cluster.

In the solution validation section, we prove that acceptable performance is achieved when running a MongoDB in a vSAN cluster. By changing various configuration settings, we evaluate the impact on performance with various parameters and provide best practices.

With vSAN's SPBM, administrators can ease their management by changing storage policies. Different storage policies will incur different performance improvements and achieve different performance results.

In the resiliency testing section, we prove that both MongoDB's replication method and vSAN's data protection method combined with vSphere HA can protect MongoDB from a host failure or a disk failure.

This reference architecture proves that running MongoDB in a vSAN cluster is a solid solution. By changing various parameters compared to the baseline architecture, users can achieve different levels of performance to meet specific business requirements.

About the Author

This section lists the writers of this reference architecture.

Victor Chen, Solutions Architect in the Product Enablement team of the Storage and Availability Business Unit wrote the original version of this paper.

Catherine Xu, Senior Technical Writer in the Product Enablement team of the Storage and Availability Business Unit edited this paper to ensure that the contents conform to the VMware writing style.