Technical Report

NetApp E-Series and MongoDB

Stephen Carl and Mitch Blackburn, NetApp

April 2016 | TR-4501

Abstract

This technical report describes the integrated architecture of the NetApp® E-Series and MongoDB NoSQL database design. Optimized for node storage balance, reliability, performance, storage capacity, and density, this design employs the MongoDB clustered node model, with higher scalability. In addition, this document summarizes the performance test results obtained from a MongoDB simulated client machine reference architecture utilizing Docker containers to virtualize a sharded MongoDB cluster using the I/O load generation tool Yahoo Client Server Benchmark (YCSB) in simulated customer workloads.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

NetApp E-Series enables NoSQL database environments such as MongoDB to maintain the highest levels of performance and uptime for workloads by providing advanced fault recovery features and easy in-service growth capabilities to meet ever-changing business requirements. The E-Series is designed to handle the most extreme application workloads with very low latency. Typical use cases include application acceleration; improving the response time of latency-sensitive applications; and improving the power, environmental, and capacity efficiency of overprovisioned environments. E-Series storage systems leverage the latest solid-state disk (SSD) and SAS drive technologies and are built on a long heritage of serving diverse workloads to provide superior business value and enterprise-class reliability.

NoSQL encompasses a wide variety of different database technologies that were developed in response to a rise in the volume of data stored about users, objects, and products; the frequency with which this data is accessed; and performance and processing needs. Relational databases, in contrast, were not designed to cope with the scale and agility challenges that face modern or third platform applications, nor were they built to take advantage of the inexpensive storage and processing power available today.

This technical report describes the integrated architecture of the NetApp E-Series and MongoDB NoSQL database design. This design employs the MongoDB clustered node model, with higher scalability. In addition, this document summarizes the performance test results obtained from a MongoDB simulated client machine reference architecture utilizing Docker containers to virtualize a sharded MongoDB cluster using the I/O load generation tool Yahoo Client Server Benchmark (YCSB) in simulated customer workloads. Three areas of E-Series E5600 performance are tested:

- All-SSD drives (equivalent to an EF560)
- Hard-disk drives (HDDs) and employing SSD drives for read acceleration (SSD read cache)
- E5600 configured with Dynamic Disk Pools (DDP) for enhanced recovery times when a drive fails (performance on failure)

The E-Series all-flash EF560 is an officially certified flash storage solution for MongoDB. For more information, visit the MongoDB and NetApp partner site.

2 MongoDB Overview

Document databases, including MongoDB, provide a rich set of indexing options to optimize a wide variety of queries, including text indexes, geospatial indexes, compound indexes, sparse indexes, time to live (TTL) indexes, unique indexes, and others. Furthermore, some of these products provide the ability to analyze data in place, without it needing to be replicated to dedicated analytics or search engines. MongoDB, for instance, provides both the aggregation framework for providing real-time analytics (along the lines of the SQL Server GROUP BY functionality) and a native MapReduce implementation for other types of sophisticated analyses. To update data, MongoDB provides a find and modify method so that values in documents can be updated in a single statement to the database, rather than making multiple round trips like other NoSQL alternatives.

All of your IT applications, systems, and technology infrastructure generate data every millisecond of every day. This machine data is one of the fastest growing and most complex areas of big data. MongoDB collects all of your data sources—streaming and historical—by using a technology called universal indexing. MongoDB is scalable enough to work across all of your data centers, and it is powerful enough to deliver real-time dashboard views to any level of the organization. Using this data can be a challenge for traditional data analysis, monitoring, and management solutions that are not designed for large-volume, high-velocity diverse data.

MongoDB offers a unique way to sift, distill, and understand these immense amounts of machine data, which can change how IT organizations manage, analyze, secure, and audit themselves. It enables users to develop valuable insights into how to innovate and offer new services, as well as into trends and customer behaviors.

2.1 Use Cases

MongoDB is typically used as the primary datastore for operational applications with real-time requirements (that is, low-latency, high availability). It is generally a good fit for 60% to 80% of the applications you may be building today and is easy to operate and scale in ways that are hard if not impossible with relational databases.

MongoDB excels in many use cases where relational databases aren't a good fit, such as applications with unstructured, semistructured, and polymorphic data, as well as applications with large scalability requirements or multi–data center deployments.

MongoDB may not be a good fit for some applications: for example, applications that require complex transactions (for example, a double-entry bookkeeping system) and scan-oriented applications that access large subsets of the data most of the time. MongoDB is not a drop-in replacement for legacy applications built around the relational data model and SQL Server.

Some common use cases include mobile apps, product catalogs, real-time personalization, content management, and applications delivering a single view across multiple systems. For further information, see MongoDB Use Cases.

Mobile

Your customers live on multiple screens, including smartphones and tablets. MongoDB powers the back end of these systems. Example applications include:

- Smartphone app for your insurance customers to submit claims with geo tags and pictures taken on their phones.
- Mobile app for government healthcare programs, including appointment scheduling, check-ins, and prescription refills.
- Quick app evolution: As new devices hit the market, ship your apps first, without redesigning your entire stack.
- Tablet-optimized app with not just a product catalog, but also a barcode scanner, local deals, loyalty program, and store locator.

Product Catalogs

A catalog stores lists of SKUs, F/X trades, and equipment—any asset or entity—as well as the associated metadata. This list goes far beyond the mail order catalogs that litter your mailbox or even the product catalogs on e-commerce websites. Example applications include:

- Central repository of trades across multiple asset classes, exposing trades for operational applications as well as aggregated analysis
- One datastore for thousands of assets under an organization’s purview, including vehicles, personnel, locations, and even information assets
- Master metadata for all your entities, physical or virtual, helping you match users with the right products at the right time
- Omnichannel product catalog and inventory management in real time, in the same datastore, with informed recommendations

Personalization

Personalization engines create customized online experiences for your customers in real time based on analysis of behavioral and demographic profiles, historical interactions, and preferences. They are built on top of legacy customer data management systems or replace them altogether. Example applications include:

- Deliver relevant lending offers with customized rates based on transaction history and credit scores.
- Make your citizens’ first touchpoint with government services a digital one. Users interact through portals customized for their location and status.
- Stay compliant by identifying customer location and making sure that data doesn’t travel across geographic borders.
- Recognize your customers when they return to your site or app. Match their preferences and history to offer products that have context.

Content Management Systems

Content management systems store and serve information assets and associated metadata to a range of applications such as websites, online publications, and archives. Example applications include:

- Replace expensive software such as SharePoint; aggregate, store, and serve equity research and its metadata for customers and internal users.
- Publish government archives online, including census data, birth and death records, and even historic artifacts.
- Consolidate your services and app back ends, websites, and media assets into a single database to integrate them more cleanly and simplify ops.
- Get visitors to click, interact, and add to their cart by pairing product listings with YouTube videos, live demos, and Twitter feeds that get them closer to the product.

Real-Time Analytics

Analytics applications fall along a spectrum of latency and availability. On one end of the spectrum sit batch analytical applications, which are used for complex, long-running analyses. They tend to have slower response times (up to minutes, hours, or days) and lower requirements for availability. Examples of batch analytics include Hadoop-based workloads.

On the other end of the spectrum sit real-time analytical applications, which provide lighter-weight analytics very quickly. Latency is low (subsecond), and availability requirements are high (for example, 99.99%). MongoDB is typically used for real-time analytics. Example applications include:

- Analyze ticks, tweets, satellite imagery, weather trends, and any other type of data to inform trading algorithms in real time.
- Identify social program fraud within seconds based on program history, citizen profile, and geospatial data.
- Identify unique individuals across any type of device, browser, or app and use a holistic behavioral model to advertise to them.
- Set up a digital geo-fence around your brick-and-mortar locations to push in-store incentives to shoppers in real time.

IoT

The Internet of Things is a world where all your physical assets and devices are connected to each other and share information, making life easier and more convenient.

- Remotely monitor vehicle performance and driver behavior using telematic sensor data to text those metrics with insurance premiums.
- Use biometric sensor data from patients to alert doctors early so they can prevent medical emergencies.

- Give your users the tools and toys to quantify their lifestyles with wearable tech, analyzing diet, exercise, sleep, and the rest of their activities.
- Present enticing offers to shoppers using in-store beacons and purchase history data as they walk through your store.

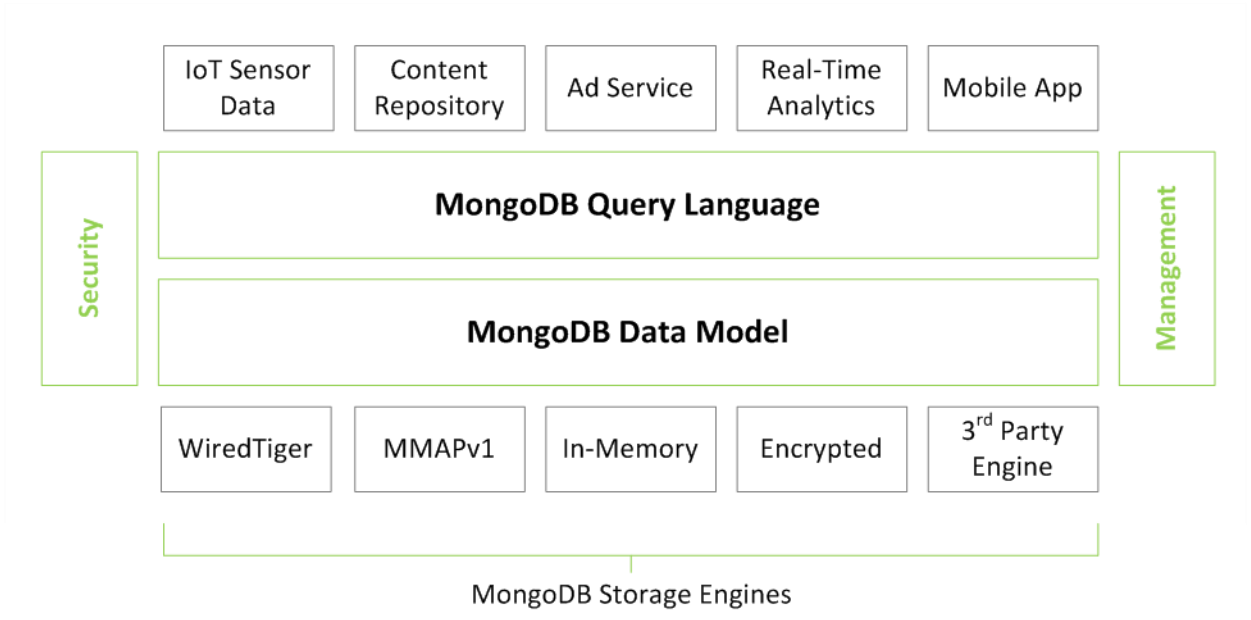
2.2 Architecture

While offering innovations, NoSQL systems have sacrificed the critical capabilities that people have come to expect and rely on from relational databases. The MongoDB database harnesses the innovations of NoSQL while maintaining the foundation of relational databases. For in-depth information, see the MongoDB Architecture Guide. Some of the important features from this guide are summarized in the following sections.

Flexible Storage Architecture

With MongoDB's flexible storage architecture, as shown in Figure 1 (source: MongoDB, 2015), the database automatically manages the movement of data between storage engine technologies using native replication. This approach significantly reduces developer and operational complexity when compared to running multiple distinct database technologies. Users can leverage the same MongoDB query language, data model, scaling, security, and operational tooling across different parts of their application, with each powered by the optimal storage engine.

Figure 1) MongoDB flexible storage architecture.



Data as Documents

MongoDB stores data as documents in a binary representation called binary JSON (BSON). The BSON encoding extends the popular JavaScript object notation (JSON) representation to include additional types such as int, long, date, and floating point. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, binary data, and subdocuments.

Query Model

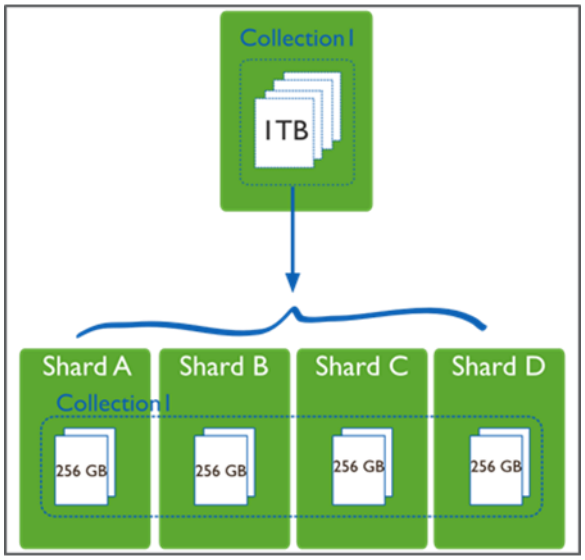
MongoDB provides native drivers for all popular programming languages and frameworks to make development natural. Supported drivers include Java, .NET, Ruby, PHP, JavaScript, node.js, Python, Perl, PHP, Scala, and others, in addition to 30+ community-developed drivers. MongoDB drivers are designed to be idiomatic for the given language.

One fundamental difference as compared to relational databases is that the MongoDB query model is implemented as methods or functions within the API of a specific programming language, as opposed to a completely separate language such as SQL Server. This fact, coupled with the affinity between MongoDB's JSON document model and the data structures used in object-oriented programming, makes integration with applications simple.

Data Management

Sharding, or horizontal scaling, divides the dataset and distributes the data over multiple servers, or shards. Each shard is an independent database. Collectively, the shards make up a single logical database. Figure 2 shows how shards in a collection can be spread across multiple databases.

Figure 2) Shards spread across multiple servers.

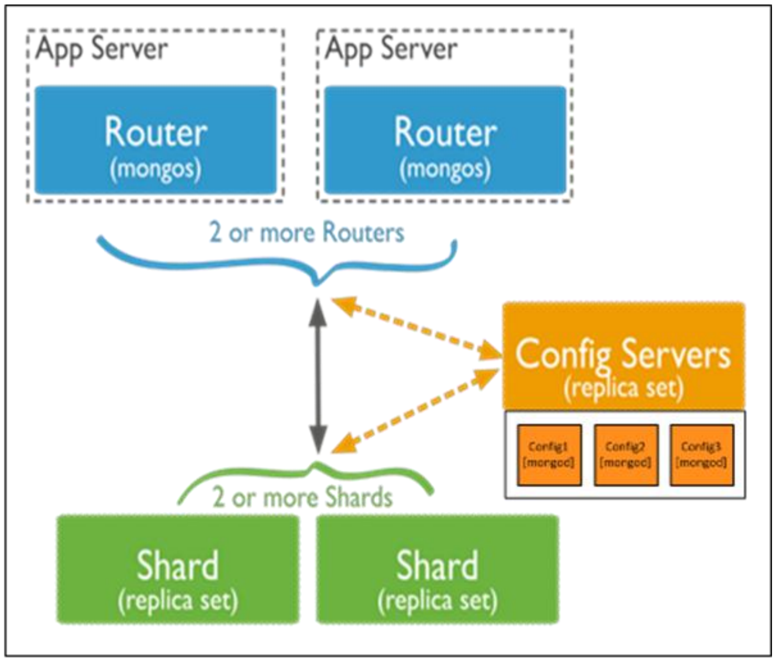


Shards store data. In a production sharded cluster, each shard is a replica set. This strategy provides high availability and data consistency.

Query routers, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. A client sends requests to a mongos, which then routes the operations to the shards and returns the results to the clients. A sharded cluster can contain more than one mongos to divide the client request load. For this reason, most sharded clusters have more than one mongos.

Configuration servers (config servers) store the cluster's metadata, which contains a mapping of the cluster's dataset to the shards. The query router uses this metadata to target operations to specific shards, as Figure 3 shows.

Figure 3) Query routers (mongos instances) using metadata to route data to shards.



Note: Starting in MongoDB 3.2, config servers for sharded clusters can be deployed as a replica set. The replica set config servers must run the WiredTiger default storage engine. MongoDB 3.2 deprecates the use of three mirrored mongod instances for configured servers.

3 NetApp E-Series Overview

The E-Series E5600 is an industry-leading storage system that delivers high input/output operations per second (IOPS) and bandwidth with consistently low latency to support the demanding performance and capacity needs of science and technology, simulation modeling, and decision support environments. In addition, the E5600 is equally capable of supporting primary transactional databases, general mixed workloads, and dedicated workloads such as video analytics in a highly efficient footprint with extreme simplicity, reliability, and scalability.

The E5600 provides the following benefits:

- Support for wide-ranging workloads and performance requirements
- Fully redundant I/O paths, advanced protection features, and proactive support monitoring and services for high levels of availability, integrity, and security
- Increased IOPS performance by up to 35% compared to the previous high-performance generation of E-Series products
- A level of performance, density, and economics that leads the industry
- Interface protocol flexibility to support FC host and iSCSI host workloads simultaneously
- Support for private and public cloud workloads behind virtualizers such as FlexArray®, Veeam Cloud Connect, and StorageGRID®

3.1 E-Series Hardware Overview

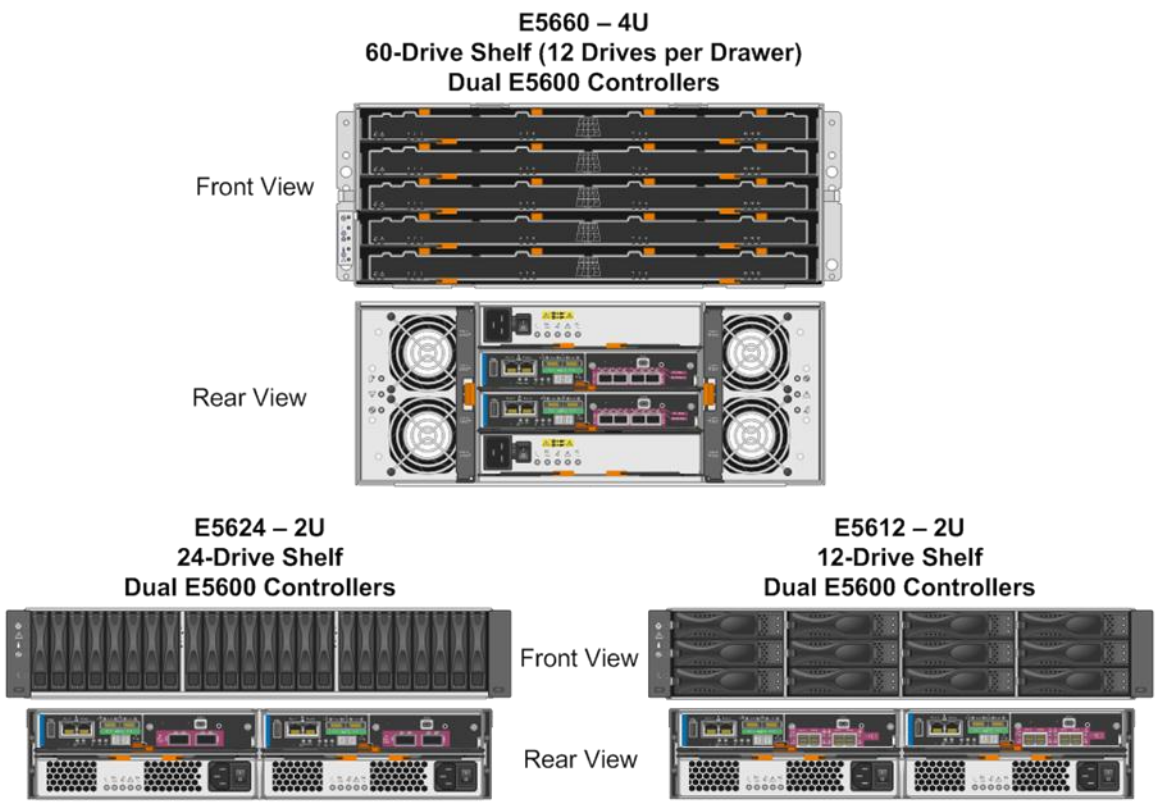
As shown in Table 1, the E5600 is available in three shelf options, which support both HDDs and SSDs to meet a wide range of performance and application requirements.

Table 1) E5600 controller shelf and drive shelf models.

Controller Shelf Model	Drive Shelf Model	Number of Drives	Type of Drives
E5660	DE6600	60	2.5” and 3.5” SAS drives (HDDs and SSDs)
E5624	DE5600	24	2.5” SAS drives (HDDs and SSDs)
E5612	DE1600	12	3.5” SAS drives (HDDs only)

All three shelf options include dual controller modules, dual power supplies, and dual fan units for redundancy (the 12-drive and 24-drive shelves have integrated power and fan modules). The shelves are sized to hold 60 drives, 24 drives, or 12 drives, as shown in Figure 4.

Figure 4) E5600 controller-drive shelf options.



Each E5600 controller shelf includes two controllers, with each controller providing two Ethernet management ports for out-of-band management. The system also supports in-band management access and has two 6Gbps wide-port SAS drive expansion ports for redundant drive expansion paths. The E5600 controllers do not include built-in host ports, but must be ordered with one of the following host interface cards (HICs) installed in each controller:

Note: Both controllers in an E5600 array must be identically configured.

- 4-port 12Gb SAS HIC
- 2-port 56Gb InfiniBand (IB) HIC. This HIC runs the iSCSI Extensions for RDMA (iSER) protocol as shipped, but it can be converted to SCSI RDMA Protocol (SRP) before initial use by applying a software feature pack in the field at no additional cost.

- 4-port optical HIC, which can be factory-configured as either 16Gb Fibre Channel or 10Gb iSCSI. A software feature pack can be applied in the field to change the host protocol of this HIC:
  - From FC to iSCSI
  - From iSCSI to FC
  - From either FC or iSCSI to FC-iSCSI split mode
  - From FC-iSCSI split mode back to FC or iSCSI

Note: In FC-iSCSI split mode, ports 1 and 2 operate as iSCSI, and ports 3 and 4 operate as FC.

E5600 controllers are available with two memory options: a standard 12GB DIMM, which can be used with all protocols, and a 48GB DIMM, which can be used with iSCSI and FC only. Controller memory upgrades from 12GB to 48GB are not available.

3.2 SANtricity Software

E-Series systems are managed by the SANtricity® Storage Manager application. Simple to download and install, SANtricity Storage Manager provides an intuitive, wizard-led GUI as well as full support for a CLI. SANtricity Storage Manager can be installed on a Microsoft Windows, Solaris, or Linux operating system (OS) platform for out-of-band management of the storage array.

To create volume groups on the array, the first step when configuring SANtricity is to assign a redundant array of inexpensive disks (RAID) level. This assignment is then applied to the disks selected to form the volume group. The E5600 storage systems support RAID levels 0, 1, 3, 5, 6, and 10 or DDP. DDP was used for all configurations described in this document.

To simplify the storage provisioning, NetApp provides a SANtricity automatic configuration feature. The configuration wizard analyzes the available disk capacity on the array. It then selects disks that maximize array performance and fault tolerance while meeting capacity requirements, hot spares, and any other criteria specified in the wizard.

Dynamic Capabilities

From a management perspective, SANtricity offers a number of capabilities to ease the burden of storage management, including the following:

- New volumes can be created and are immediately available for use by connected servers.
- New RAID sets (volume groups) or Dynamic Disk Pools can be created any time from unused disk devices.
- Volumes, volume groups, and disk pools can all be expanded online as necessary to meet any new requirements for capacity or performance.
- Dynamic RAID migration allows the RAID level of a particular volume group, for example, from RAID 10 to RAID 5, to be modified online if new requirements dictate a change.
- Flexible cache block and segment sizes enable optimized performance tuning based on a particular workload. Both items can also be modified online.
- There is built-in performance monitoring of all major storage components, including controllers, volumes, volume groups, pools, and individual disk drives.
- Automated remote connection to the NetApp AutoSupport® function enables “phone home” capabilities and automated parts dispatching in case of component failures.
- Path failover and load balancing (if applicable) between the host and the redundant storage controllers in the E5600 are provided.

SSD Read Cache

The SANtricity SSD cache feature uses SSD storage to hold frequently accessed data from user volumes. It is intended to improve the performance of workloads that are performance limited by HDD

IOPS. Workloads with the following characteristics can benefit from using the SANtricity SSD cache feature:

- Read performance is limited by HDD IOPS.
- There is a high percentage of read operations relative to write operations.
- A large number of reads are repeat reads to the same or adjacent areas of disk.
- The size of the data that is repeatedly accessed is smaller than the SSD cache capacity.

For more information about SSD read cache, see TR-4099: NetApp SANtricity SSD Cache for E-Series.

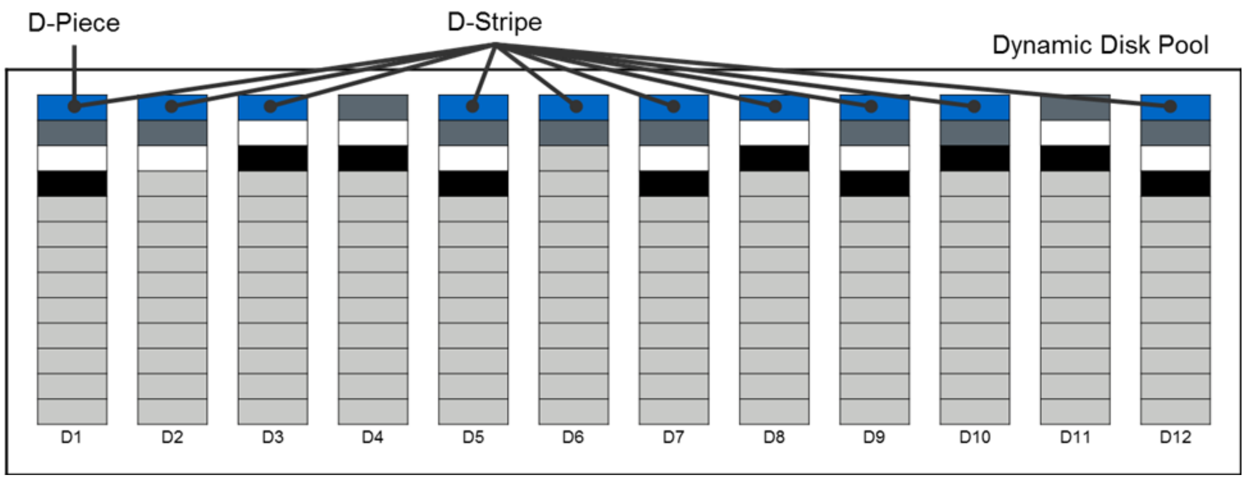
Dynamic Disk Pools

With seven patents pending, the DDP feature dynamically distributes data, spare capacity, and protection information across a pool of disk drives. These pools can range in size from a minimum of 11 drives to all the supported drives in a system. In addition to creating a single DDP, storage administrators can opt to mix traditional volume groups and DDP or even multiple DDPs, offering an unprecedented level of flexibility.

Dynamic Disk Pools are composed of several lower-level elements. The first of these is a D-piece. A D-piece consists of a contiguous 512MB section from a physical disk that contains 4,096 128KB segments. Within a pool, 10 D-pieces are selected using an intelligent optimization algorithm from selected drives within the pool. Together, the 10 associated D-pieces are considered a D-stripe, which is 4GB of usable capacity in size. Within the D-stripe, the contents are similar to a RAID 6 8+2 scenario. There, 8 of the underlying segments potentially contain user data, 1 segment contains parity (P) information calculated from the user data segments, and 1 segment contains the Q value as defined by RAID 6.

Volumes are then created from an aggregation of multiple 4GB D-stripes as required to satisfy the defined volume size up to the maximum allowable volume size within a DDP. Figure 5 shows the relationship between these data structures.

Figure 5) Dynamic Disk Pool components.

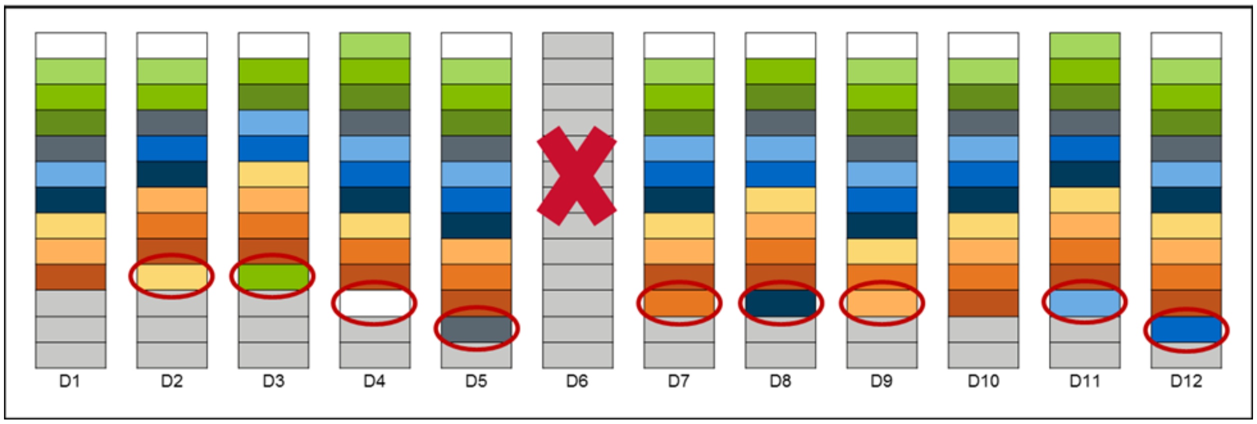


Another major benefit of a DDP is that, rather than using dedicated stranded hot spares, the pool contains integrated preservation capacity to provide rebuild locations for potential drive failures. This approach simplifies management, because individual hot spares no longer need to be planned or managed. The approach also greatly improves the time for rebuilds, if required, and enhances volume performance during a rebuild, as opposed to traditional hot spares.

When a drive in a DDP fails, the D-pieces from the failed drive are reconstructed to potentially all other drives in the pool using the same mechanism normally used by RAID 6. During this process, an algorithm internal to the controller framework verifies that no single drive contains two D-pieces from the same D-stripe. The individual D-pieces are reconstructed at the lowest available LBA range on the selected disk drive.

In Figure 6, disk drive 6 (D6) is shown to have failed. Subsequently, the D-pieces that previously resided on that disk are recreated simultaneously across several other drives in the pool. Because there are multiple disks participating in the effort, the overall performance impact of this situation is lessened, and the length of time needed to complete the operation is dramatically reduced.

Figure 6) Dynamic Disk Pool drive failure.



When multiple disk failures occur within a DDP, priority for reconstruction is given to any D-stripes missing two D-pieces to minimize data availability risk. After those critically affected D-stripes are reconstructed, the remainder of the necessary data is reconstructed.

From a controller resource allocation perspective, there are two user-modifiable reconstruction priorities within DDP:

- Degraded reconstruction priority is assigned to instances in which only a single D-piece must be rebuilt for the affected D-stripes; the default for this value is high.
- Critical reconstruction priority is assigned to instances in which a D-stripe has two missing D-pieces that need to be rebuilt; the default for this value is highest.

For very large disk pools with two simultaneous disk failures, only a relatively small number of D-stripes are likely to encounter the critical situation in which two D-pieces must be reconstructed. As discussed previously, these critical D-pieces are identified and reconstructed initially at the highest priority. Doing so returns the DDP to a degraded state quickly so that further drive failures can be tolerated.

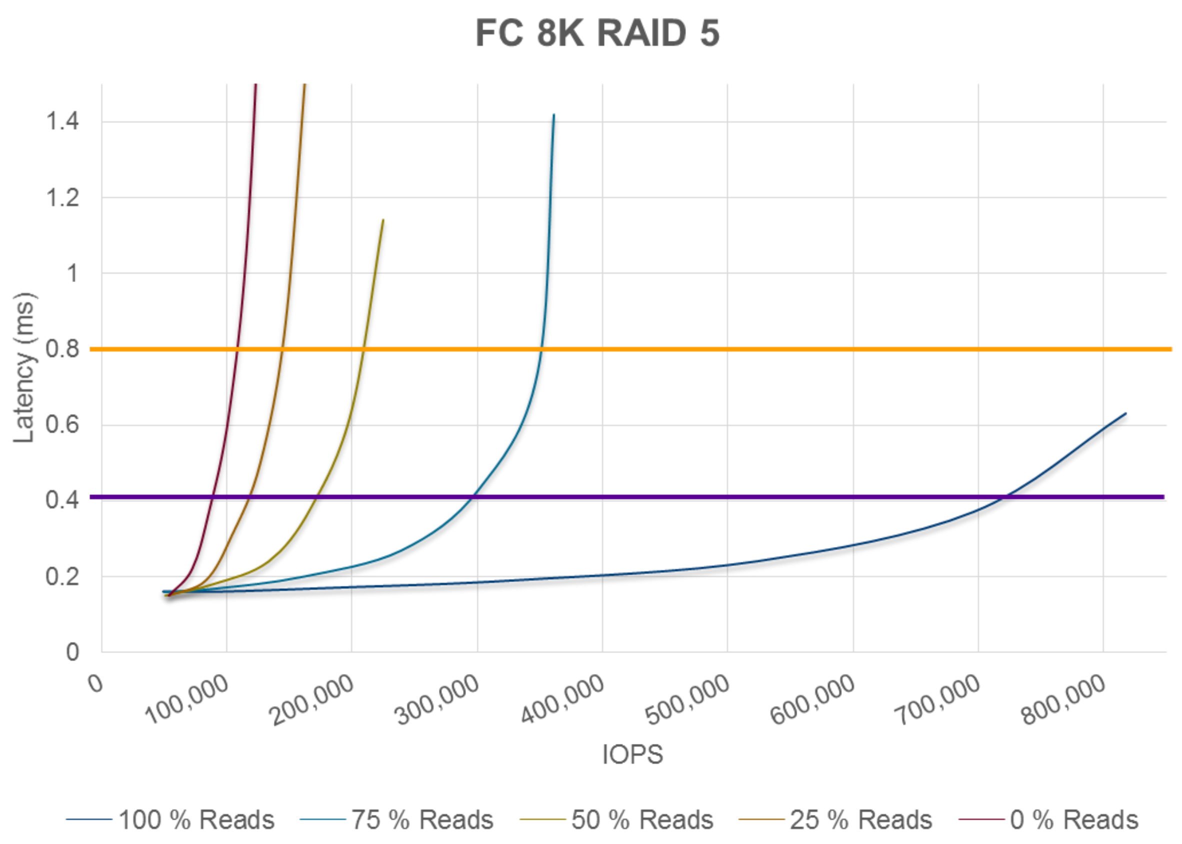
In addition to improving rebuild times and providing superior data protection, DDP can also greatly improve the performance of the base volume when under a failure condition compared to the performance of traditional volume groups.

3.3 Performance

An E5600 with SSD drives installed can perform at very high levels, in both IOPS and throughput, while still providing extremely low latency. In many MongoDB deployments, internal SSDs located in the server are often deployed to provide the level of performance required of an expected high-performance NoSQL cluster. The E5600, through its ease of management, higher degree of reliability, and exceptional performance, can meet the extreme performance requirements expected when a document is not located in the memory of a MongoDB cluster server.

An E5600 is capable of providing over 800,000 8KB random read IOPS at less than 800µs average response time in a RAID 5 configuration, as shown in Figure 7.

Figure 7) Performance of the E5600 using 48 SSD drives.



Many factors can affect the performance of the E5600, including different volume group types or the use of DDP, the average I/O size, and the read versus write percentage provided by the attached servers. Figure 7 also provides performance graphs across various read percentages for the same RAID 5 8KB I/O size with a Fibre Channel host interface.

4 MongoDB and E5600 Testing

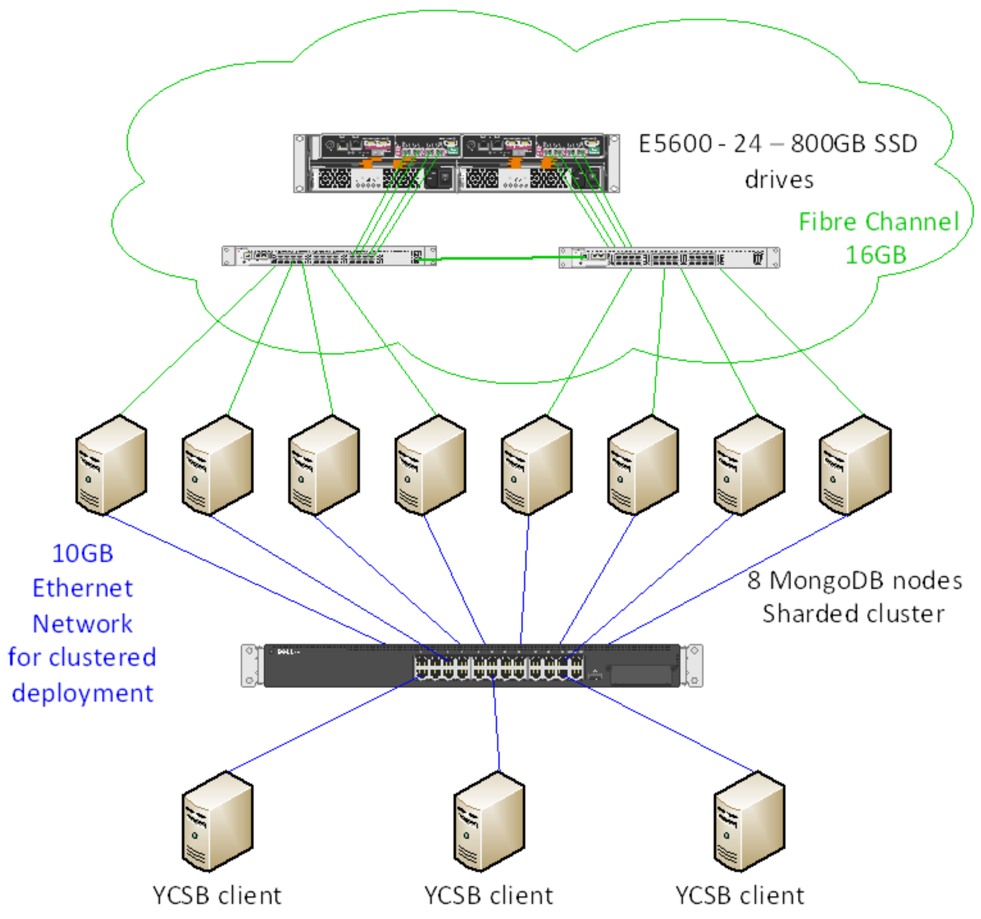
NetApp recently tested a simulated MongoDB cluster environment with both E-Series and commodity servers configured in a MongoDB cluster of nodes using sharding to replicate data throughout the cluster. This configuration enabled testing the E-Series compared to the commodity server DAS for the indexing and search functions a MongoDB cluster requires. The server hardware was chosen following recommendations from the MongoDB reference architecture system requirements. The MongoDB cluster server hardware used is listed in Table 2.

Table 2) MongoDB cluster server hardware.

MongoDB Cluster	Qty	Type	CPU	CPUs	Cores/CPU	Speed	RAM	OS
MongoDB nodes	8	Dell 730xd	E2-2670 v3	2	8	2.3 GHz	128GB	CentOS 7
YCSB client nodes	3	Dell 730	E2-2670 v3	2	8	2.3 GHz	128GB	CentOS 7

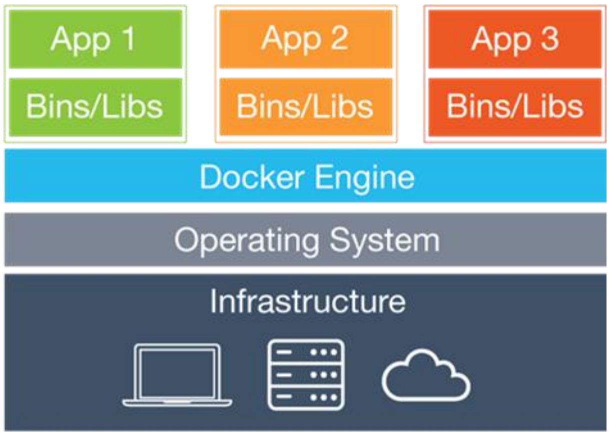
Figure 8 shows the diagram for the E-Series configuration used for testing: an E5600 with 24 800GB SSD drives configured as a DDP using SANtricity 11.25. The pool is then shared across the MongoDB server nodes.

Figure 8) E-Series and MongoDB cluster configuration.


E5600 - 24 – 800GB SSD drives
Fibre Channel 16GB
10GB Ethernet Network for clustered deployment
8 MongoDB nodes Sharded cluster
YCSB client
YCSB client
YCSB client

Docker containers are used to install the primary and secondary MongoDB instances on each server used in the test. The use of Docker containers allows the MongoDB application and all of its dependencies, but shares the kernel with other containers. They run as an isolated process in user space on the host operating system. They’re also not tied to any specific infrastructure: Docker containers run on any computer, on any infrastructure, and in any cloud. This arrangement is advantageous for orchestration software products to deploy the MongoDB containers to servers. In our test environment, the containers share access to the E-Series storage for the MongoDB database volumes. The Docker container architecture is depicted in Figure 9 (source: Docker, nd).

Figure 9) Docker container architecture.



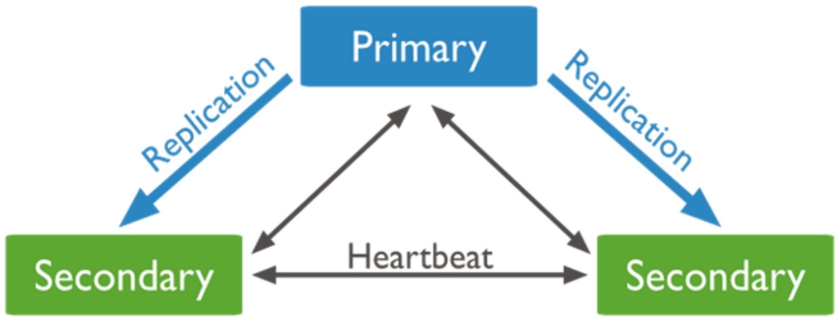
For more information about the Docker containers, see the Docker website.

4.1 Building the MongoDB Sharded Cluster

MongoDB Deployment

MongoDB configured in a clustered environment uses a primary server to perform the heavy lifting for database operations. In the cluster, there are secondary nodes to replicate copies of data in the database. This configuration allows the cluster to remain in operation in the case a primary server node is unavailable. In a sharded cluster, the copies are spread throughout the servers to spread the data out to separate the data copies. A typical MongoDB cluster as shown in Figure 10 has three or more copies of data that is available for database operations. A cluster setup for multiple copy replication is known as a replica dataset.

Figure 10) MongoDB cluster replication.



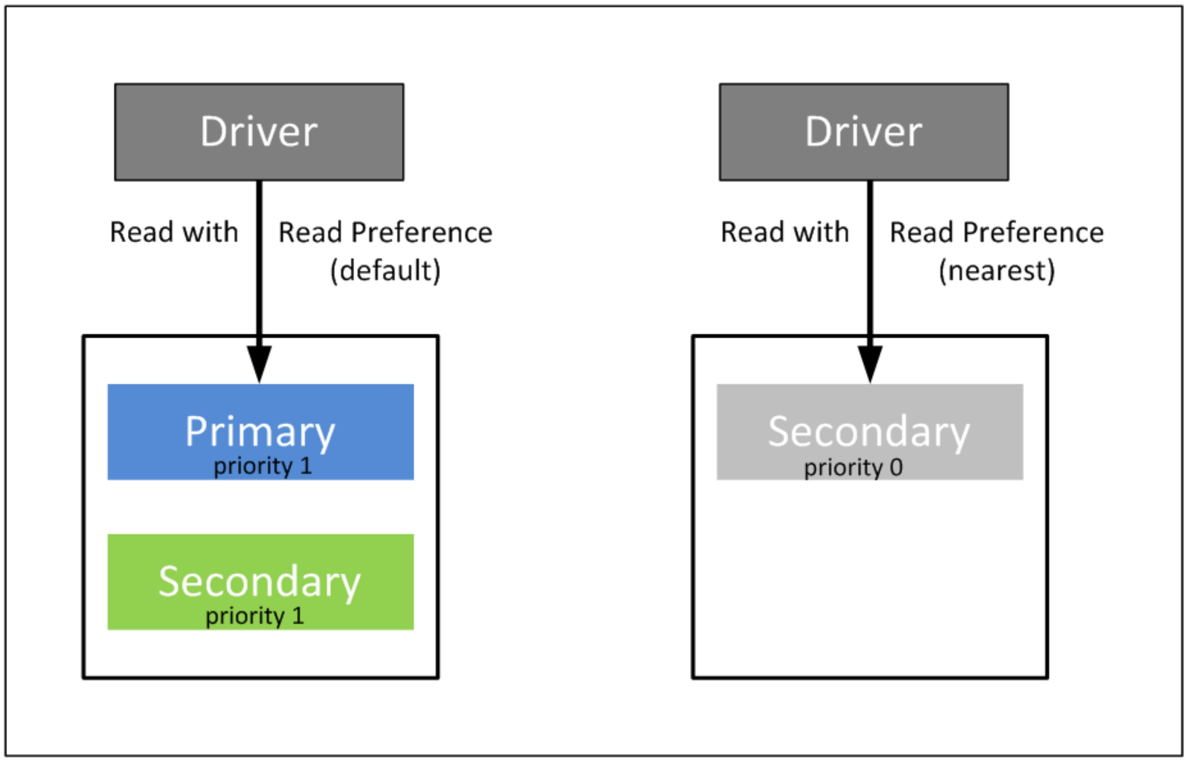
MongoDB Cluster Sharding

Sharding (horizontal scaling) divides the dataset and distributes the data over multiple node groups, or shards. Each shard is an independent database, and collectively, the shards make up a single logical database. A shard is a replica set, and thus it usually consists of multiple nodes.

We consider two major deployment types available for an eight–physical node cluster below. MongoDB configuration servers are kept on separate servers, which are not shown in the drawings.

Read preference describes how MongoDB clients route read operations to the members of a replica set. By default, an application directs its read operations to the primary member in a replica set, as shown in Figure 11.

Figure 11) Read preferences.



However, MongoDB drivers support up to five read preference modes, as shown in Table 3.

Table 3) MongoDB cluster replica read preference.

Read Preference Mode	Description
Primary	Default mode. All operations read from current replica dataset.
Primary preferred	In most situations, operations read from the primary. If not available, operations read from the secondary.
Secondary	All operations read from the secondary members of the replica set.
Secondary preferred	In most situations, operations read from the secondary. If not available, operations read from the primary.
Nearest	Operations read from the member of the replica set with the least network latency, regardless of the replica member type of primary or secondary.

Deployment Models

In the NetApp E-Series and MongoDB deployment testing, two major cluster models are used. The first deployment model has 8 primaries and 8 secondaries distributed across 8 cluster servers (hosts 1–8), as shown in Figure 12. This deployment may be horizontally scaled, as shown in Figure 13. This deployment type was used only for the E5600 all-SSD drive configuration test.

Figure 12) MongoDB deployment: 8 primaries and 8 secondaries.

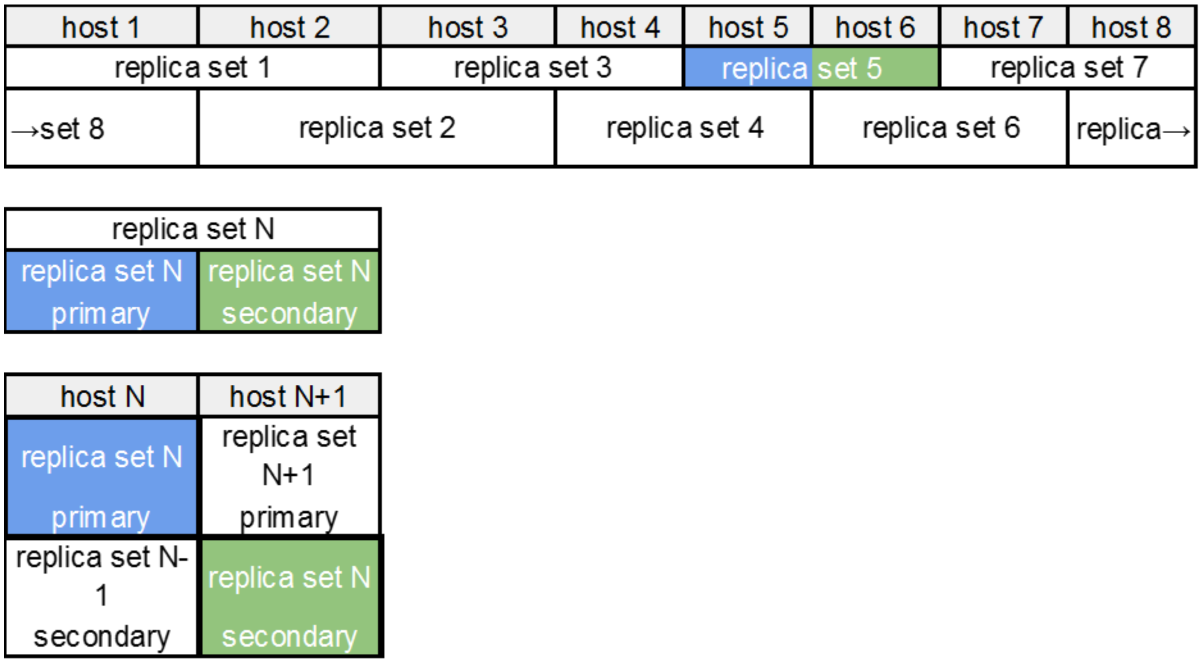
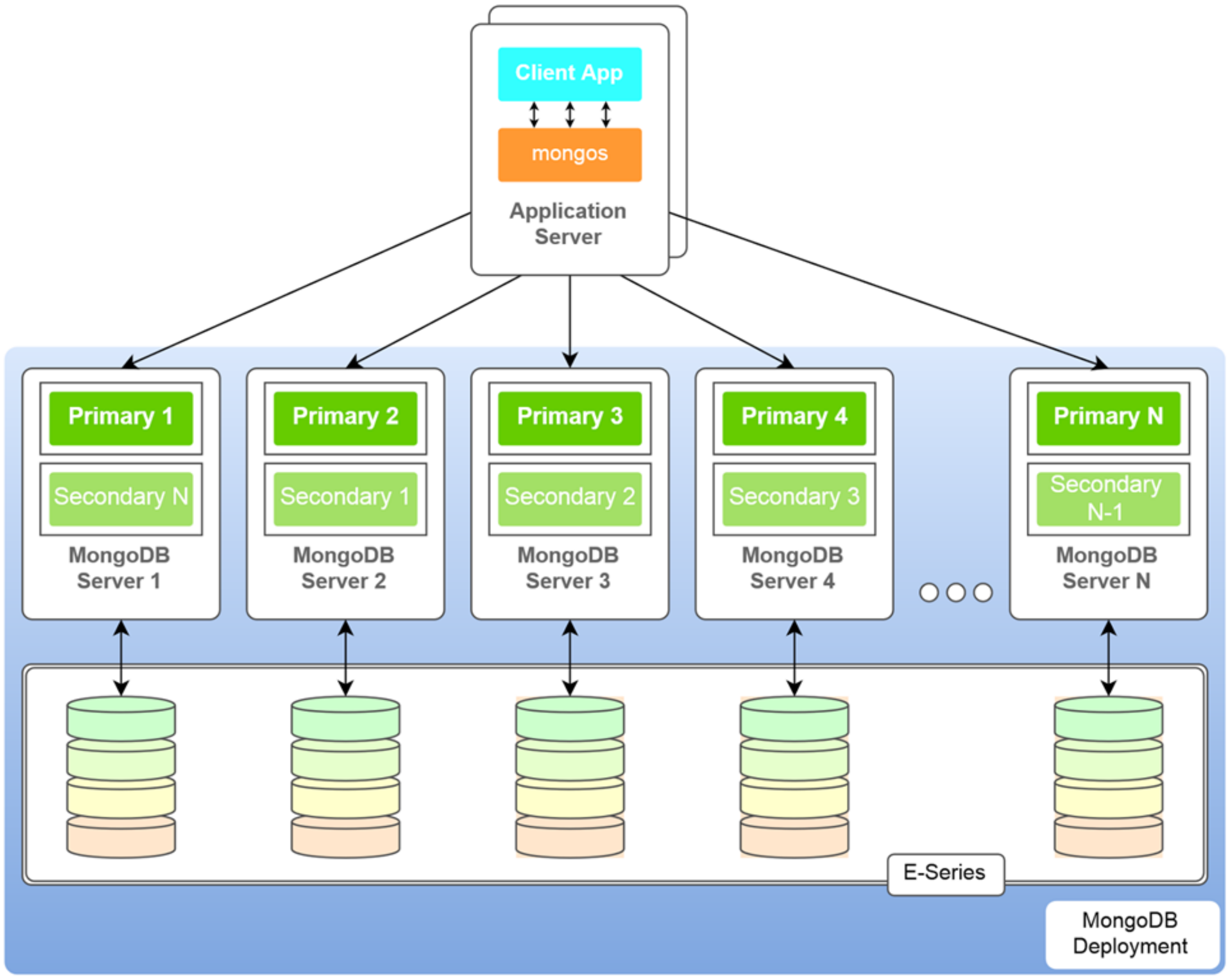
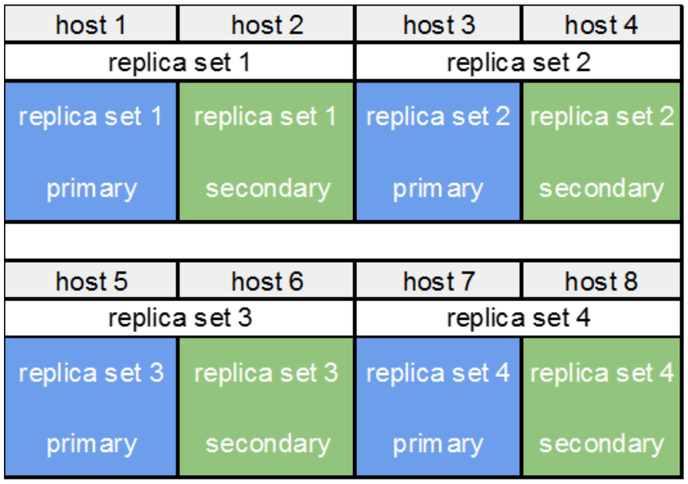


Figure 13) MongoDB cluster nodes: 8 primaries and 8 secondaries.



The second deployment model of 4 primaries and 4 secondaries is the most popular MongoDB deployment type with sharding replication. There is a single mongod process per physical server, as depicted in Figure 14, showing the location of the replica set data distributed in the cluster.

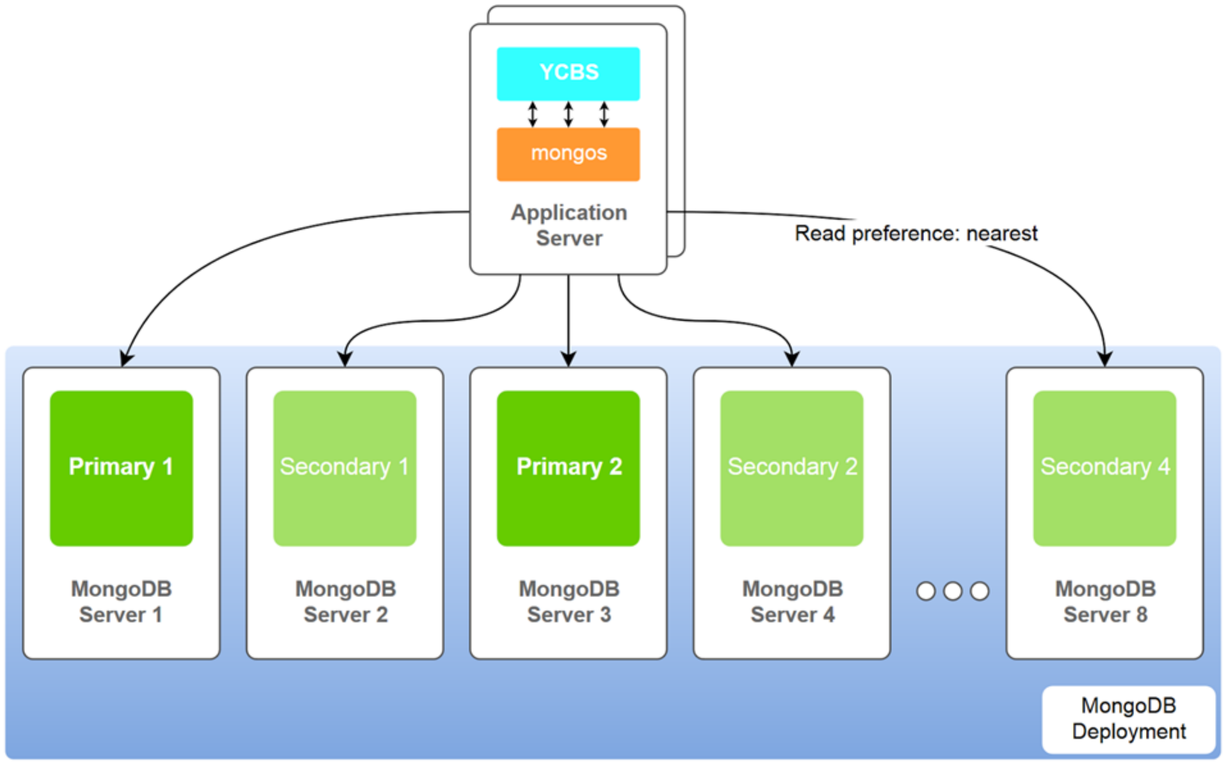
Figure 14) MongoDB sharding deployment: 4 primaries and 4 secondaries.



host 1	host 2	host 3	host 4
replica set 1		replica set 2	
replica set 1 primary	replica set 1 secondary	replica set 2 primary	replica set 2 secondary
host 5	host 6	host 7	host 8
replica set 3		replica set 4	
replica set 3 primary	replica set 3 secondary	replica set 4 primary	replica set 4 secondary

Figure 15 depicts the server deployment using containers of 4 primary MongoDB server nodes and 4 secondary MongoDB server nodes.

Figure 15) MongoDB cluster nodes: 4 primaries and 4 secondaries.



The second deployment model is used in the E-Series E5600 configured for heterogeneous storage with HDDs plus SSD drives for caching.

4.2 Performance Testing

Yahoo Cloud Server Benchmarking (YCSB)

In the test environment, MongoDB cluster performance was measured using the Yahoo Cloud Server Benchmarking tool. A particular test is defined by the following choices:

- YCSB workload type: which create, read, update, delete (CRUD) operation to perform on the document and which document is selected as the target
- Document size: document size in bytes, balance between data and metadata
- Total number of documents: total data size in cluster, percentage of data that fits in memory
- Write durability options: how many nodes have the result of the operation persisted in memory and on disk before it is considered successful

Database performance is defined by the speed at which a database computes basic CRUD operations. A basic operation is an action performed by the workload executor that drives multiple client threads. Each thread executes a sequential series of operations by making calls to the database interface layer, both to load the database (the load phase) and to execute the workload (the transaction phase). The threads throttle the rate at which they generate requests, so that we may directly control the offered load against the database. In addition, the threads measure the latency and achieved throughput of their operations and report these measurements to the statistics module.

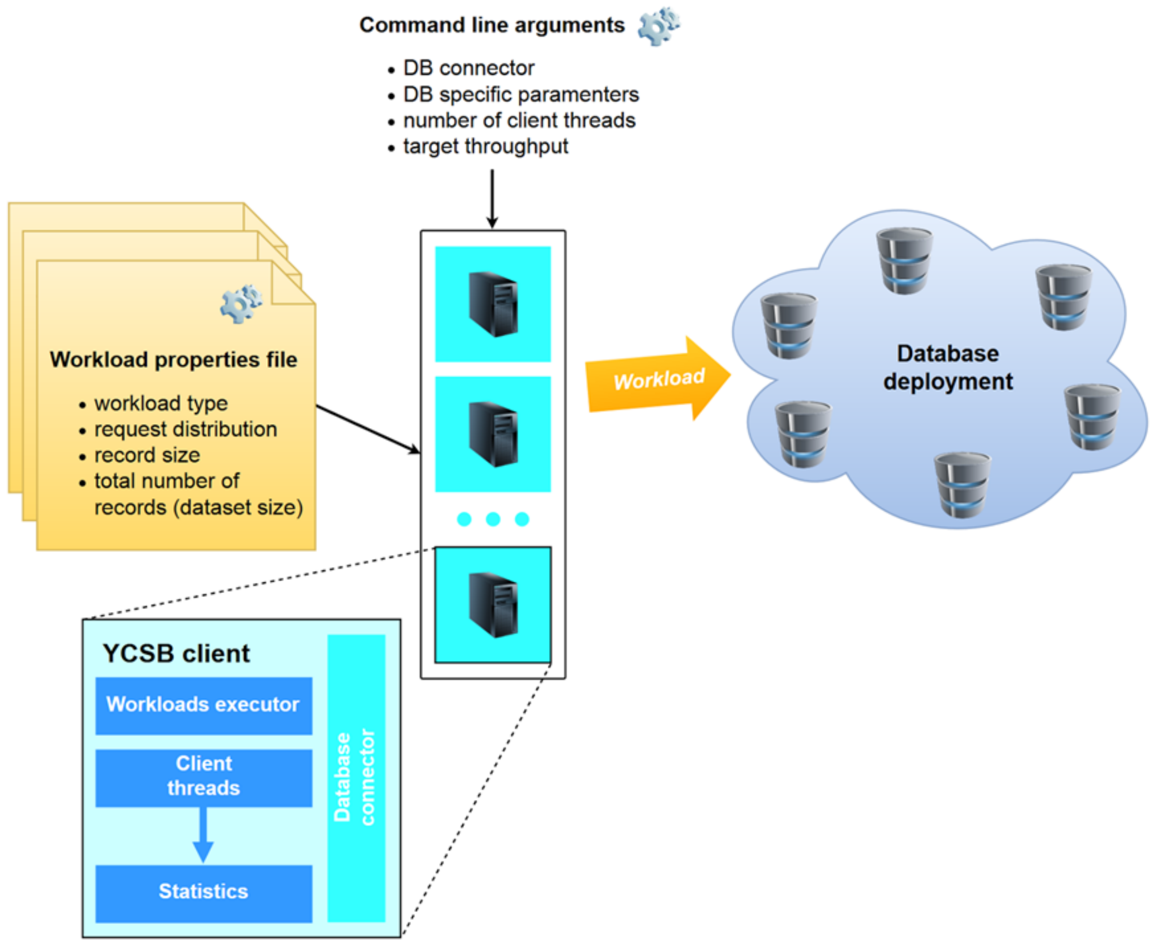
A very common use case for applications deploying MongoDB is environments where heavy read operations are the most prevalent. YCSB was configured during all tests with the two following workloads to documents within the database:

- 100% read
- 95% reads, 5% updates

These tests were conducted on 8 MongoDB cluster servers. Performance and scalability were tested with a compaction sequence introduced before increasing the number of cluster servers after the update tests were complete.

The typical YCSB workflow into a cluster is depicted in Figure 16.

Figure 16) YCSB deployment diagram.



The MongoDB cluster with YCSB is tested with a document size available and chosen for the configurations.

Example of the document size parameter variations:

- Small document size (<1kb row size)
- Big document size (~30kb row size)

A primary key identifies each document, which is a string, such as –user234123. Fields are named field0, field1, and so on. For our testing with YCSB, a 2K document size has been chosen for a normal size document in a read-heavy MongoDB deployment.

Number of documents in MongoDB for the tests is calculated using the following formula:

N documents = RAM * factor ÷ document size, where the factor is determined during the testing

The factor is expected to be high in order to achieve better disk utilization. Using 128GB of RAM in each MongoDB server node, the document size to be tested is 1.2 billion and 600 million for the deployment strategies.

All-SSD Testing

Configuration

A dataset containing 1.2 billion documents (2KB each) was loaded into a single MongoDB sharded collection. An 8-node cluster was split into 8 shards for the configuration. Each shard was a replica set of 2 nodes. After that, the YCSB client machines run 100% read workloads, picking uniformly random documents out of the whole dataset. The document read request is served by an arbitrary node (nearest read preference).

This storage configuration was tested with the server deployment using containers of 8 primary MongoDB server nodes and 8 secondary MongoDB server nodes, as described previously.

Results

Figure 17 shows the MongoDB dataset of 1.2 billion documents measuring throughput (ops/sec) of the documents per the number of YCSB client threads.

Figure 17) All-SSD drives throughput.

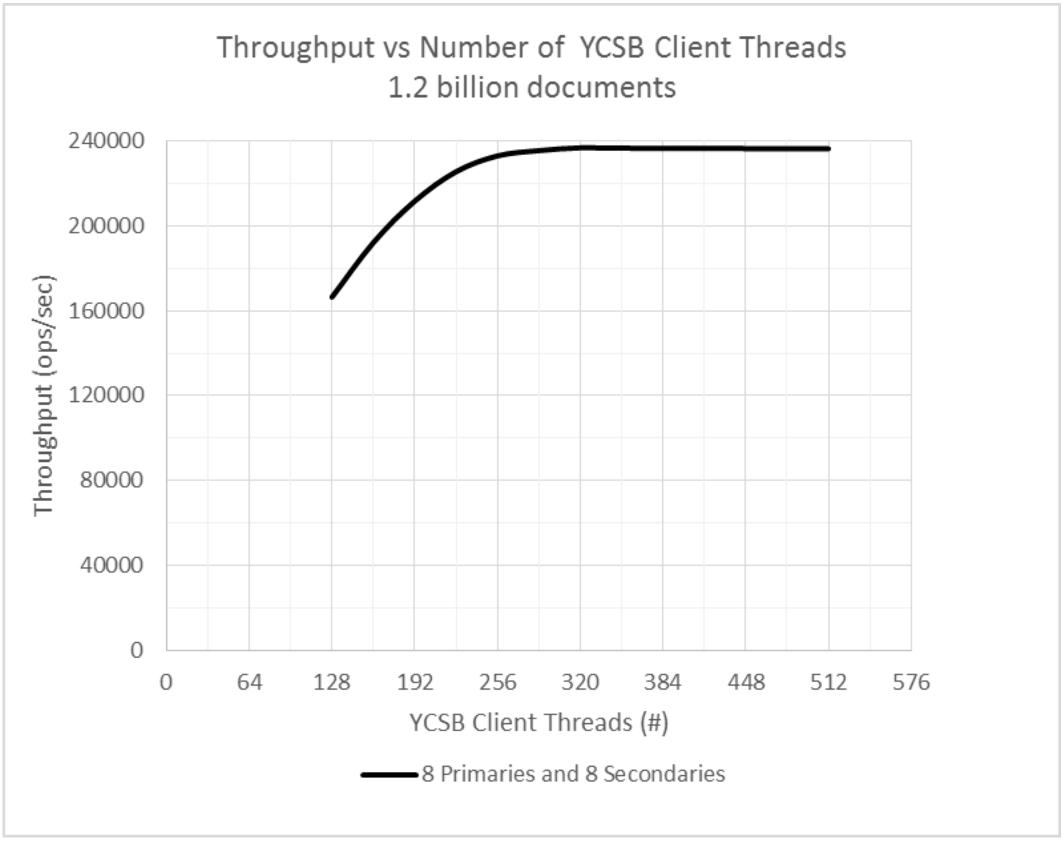
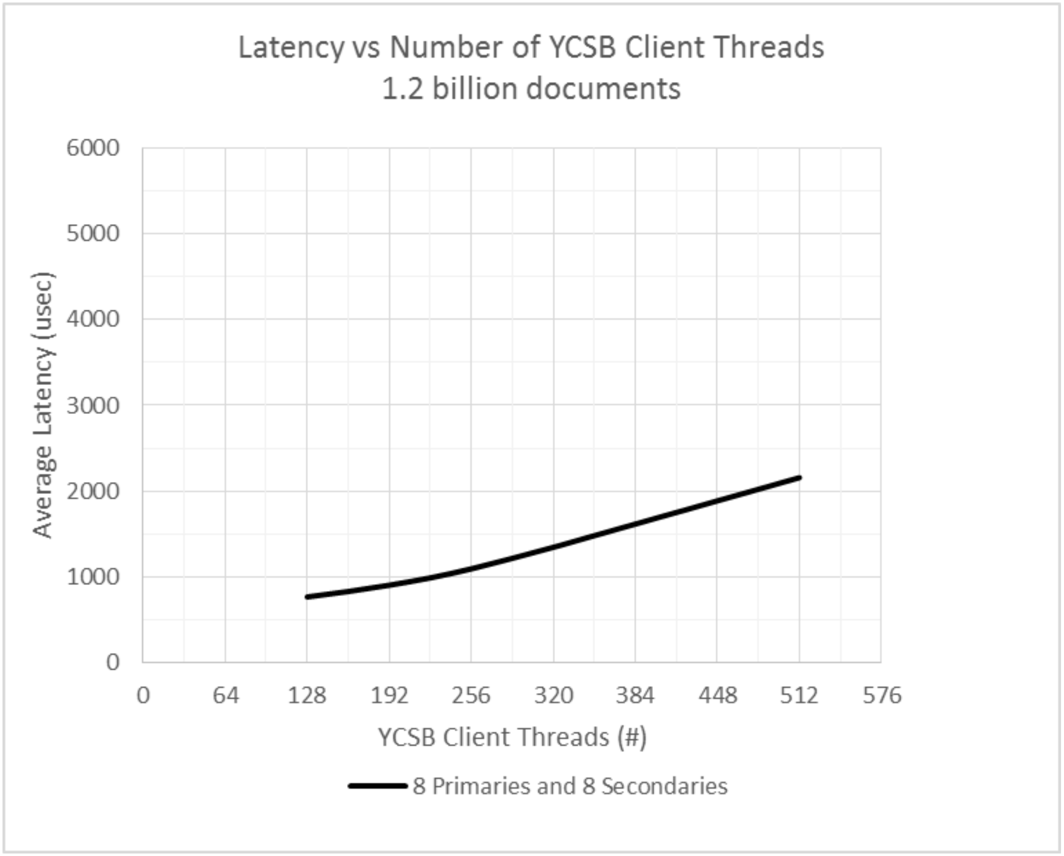


Figure 18 shows the MongoDB dataset of 1.2 billion documents measuring latency (in microseconds [µsec]) of the documents per the number of YCSB client threads.

Figure 18) All-SSD drives latency.



SSD Read Cache Testing

Configuration

The SSD read cache feature is a candidate to improve HDD performance to closely approximate the performance behavior seen by an all-SSD configuration.

A dataset containing 1.2 billion documents (2KB each) was loaded into a single MongoDB sharded collection. An 8-node cluster was split into 4 shards for the configuration. Each shard was a replica set of 2 nodes. After that, the YCSB client machines run 100% read workloads picking uniformly random documents out of the active dataset. The document read request is served by an arbitrary node.

This storage configuration was tested with the server deployment using containers of 4 primary MongoDB server nodes and 4 secondary MongoDB server nodes, as described previously.

For the all-SSD deployment, the E5600 under test was configured with a single DDP of 24 800GB SSDs, with a pool preservation capacity of 2 drives offering ~10TiB of usable capacity. Eight 1.2TB volumes were created, with one volume per each MongoDB cluster server node.

For the SSD read cache deployment, the E5600 was configured with 4 800GB SSD drives enabled for SSD read cache with 20 10K rpm SAS HDD configured as a DDP with a pool preservation capacity of 2 drives offering ~15TiB of usable capacity. Eight 1.5TB volumes were created, with one volume per each MongoDB cluster server node.

The active dataset (a fraction of the whole dataset) was varied from 1.2 billion documents (full dataset) to 600 million (half of the dataset) during the tests. The smaller the active dataset, the better it fits into server RAM + SSD cache. The RAM and SSD caching impact on the overall performance is tested by comparing achieved throughput and latency values for RAM+SSD+HDD (SSD read cache enabled) with the values measured for a RAM+HDD configuration (SSD read cache disabled) E-Series configuration.

RAM indicates the cache used by MongoDB, SSD stands for E-Series SSD cache enabled, and HDD denotes the largest (in terms of capacity) storage device type. The cache was warmed up for 10 to 12 hours before each test.

The datasets tested were:

- Active dataset is equal to the whole dataset and contains 1.2 billion documents.
- Active dataset is half of the whole dataset and contains 600 million documents in the active dataset, resulting in better SSD cache efficiency (when used).

The storage configuration was:

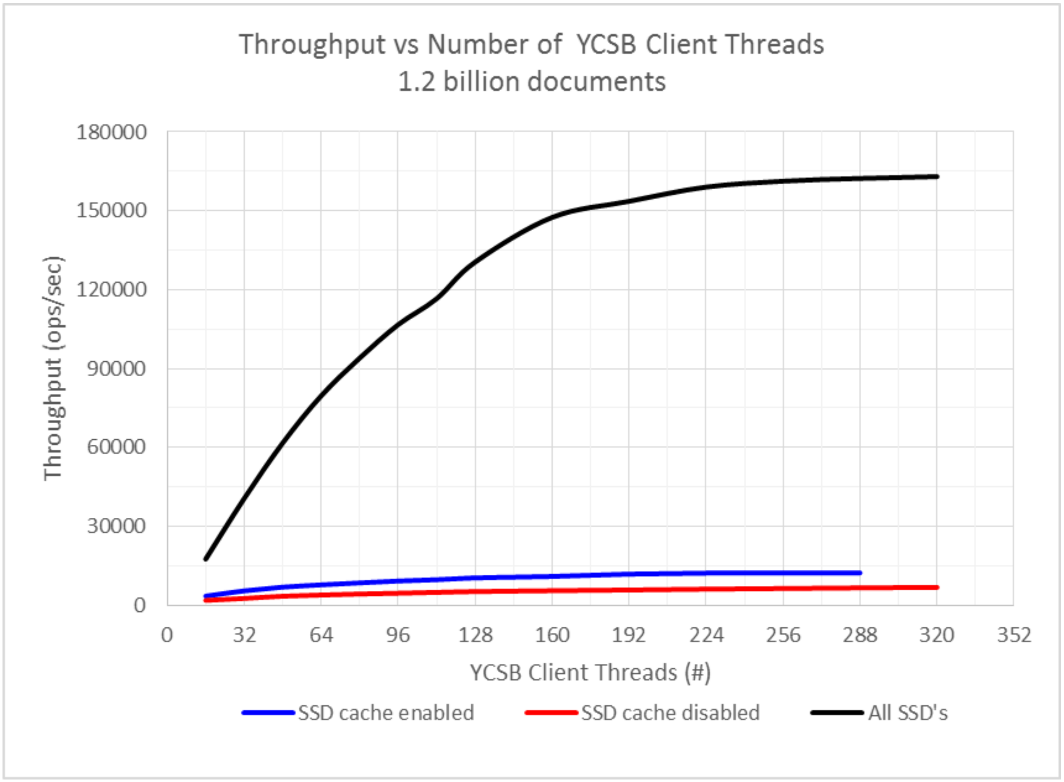
- All-SSD deployment as the practical performance upper bound
- SSD read cache deployment using RAM + SSD + HDD as the point of study
- All-HDD deployment, with all the documents located in RAM + HDDs

Finally, the results are compared between all the storage configurations and different datasets.

Results

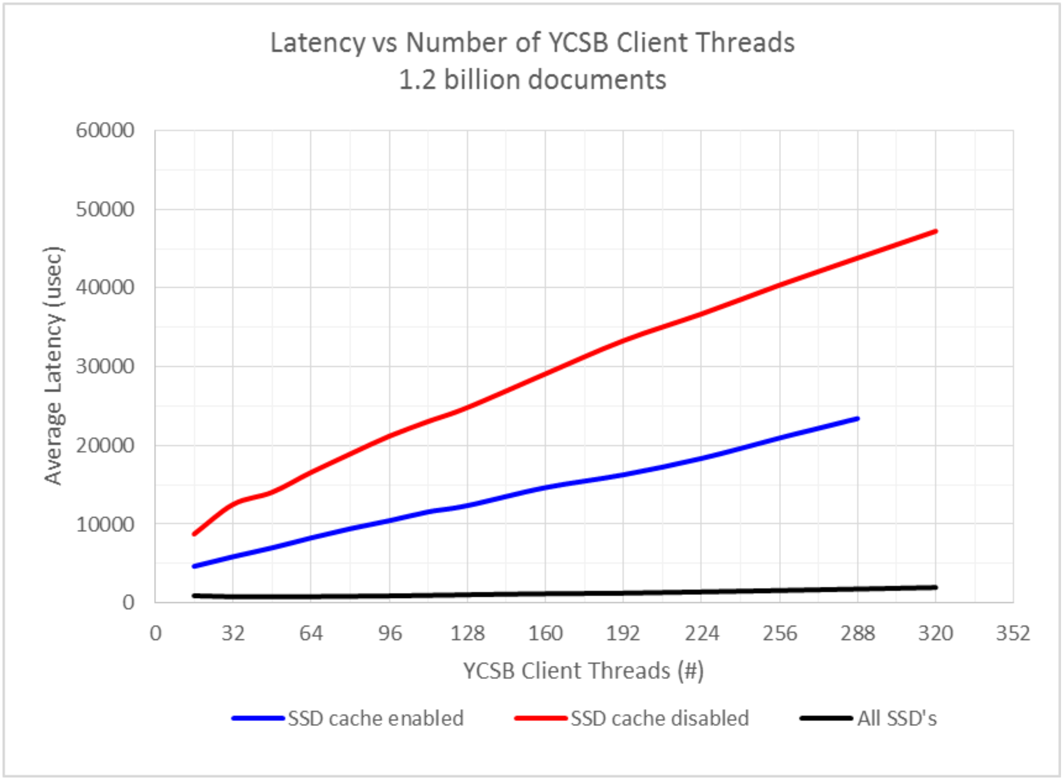
In Figure 19, the MongoDB dataset of 1.2 billion documents measuring throughput (ops/sec) of the documents per the number of YCSB client threads, comparing the all-SSD, SSD read cache enabled, and SSD read cache disabled in the 4 primary and 4 secondary deployment.

Figure 19) Throughput comparison for 1.2 billion documents.



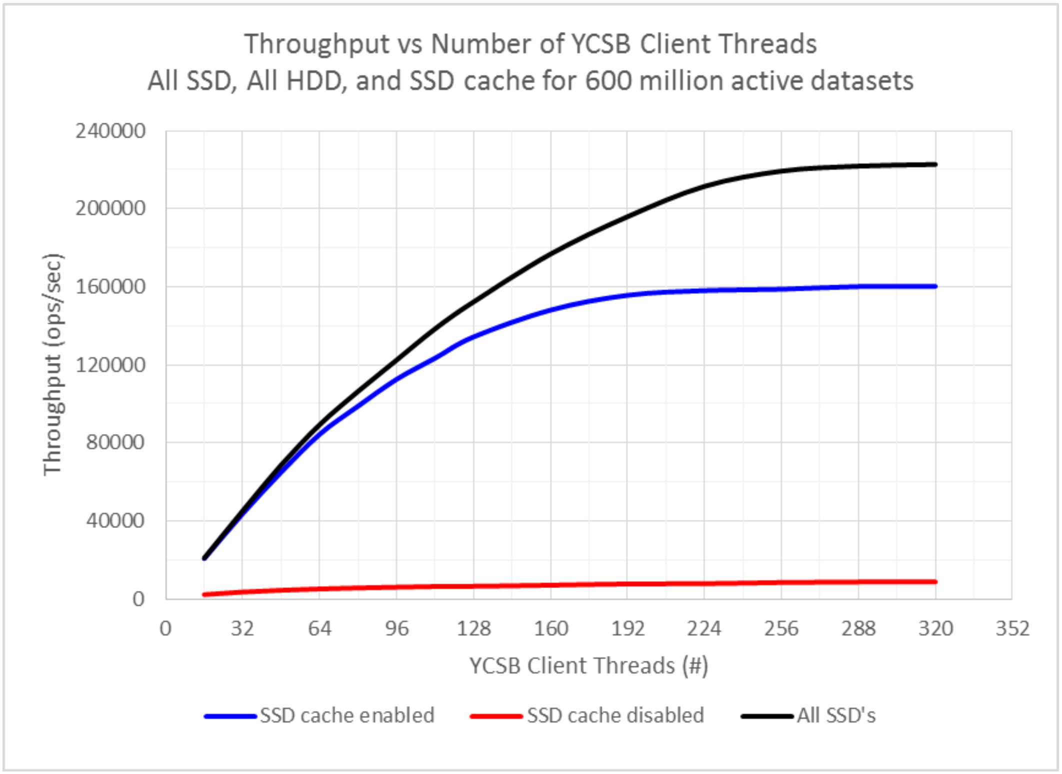
In Figure 20, the MongoDB dataset of 1.2 billion documents measures latency (in µsec) of the documents per the number of YCSB client threads, comparing the all-SSD, SSD read cache enabled, and SSD read cache disabled in the 4 primary and 4 secondary deployment.

Figure 20) Latency comparison for 1.2 billion documents.



In Figure 21, the MongoDB active dataset of 600 million documents measures throughput (ops/sec) of the documents per the number of YCSB client threads, comparing the all-SSD, SSD read cache enabled, and SSD read cache disabled in the 4 primary and 4 secondary deployment.

Figure 21) Throughput comparison for 600 million documents.



In Figure 22,Figure 21 the MongoDB active dataset of 600 million documents measures latency (µsec) of the documents per the number of YCSB client threads, comparing the all-SSD, SSD read cache enabled, and SSD read cache disabled in the 4 primary and 4 secondary deployment.

Figure 22) Latency comparison for 600 million documents.

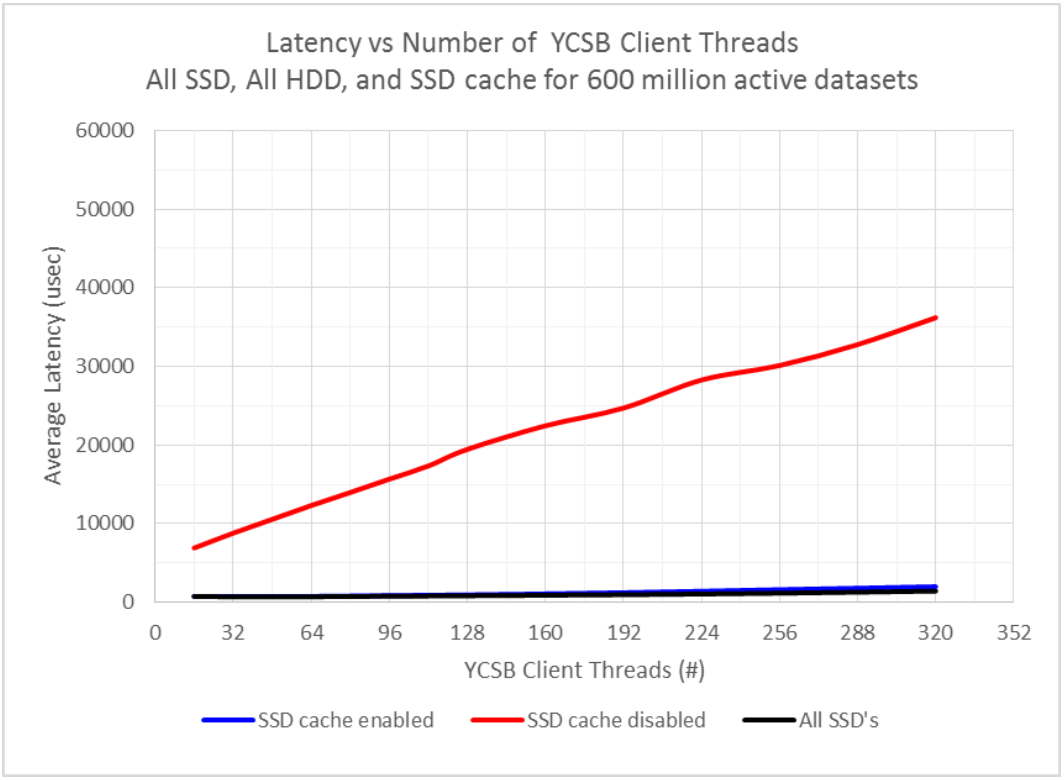


Figure 23 and Figure 24, respectively, show a comparison of different sized MongoDB active datasets for the SSD read cache configuration. Sizes of 1.2 billion, 1 billion, 800 million, and 600 million were tested measuring throughput (ops/sec) and latency (µsec) of the documents per the number of YCSB client threads.

Figure 23) Throughput for various numbers of documents.

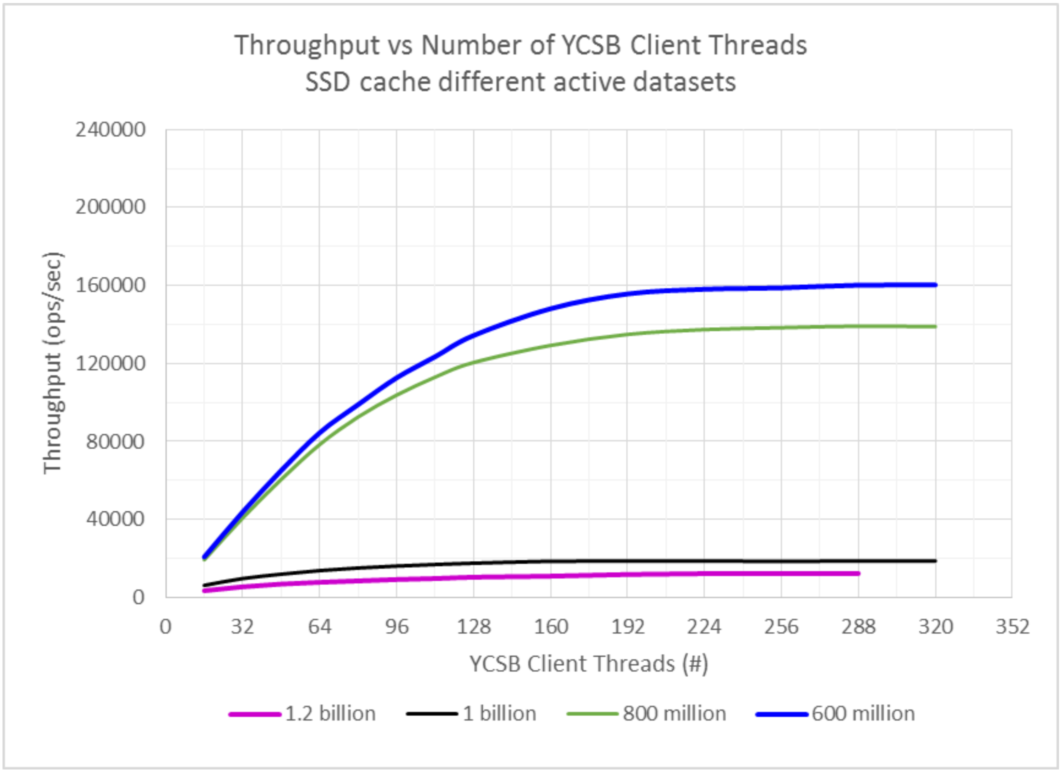
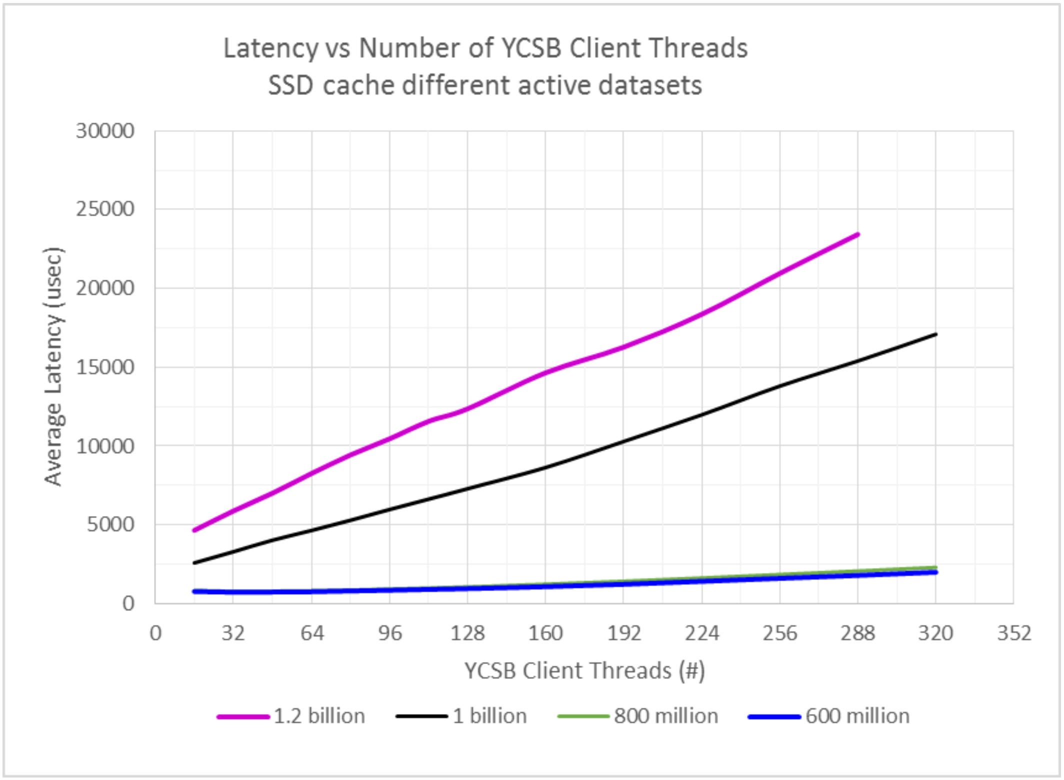


Figure 24) Latency for various numbers of documents.



4.3 MongoDB Node Failure Testing

In addition to baseline performance testing, additional testing was conducted on the E5600 under failure conditions, including controller failure and drive failure.

In a typical MongoDB deployment using only internal drives within the server, there is no redundancy at the RAID controller level. A failure of the internal controller would be similar to a complete server failure and would require rebalancing of the data across the remaining cluster servers. With the E5600 and the redundancy provided through the dual redundant controller design, no rebalancing is required because all volumes simply transition to the remaining controller. A test was conducted in which a controller within the E5600 was failed under an active workload.

The following set of tests was designed to show E-Series disk failure recovery capabilities. Two different storage deployments were tested. The first deployment was the all-SSD installation, and the second one was built of enterprise-level SAS HDDs and an SSD read cache, as described previously. Our goal was to estimate the MongoDB client performance negative impact of a single disk drive failure under heavy read workload.

Configuration

A dataset containing 1.2 billion documents (2KB each) was loaded into a single MongoDB sharded collection. An 8-node cluster was split into 4 shards. Each shard was a replica set of 2 nodes. After that, YCSB client machines run a 100% read workload picking uniformly random documents out of the active dataset. The active fraction of the whole dataset was varied from 600 million (half of the dataset) for HDD+SSD deployment to 1.2 billion documents (full dataset) for pure-SSD storage deployment.

For the SSD read cache configuration, the active datasets tested were located in RAM + SSD at 100% of the device and 50% in HDDs. This result is for 600 million documents in the dataset, resulting in less data in the HDDs.

Results

In Figure 25, the E5600 all-SSD deployment, a single SSD was failed on the 22-minute mark. The system was fully rebalanced within 50 minutes. After that, on the 89-minute mark, the failed drive was replaced, and the rebalance operation was started again and fully recovered at approximately the 148-minute mark.

Figure 25) SSD drive failure test in DDP.

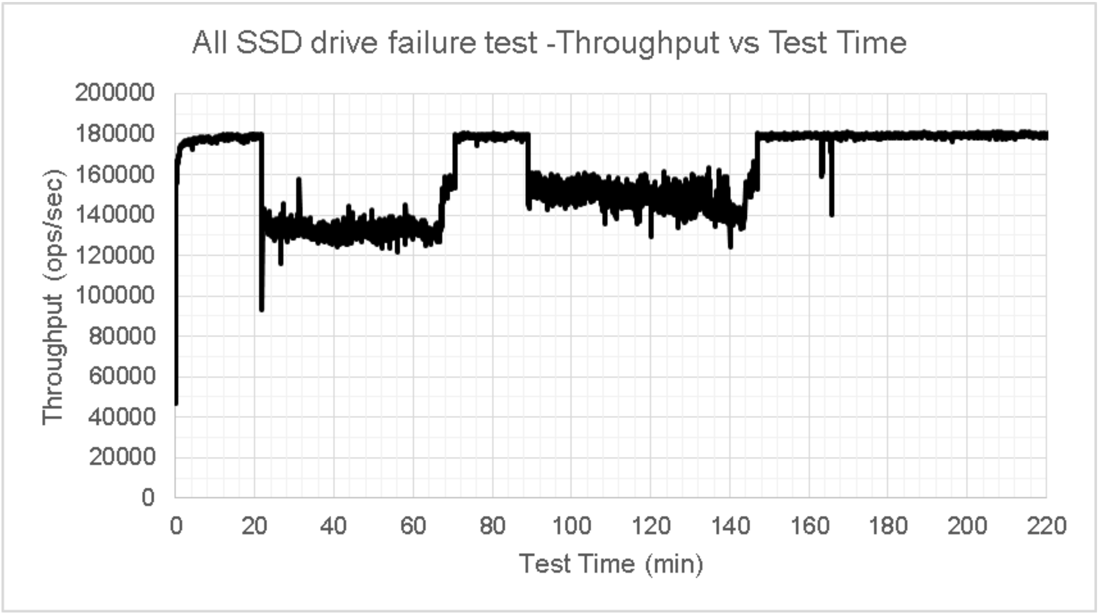
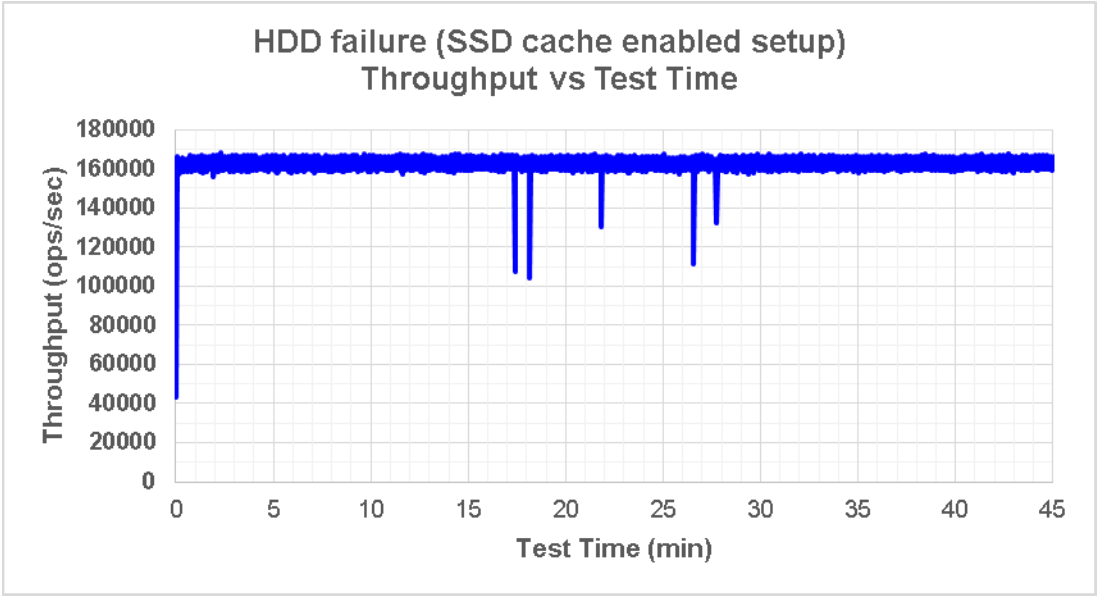


Figure 26 shows an HDD failure with SSD read cache enabled. A hard disk drive was failed on the 17-minute mark. The system was fully rebalanced in 2.5 hours. During the rebalance process, short performance drawdowns occur as shown. SSD cache utilization approached 100% consistently.

Figure 26) HDD failure with SSD read cache enabled.



The E-Series configured for both E5600 all-SSD drives and SSD cache enabled in a hybrid HDD plus SSD configured handle drive failures without a disruption of I/O in the MongoDB cluster. The DDP in each instance rebalances the disk pool while allowing an administrator time to replace the failed disk. The DDP again rebalances the disk pool with no disruptions, and the cluster remains in an optimal state for operations.

5 Summary

The NetApp E-Series provides a number of significant advantages over internal direct-access storage (DAS) for MongoDB deployments. The E5600 configured with either all-SSD or SSD read cache enables high performance and availability of MongoDB data, as seen in the test results using YCSB client testing simulation of real customer workloads.

NetApp SANtricity provides compelling customer solutions for read-heavy workloads in a clustered MongoDB environment. These advantages include Dynamic Disk Pools, which provide rapid rebuild under disk failures so the MongoDB database degradation is minimal. Dynamic Disk Pools can be increased in size dynamically to provide additional capacity and/or performance as required for the MongoDB cluster environment for both a current deployment and when the cluster needs to scale out to meet additional use case requirements.

The NetApp E-Series integrated architecture for MongoDB is optimized for node storage balance, reliability, performance, storage capacity, and density. From an administrative standpoint, the E-Series offers simplified storage management with a centralized user interface. This solution enables new volumes, volumes groups, and Dynamic Disk Pools to be created easily and provisioned immediately for use by the MongoDB cluster servers.

The NetApp E-Series is also flexible enough to allow the expansion of the SSD read cache by adding more SSD drives up to 5TB (the tested configuration was 3.2TB) for greater performance. The NetApp integrated design employs the managed DAS model, with higher scalability and performance. It also prevents unnecessary purchases of compute nodes for storage-intensive workloads for MongoDB environments that need to grow to meet organizational requirements.

References

NetApp MongoDB Certification

MongoDB Use Cases

MongoDB Architecture Guide

MongoDB Resource Center

MongoDB reference architecture system requirements

Docker Containers

TR-4494: Introduction to E-Series E5600 Hardware Using SANtricity 11.25

TR-4099: NetApp SANtricity SSD Cache for E-Series

www.netapp.com