Benchmarking DataStax Enterprise (DSE) on AWS: A Guide to Raising Performance and Reducing Costs

Summary

Kenzan is a software engineering firm that specializes in building scalable, data-driven solutions. To that end, Kenzan often leverages DataStax Enterprise (DSE) for customers that have high expectations for both performance and value. That means making careful, informed choices when designing the cloud infrastructure that applications run on.

Controlling costs is always a priority for businesses that rely on AWS for running DSE. With AWS, it can be tempting to solve performance issues simply by adding more resources until the problem goes away. However, this “solution” results in increased costs which quickly add up over time.

Carefully tailoring your AWS infrastructure for DSE and your business provides a better way to reduce costs while achieving good performance. The goal of this paper is to use benchmark data to identify key factors that affect DSE cost and performance in AWS. More than that, we hope it encourages you to design and run benchmark tests yourself to determine the best solution for your business needs and budget.

Contents

Introduction **3**

AWS Infrastructure for DSE **3**
- CPU Explained 3
- Memory Explained 4
- Storage Explained 5
  - Elastic Block Store (EBS) 5
  - Ephemeral Storage 6
  - NVMe Ephemeral 6

AWS Best Practices **6**
- Cost 6
  - Calculating for Growth 7
  - Keeping a Quorum 8
  - Balancing Resilience and Performance 8
- DSE on AWS 9

DSE Benchmarks **9**
- Testing Methodology 9
- Guidelines 10
  - Test Multiple Workload Ratios 11
  - Use Large Datasets 11
  - Send Workloads Across Zones 11
  - Take Down Nodes 11

Interpreting Test Results **11**
- CPU Generation and Performance 11
- Data Size and the Page Cache 13
  - Linux Page Cache 14
  - Test: Effects of Caching With a 1 GB Payload 14
  - Test: Effects of Caching With a 25 GB Payload 15
- Storage Type and Throughput 16
  - Test: Comparing EBS and Ephemeral Storage 16
  - Test: General Purpose SSD vs Provisioned IOPS 17
  - The Case For EBS 19
- Raising the Bar with I3 20

Conclusion **21**

Introduction

DataStax Enterprise (DSE) is the always-on data platform powered by the best distribution of Apache Cassandra™. DSE is designed to transact and store large amounts of data distributed in a scale-out fashion across many machines in a cluster, providing scalability and high-availability, all in real time. It includes developer tooling, administration, and monitoring—along with graph, search, and operational analytics—integrated as a comprehensive platform.

The minimum hardware requirements for a DSE server greatly exceed those of smaller, single-machine databases such as MySQL. To run a DSE cluster, most organizations look to a cloud computing platform such as AWS EC2. AWS offers numerous instance types with an array of options for CPU, memory, and storage configuration. Choosing among these options can be a daunting task, and making the wrong choices can mean paying more than expected for performance that's less than desired.

Kenzan is a member of the DataStax Partner Network, working with DataStax to offer data-driven cloud solutions for our mutual customers. In this paper, DataStax and Kenzan outline some of the key factors to consider when building out infrastructure on AWS. We'll also discuss a suite of tests Kenzan conducted to measure the performance of DSE in a variety of AWS EC2 environments with differing hardware configurations.

The test results show that certain EC2 instance options can dramatically affect DSE throughput. However, the results were sometimes surprising, indicating that care must be taken in constructing test data to ensure that benchmark results are meaningful. In the end, we found that there are practical steps you can take to optimize performance with DSE in AWS while keeping costs in check.

AWS Infrastructure for DSE

AWS offers a variety of EC2 instance types, with each type optimized for different computing loads or use cases. Before we dive into AWS benchmarking, it's useful to understand some of the key hardware options that vary between instance types and how these come into play when looking at performance.

CPU Explained

Different EC2 instance types run on different hardware platforms, and these platforms in turn use a variety of Intel Xeon CPUs spanning several CPU generations. Historically, Intel processor generations have followed a "tick-tock" pattern, alternating between a die shrink using a smaller manufacturing process (the "tick") and a new architecture (the "tock"). More recently, Intel has

added extra refresh cycles, particularly on the consumer desktop line, further optimizing the architecture between ticks.

The CPU an instance runs on can potentially make a big difference in how that instance performs—and how much it costs to run. Newer Intel CPU generations feature improved architecture along with more efficient designs that use less power and generate less heat. These factors translate into better performance at lower costs. The table below shows some of the key differences between recent Intel Xeon processor generations.

Table 1: Intel Xeon CPU generations

Generation	Xeon v2	Xeon v3	Xeon v4
Code Name	Ivy Bridge	Haswell	Broadwell
Process	22nm (tick)	22nm (tock)	14nm (tick)
Features	New manufacturing process	Performance improvements	Minor performance improvements; reduced power consumption and better thermals

Let's look at an example that had a direct effect on this paper. We were in the late stages of conducting our testing when Amazon introduced the new I3 instance type. While I2 instances run on Intel Xeon v2 (Ivy Bridge) processors, I3 instances run on Intel Xeon v4 (Broadwell) processors, as do R4 instances.

The newer Broadwell architecture allows the processor to execute more instructions per clock cycle compared to Ivy Bridge. This means the CPU can run more concurrent operations, and it can handle more threads or workers without diminishing returns. In addition, Broadwell includes improvements related specifically to virtualization. Finally, Broadwell consumes less power during operation.

In real-world terms, this means Broadwell CPUs offer better performance at reduced cost. An i2.4xlarge instance costs $3.41/hour to run, while an i3.4xlarge instance costs just $1.25/hour. As we'll see later, the I3 instance delivers significantly faster performance. All of this means that you'll want to choose the newest AWS instance types with the latest Intel Xeon CPUs for your DSE cluster.

Memory Explained

Memory is another key factor in choosing an EC2 instance. Different instance types and sizes offer different amounts of memory (RAM), so it's important to pay close attention. Just because two instances are similar in size doesn't mean they'll have similar amounts of memory. For

example, an i3.2xlarge instance has 61 GB of RAM, while a c4.2xlarge instance has just 15 GB of RAM. That's because I3 instances are optimized for high I/O use cases, while C4 instances are optimized for compute.

The amount of memory an instance has can have a major impact on benchmarking. Given the JVM settings for DSE, the amount of memory designated as heap space is calculated based on the total system memory. By default, the heap size will be a quarter of the total system memory, with a maximum of 8 GB. For example, 32 GB of system memory caps the heap size at 8 GB.

Despite this limit, DSE will see performance gains even when pushing past 32 GB of system memory. This is due to a Linux feature called the Page Cache. When the operating system reads data from disk, that data is also cached in available RAM. The next time that same data is read, the operating system reads the data from the Page Cache rather than disk. The operating system will take as much unused memory as needed for page caching. As we will see, this can lead to unexpected results when running benchmarks intended to stress storage disks.

Storage Explained

Storage type is another important factor affecting overall performance of an instance when running DSE. Different types of storage offer varying levels of IOPS (input/output operations per second). Because DSE constantly reads and writes data to nodes across the cluster, an instance with higher IOPS will yield better throughput. EC2 instances can use three different types of storage: EBS, Ephemeral, and NVMe Ephemeral. Let's look at them in detail.

Elastic Block Store (EBS)

Elastic Block Store offers the flexibility of choosing exactly how much data you want to attach to an instance. What's more, EBS offers resiliency because data persists even when the instance is stopped, and you can attach EBS volumes to other instances. For additional resiliency, you can back up EBS volumes while live using snapshots. EBS volumes also simplify upgrading and rolling back DSE versions through their portability and snapshot backups.

EBS can be backed by either HDDs (hard disk drives) or SSDs (solid state drives). Because of the high IOPS needs of DSE, you should always choose SSD. AWS offers two types of SSD-backed EBS volumes, described in the table below.

Table 2: Elastic Block Store types (SSD)

EBS Type	Description	Notes	Bandwidth
gp2	General Purpose SSD	IOPS is based on volume size	160 MB/s
io1	Provisioned IOPS	IOPS is configurable	320 MB/s

Ephemeral Storage

Ephemeral Storage offers significantly faster storage compared to EBS. However, there are some caveats. First, the instance is assigned a fixed amount of storage that can't be changed. Also, because the storage is ephemeral, data does not persist if you stop the instance. (Happily, data does persist through an instance restart.)

These factors aren't necessarily a problem for DSE, as data is distributed across the cluster. Data can be copied from one node to another in the cluster, so even when using Ephemeral Storage, you won't ever lose data as long as you keep a quorum of instances. (More on quorums later.)

NVMe Ephemeral

NVMe Ephemeral is Ephemeral Storage running on NVMe (Non-Volatile Memory Express) SSDs. NVMe SSDs are Solid State Drives that are optimized for low latency and high I/O, utilizing the PCIe bus with higher data transfer rates than traditional SATA. Other than increased performance, NVMe Ephemeral behaves similarly to Ephemeral Storage—storage size cannot be changed, and data does not persist if an instance is stopped.

Before the introduction of I3 instances, EBS provided the best cost efficiency while offering acceptable performance. While EBS is not as fast as Ephemeral Storage, EBS-backed instances are typically less expensive than ephemeral-backed instances. However, the new I3 instances with NVMe Ephemeral Storage change the equation. As we'll see, combining NVMe-backed storage with Intel Xeon v4 CPUs makes I3 instances highly performant when running DSE, and they are cost-effective as well.

AWS Best Practices

Experience building data-driven solutions for our customers has taught us that performance alone isn't the only thing to consider when architecting a DSE cluster on AWS. Other factors can have a big impact on operations and cost over time. Before we press on to benchmarking, let's review a few best practices to keep in mind when designing a DSE solution.

Cost

Cost is always an important factor when designing a DSE solution. However, you can't just consider initial expense—your have to plan for the future as well. What's more, you have to keep performance and data integrity in mind.

Calculating for Growth

One of the great things about DSE is how well it scales linearly. Typically, adding more instances gives you more performance. That means you don't have to start with a big, expensive solution just because you anticipate future growth. Instead, you can simply add additional instances to expand the capacity of your DSE cluster as needed. That said, it's a good idea to make sure your cluster is designed for growth from the get-go.

To scale a DSE cluster in a linear fashion, all instances in a DC (datacenter) ring should be the same type. Having differently sized and scaled DC rings in a cluster is possible without tradeoffs, but a single ring should always have instances of equal specifications. This means that you won't want to start out with a large and expensive instance. Doing so might keep you from having to scale your cluster as often. However, each time you do need to scale, you'll see a huge jump in costs as you add another big, costly instance. What's more, it might be a while before you really need all of that capacity.

On the flip side, you don't want to go too small and cheap at the outset, either. While it's not a problem to add additional instances whenever you need to increase compute power, there are other features that don't scale as well, such as network throughput. That means you'll want to choose an initial instance type that offers adequate networking and other resources.

Replication in sets of three with a network topology strategy and three unique racks (availability zones in AWS) is a healthy production configuration for DSE. Capacity increases with this configuration require adding three nodes at a time to maintain a balanced cluster. As you can see from the data below, adding three i3.16xlarge instances has a much larger cost jump than adding three i3.2xlarge nodes.

Table 3: Comparing costs of scaling instances

Capacity	i3.2xlarge @ $455.52/month	i3.16xlarge @ $3,644.16/month
1x capacity	3 nodes @ $1,366.56/month	3 nodes @ $1,0932.48/month
2x capacity	6 nodes @ $2,733.12/month	6 nodes @ $21,864.96/month
3x capacity	9 nodes @ $4,099.68/month	9 nodes @ $32,797.44/month

At this point you might be wondering about the possibility of upgrading your cluster to new instances if the ones you're currently using aren't up to snuff, or if Amazon releases some exciting new instance types. While this might be tempting, the reality is that moving your cluster to new instances can be tricky and tedious. You'll have to think hard about how much you're really saving and if the move is even worth it. A better approach is to plan for growth from the

beginning—that way you can simply scale your cluster as needed, and you won't have to worry about the hassle of upgrading in the future.

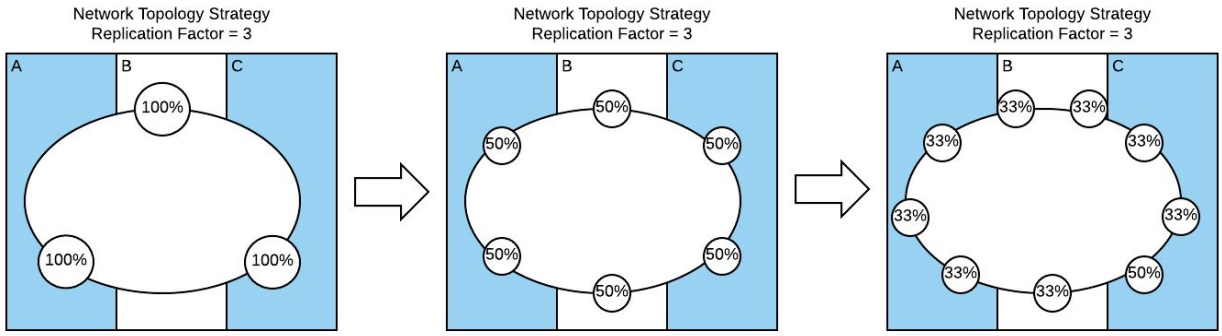
Keeping a Quorum

As a general rule, you should scale your cluster in sets of three instances. For example, start with three instances, then scale to six, and then to nine. You always want three copies of your data so you don't lose a quorum of instances.

A quorum represents the replication factor times the number of nodes. Every time you query DSE, it will calculate to see how many nodes can acknowledge the request. If it can only get one acknowledgement, the write will fail, leading to internal server (500) errors. You won't lose data, but users of your application might have to repeat their requests, and writes might get blocked.

As shown in the figure below, scaling a cluster in sets of three instances means that, even if you lose an instance, you still have two instances that can acknowledge a write. In this way you'll always maintain a quorum, and writes won't fail or get blocked. In addition, if a zone goes down, you will always retain two full copies of your data.

Figure 1: Scaling a DSE cluster



Balancing Resilience and Performance

Resilience and performance are both important factors. So how do you balance between them? In large part, this balance comes down to storage. As we've learned, EC2 instances offer two main types of storage: EBS and Ephemeral.

Despite not being as performant, EBS remains a good choice in many cases because data persists when an instance is shut down. What's more, you can attach an EBS volume to another instance if needed. These features are beneficial for mission-critical data.

In contrast, if an instance with Ephemeral Storage goes down, you'll have to rely on data being replicated back from other nodes in the cluster. Luckily, DSE does this automatically if you've

scaled your cluster correctly across regions. That means you don't have to sacrifice resilience for performance—but it does mean you have to take care in setting up your cluster.

DSE on AWS

DSE adds a number of valuable data management features that aren't part of the Apache Cassandra open source project. For example, with DSE, tiered storage can be configured per node. With tiering, different parts of the file system can be used for different tables or data. You might put the most critical tables on Ephemeral Storage to optimize performance, while keeping less frequently used tables on EBS to reduce costs.

DSE OpsCenter, for administration and monitoring, offers a host of useful metrics and reports that go beyond what's available with AWS CloudWatch. In our experience, the monitoring tools in OpsCenter are often vital for testing and troubleshooting cluster performance.

DSE Benchmarks

Now that we've identified the key features of EC2 instances and laid out some best practices for running DSE on AWS, it's time to get to the main event: benchmarking DSE performance.

Testing Methodology

To perform the benchmarking, we assembled a test bench using the tools listed in the table below.

Table 4: DSE test bench

Test Component	Software
Stress Tool	YCSB (Yahoo! Cloud Serving Benchmark)
Operating System	Ubuntu 16.04
DSE Version	5.0.5
Monitoring Suite	DSE OpsCenter

YCSB is a standard framework for comparing the performance of cloud data stores. It offers customizable workloads, allowing you to specify workload parameters such as read/write ratio, number or rows, number of fields per row, and the size of each field. YCSB also makes it easy to calculate the test payload size—which turns out to be important, as we'll see. In YCSB, every one million records equals one gigabyte of data on disk, and this can be multiplied by the field size for testing with larger partitions.

We ran a number of benchmarks with DSE to simulate various workloads under different AWS hardware profiles. All of the YCSB tests we used for this analysis are available on GitHub: https://github.com/kenzanlabs/cassandra-ycsb-tests

The operating system used for all instances was Ubuntu 16.04, and all data drives were formatted using XFS. The tests were run against version 5.0.5 of Datastax Enterprise edition, rather than open source Cassandra. This allowed us to use OpsCenter to provision DSE with the recommended settings. In addition, during the tests, we could use OpsCenter to continuously monitor the vitals of the DSE cluster.

We performed tests on the following AWS instance types.

Table 5: Test instance types

Instance	Processor	Generation	CPUs	ECU*	RAM	Storage
i2.2xlarge	Intel Xeon CPU E5-2670 v2 @ 2.50 GHz	Ivy Bridge	8	27	61 GB	Ephemeral
i3.4xlarge	Intel Xeon CPU E5-2686 v4 @ 2.30 GHz	Broadwell	16	53	122 GB	NVMe Ephemeral
c4.2xlarge	Intel Xeon CPU E5-2666 v3 @ 2.90 GHz	Haswell	8	31	15 GB	Ephemeral
c4.2xlarge	Intel Xeon CPU E5-2666 v3 @ 2.90 GHz	Haswell	8	31	15 GB	EBS
r4.2xlarge	Intel Xeon CPU E5-2686 v4 @ 2.30 GHz	Broadwell	8	27	61 GB**	EBS
m4.2xlarge	Intel Xeon CPU E5-2686 v4 @ 2.30 GHz	Broadwell***	8	26	32 GB	EBS

*ECU (EC2 Compute Unit) is a standard unit of integer processing power in AWS

**r4 instances use high-frequency DDR4 RAM (assumed frequency of 2400Mhz)

***m4 instances may use Haswell or Broadwell generation CPUs (our test instance used Broadwell)

Guidelines

For our tests, we used the guidelines below to make sure we were fully putting DSE through its paces. We recommend that others follow similar guidelines when benchmarking DSE.

Test Multiple Workload Ratios

To simulate a range of real-world conditions, we conducted tests using several different read-write ratios:

- 100% Writes
- 100% Reads
- 80% Read/20% Write
- 20% Read/80% Write
- 50% Read/50% Write

Use Large Datasets

From the outset, we wanted to use large data sets to simulate real-world usage. And as we'll see, the size of the test payload had a significant impact on test results. For our tests, we used the following data sizes: 10 GB, 25 GB, and 250 GB.

Send Workloads Across Zones

To make sure network throughput is stressed, it's important to send workloads across AWS zones, not just within zones.

Take Down Nodes

To simulate the performance effects of hardware issues or losing instances, take down nodes in the cluster and then run tests again.

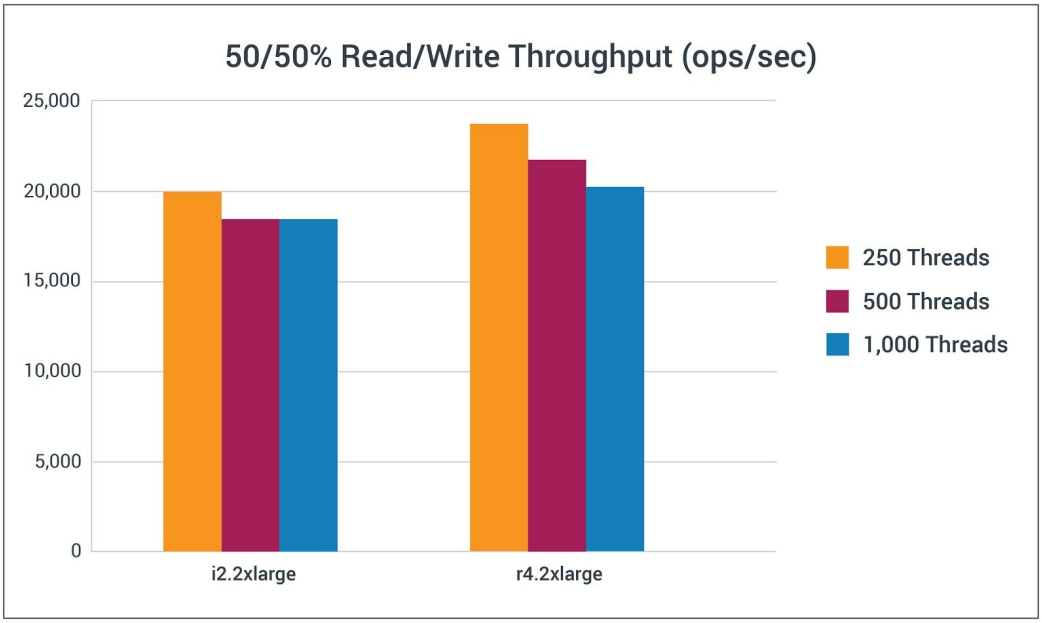
Interpreting Test Results

Testing different DSE configurations in AWS sometimes felt like a science experiment in that the findings we observed weren't always what we expected. However, as we dug deeper, we began to recognize some interesting patterns and effects. To see the raw test data these results are based on, refer to the DSE Benchmark Data spreadsheet.

CPU Generation and Performance

When comparing i2.2xlarge and r4.2xlarge instance types, we expected the I2 instance to have similar or better performance than the R4 instance. This is because both instance types have similar memory capacity (61 GB) and compute capacity (8 vCPUs with 27 compute units), while I2 instances offer faster SSD-backed storage. However, the results were just the opposite of expected—read/write throughput was consistently lower for I2 instances compared to R4 instances, as shown in the figure below.

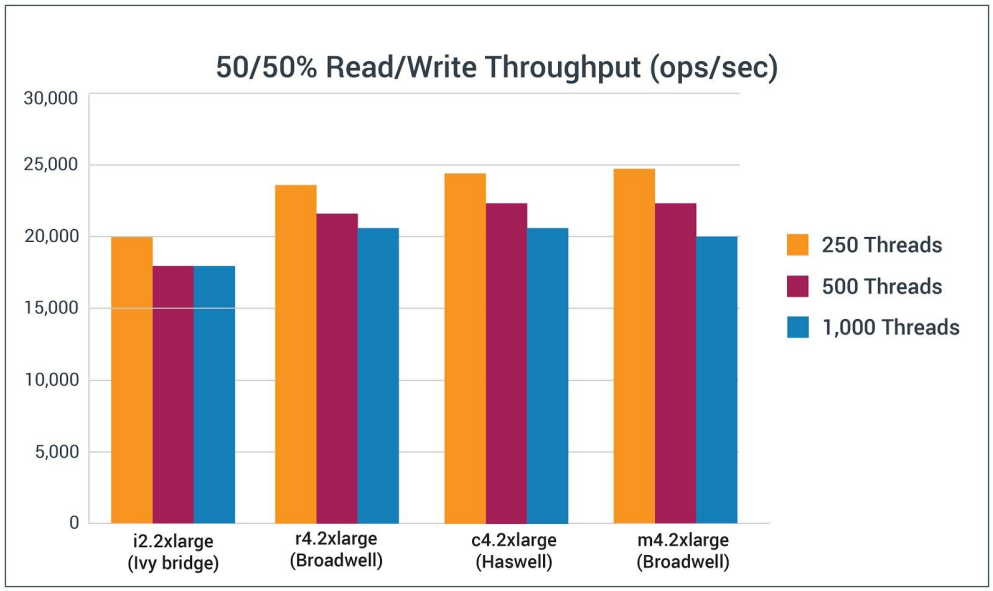
Figure 2: Throughput for i2.2xlarge and r4.2xlarge Instances



Digging a little deeper into the specifications for each instance type reveals the reason for this unexpected result. While the two instances offer similar compute capacity, the I2 instance uses an older Intel Xeon E5-2670 v2 (Ivy Bridge) processor, while the R4 instance uses a new Intel Xeon E5-2686 v4 (Broadwell) processor. In this case, the architectural improvements in the newer generation processor lets the R4 instance handle more threads at once, improving throughput and more than making up for any disadvantage caused by its slower EBS storage.

Testing additional instance types confirmed the importance of CPU generation with regard to throughput. We measured throughput on four more instances types. As the graph below shows, newer generation CPUs yield consistently higher throughput—even when running at lower clock speeds or when paired with less RAM.

Figure 3: Throughput for different CPU generations



Data Size and the Page Cache

Having identified the importance of CPU generation, we next conducted tests focused on comparing the performance of EBS and Ephemeral Storage. This seemed like a straightforward comparison at first—Ephemeral would be faster, right? However, once again there were some unanticipated factors that had an impact on the results.

As we tested, we sometimes observed no disk read activity, indicating that the test was not hitting the disk enough to register metrics. We concluded that small test payloads weren't touching the disk, and instead were fitting entirely within the instance's available RAM. The graphs below (from OpsCenter) show the tipping point during a load test where the memory becomes saturated. Disk read rate and throughput do not register any results until free memory becomes limited. But why is data being served from memory when we didn't enable any sort of caching in DSE? The answer turned out to be the Page Cache feature in Linux.

Figure 4: Free memory and disk activity in OpsCenter

Linux Page Cache

As we discussed earlier, when Linux reads data from disks, it also caches that data in free areas of RAM. Subsequent reads hit the Page Cache rather than disk, until the data is evicted from the cache. Linux caches DSE SSTable files just like any data. That means, when using smaller test payloads, our test data is being entirely served from memory.

Test: Effects of Caching With a 1 GB Payload

We can easily see the effects of the page cache by logging onto an instance with 4GB of RAM and using the **dd** (data duplicator) command.

1. Read a 1 GB file, and then read it again. As you can see, the second read operation is significantly faster—in this case 5.9 GB/s compared to 68.1 MB/s.

```
$ dd if=file.bin of=/dev/null bs=8k
1073741824 bytes (1.1 GB) copied, 15.778 s, 68.1 MB/s

$ dd if=file.bin of=/dev/null bs=8k
1073741824 bytes (1.1 GB) copied, 0.181274 s, 5.9 GB/s
```

2. Next, read five 1 GB files. (Remember, the instance has just 4 GB of RAM.)

```
$ dd if=file1.bin of=/dev/null bs=8k
$ dd if=file2.bin of=/dev/null bs=8k
$ dd if=file3.bin of=/dev/null bs=8k
$ dd if=file4.bin of=/dev/null bs=8k
$ dd if=file5.bin of=/dev/null bs=8k
```

3. Now read the first file again. Notice the slower read speed of 68.1 MB/s—this tells us that the first file has been evicted from RAM, and so is being read from disk.

```
$ dd if=file1.bin of=/dev/null bs=8k
1073741824 bytes (1.1 GB) copied, 15.7787 s, 68.1 MB/s
```

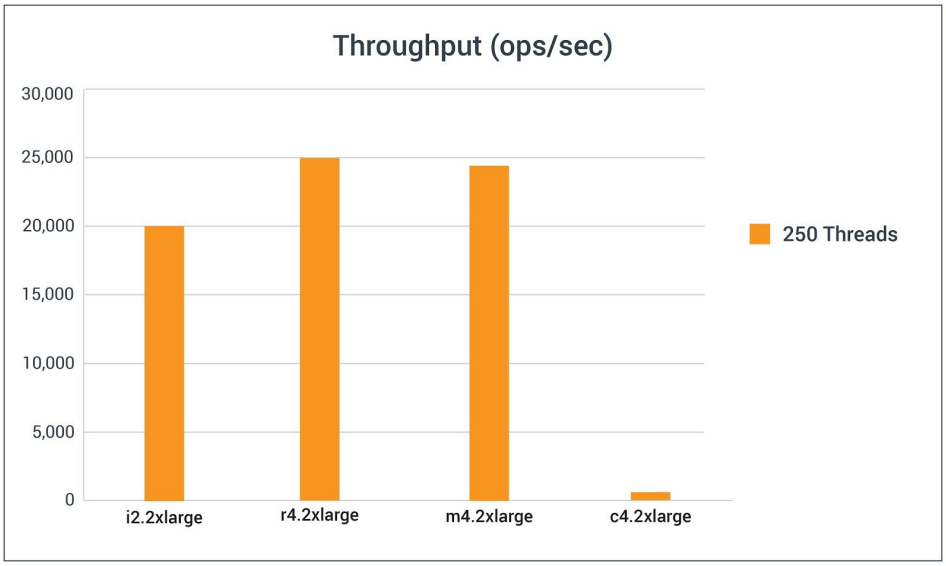
Test: Effects of Caching With a 25 GB Payload

To further explore the effects of caching, we loaded a larger data set (about 25 GB) and compared read performance in warm (cached) instances to cold (non-cached) instances. To produce a cold instance, we shut down DSE and flushed the OS page cache:

```
$ systemctl stop dse
$ sync; echo 3 > /proc/sys/vm/drop_caches
$ systemctl start dse
```

The i2.2xlarge, r4.2xlarge, and m4.2xlarge tests were not impacted by the larger payload size, taking about 6–10 minutes to complete for a cold instance. However, the c4.2xlarge test was much slower, taking over two hours to complete, as shown in the graph below.

Figure 5: Throughput for cold (non-cached) Instance



The difference in performance can be explained by the difference in RAM assigned to each instance. The I2 and R4 instances have 61 GB of RAM, and the M4 instance has 32GB of RAM. That's enough to serve the test payload entirely from memory. However, the C4 instance only has 15 GB of RAM. That means the 25 GB payload could not fit in memory, and instead data had to be read from disk (an EBS volume in this case).

Storage Type and Throughput

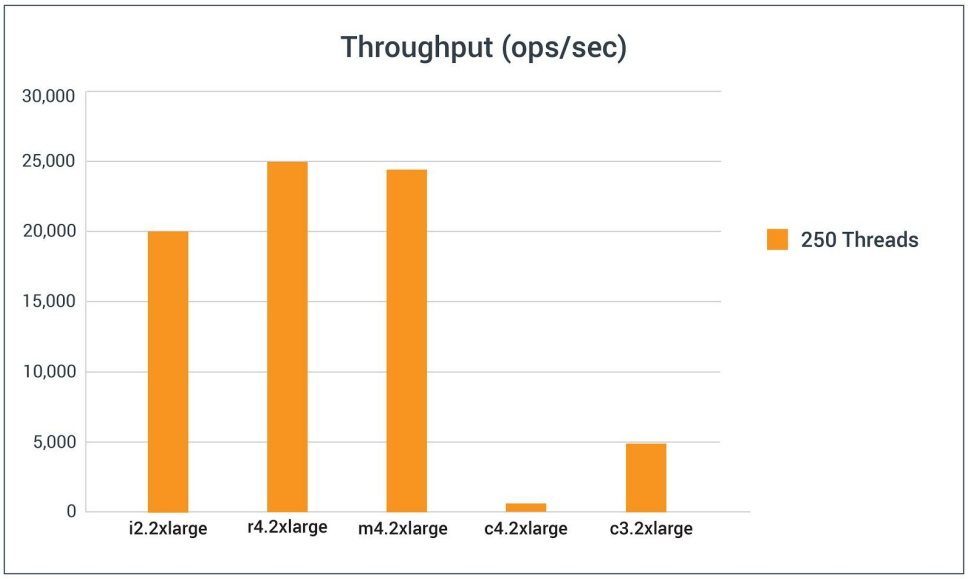
So far we've made some interesting discoveries showing how CPU and memory play a role in performance, but we still haven't settled the debate between Ephemeral and EBS storage types. Specifically, we were curious if using faster storage could help overcome the effects of a payload not fitting into memory.

Test: Comparing EBS and Ephemeral Storage

To test the performance of different storage types, we compared a c4.2xlarge instance to a c3.2xlarge instance. Both have similar CPU and RAM (15 GB), but the C4 instance is backed by EBS, while the C3 instance is backed by faster Ephemeral Storage.

As the graph below shows, Ephemeral storage does improve throughput. The C3 instance with Ephemeral storage achieves 4850 ops/s, while the C4 instance with EBS only achieves 1032 ops/s. However, performance still lags far behind instances that can accommodate the test data in memory.

Figure 6: Throughput for EBS vs. Ephemeral Storage



Test: General Purpose SSD vs Provisioned IOPS

As we discussed earlier, AWS offers two types of SSD-backed EBS—general purpose (gp2) offering 160 MB/s bandwidth, or provisioned IOPS (io1) offering 320 MB/s bandwidth. All of our prior tests were run using gp2 EBS, so we ran them again using io1 EBS to see how using the faster EBS volumes would compare with Ephemeral Storage.

Our expectation was that that io1 EBS would show better performance than gp2 EBS. Yet once again, our initial results didn't meet our preconceived ideas. Testing a c4.2xlarge instance with io1 EBS showed no performance gains at all. How can that be?

On further reflection, this result actually made sense. A c4.2xlarge instance only offers 1,000 Mbps of EBS bandwidth, which equates to 125 MB/s. That means it can't possibly reach the 320 MB/s bandwidth offered by io1 EBS. Our next step was clear: we had to go bigger.

To see a benefit from io1 performance, we needed an instance with at least 3,000 Mbps of EBS bandwidth. The smallest instance type with that level of bandwidth has 122 GB of RAM. That's a lot of memory! And it meant we needed an even bigger payload to make sure that our test data wouldn't fit into RAM, and would instead stress the disk. With that in mind, we ran our tests again, this time using a 250 GB payload.

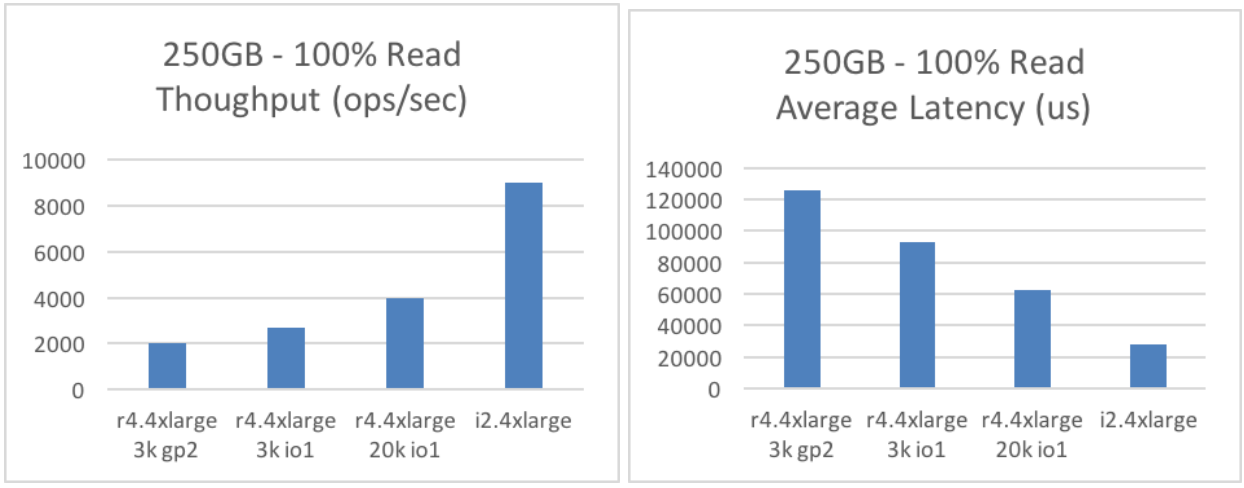
For this test, we used the following instance types (all with 16 cpus and 122 GB memory):

- I2.4xlarge with Ephemeral
- r4.4xlarge with EBS (3k IOPS gp2)
- r4.4xlarge with EBS (3k IOPS io1)
- r4.4xlarge with EBS (20k IOPS io1)

We bootstrapped DSE with 250 GB of data and then read the data out after compaction. The tests took over 12 hours to complete. Looking at the results in the graphs below, we can draw several conclusions:

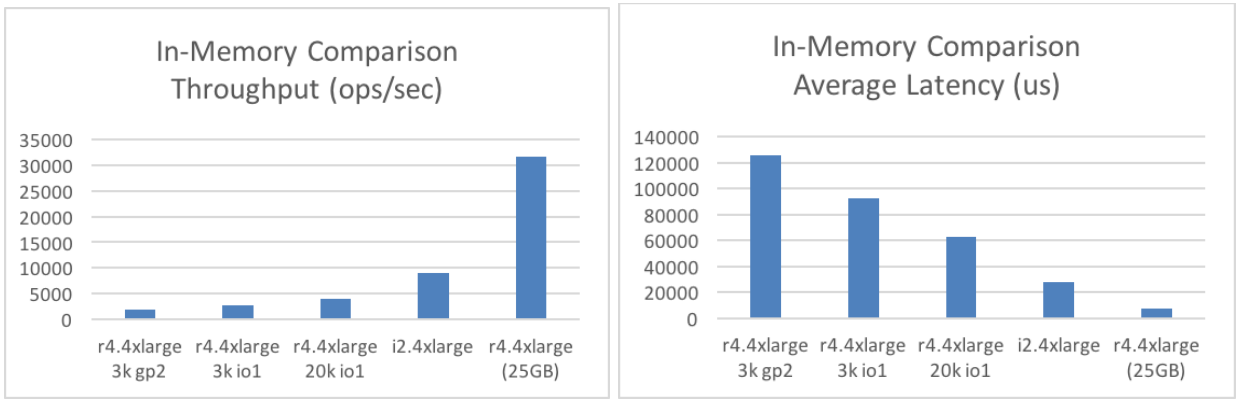
- The I2 instance with Ephemeral Storage has the best disk performance.
- Increasing IOPS on the EBS volumes does increase disk performance.
- An io1-backed instance is more performant than a gp2-backed instance with similar IOPS.

Figure 7: Throughput for gp2 and i01 EBS compared to Ephemeral



Finally, we were curious how all of these storage options compared to a scenario where the payload fits entirely in RAM. Running the test again in an r4.4xlarge instance with a data set that fits in memory, we again see better performance. The R4 instance (with enough RAM to hold the entire payload) has much higher throughput and lower latency compared to even an I2 instance with Ephemeral Storage.

Figure 8: Throughput for EBS and Ephemeral compared to data set in RAM



The Case For EBS

Given the better performance of Ephemeral Storage, it may seem like Ephemeral is the clear choice for instance storage. But as we've discussed, EBS does have advantages in terms of flexibility and resiliency.

You might wonder if using RAID EBS would boost performance to levels closer to Ephemeral. However, there are a couple of important caveats. First, EBS bandwidth is shared across all volumes on an instance, so using RAID won't increase overall available bandwidth. Second, using RAID0 might increase performance—DSE will natively stripe data when multiple disks are configured. However, using RAID0 sacrifices resiliency, which is one of the big advantages of EBS.

So if we can't make the case for EBS based on performance, what about making it based on cost? The table below shows estimated costs for operating a DSE cluster on different instance types. All of these instances have similar levels of compute power (53 ECU) but differ in storage type and, in one case, CPU generation.

Historically, EBS has been a bit less expensive to operate. In fact, you could operate two r4.4xlarge instances with EBS for less than the cost of a single i2.4xlarge instance with Ephemeral Storage, thereby doubling your compute power. Only when moving to an instance with 20,000 IOPS provisioned does EBS approach the same costs as Ephemeral.

Table 6: Monthly cost of instance types

Instance	CPU Generation	Storage Size	Storage Type	Monthly Cost
i2.4xlarge	Ivy Bridge	3.2 TB	Ephemeral	$2745.74
r4.4xlarge	Broadwell	3.2 TB	EBS gp2	$1205.44
r4.4xlarge	Broadwell	3.2 TB	EBS 3k IOPS io1	$1511.24
r4.4xlarge	Broadwell	3.2 TB	EBS 10k IOPS io1	$2011.74
r4.4xlarge	Broadwell	3.2 TB	EBS 20k IOPS io1	$2726.74
r4.4xlarge	Broadwell	3.2 TB	EBS gp2 (x2)	$2410.88

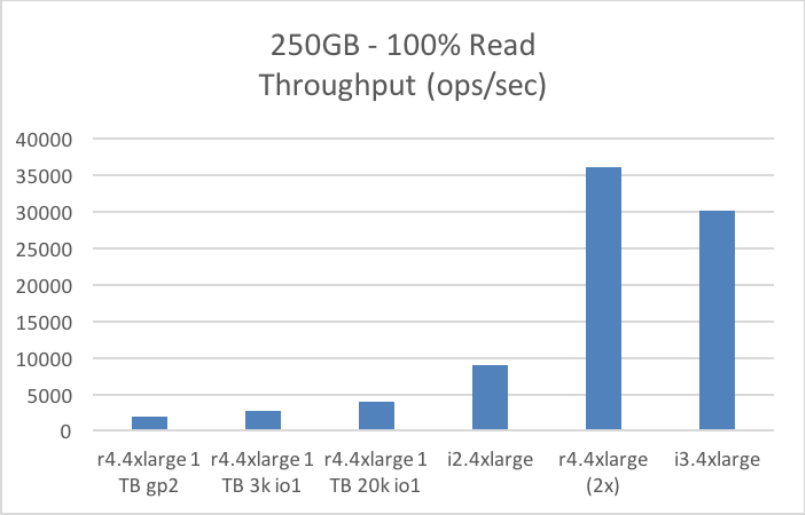
If this was the end of our story, you might conclude that the resiliency and reduced costs of EBS make it a good choice compared to Ephemeral Storage. However, technology keeps moving forward, and this adds a new twist to our tale.

Raising the Bar with I3

As we mentioned earlier, Amazon introduced a replacement for the I2 instance type late in our testing process. Compared to I2, the new I3 instance type uses a later generation (Broadwell) Xeon CPU. It also uses a newer type of Ephemeral Storage based on faster NVMe SSDs. Finally, the I3 instance is offered at a lower hourly usage cost.

Just how much faster is an I3 instance? We loaded a 250 GB test payload into our cluster and ran our tests against an i3.4xlarge instance. As the graph below shows, the I3 instance has far higher read throughput than the previous generation I2 instance. In fact, a single I3 instance nearly matches the performance of two R4 instances!

Figure 9: Performance of i3.4xlarge instance



What about the operating costs of an I3 instance? Let's take our previous table comparing monthly costs and add in the I3 instance. As can be seen below, not only does the I3 instance offer far better performance, it does so at greatly reduced costs thanks to its newer, more efficient hardware. So unless you need the flexible storage size or resiliency of EBS, choosing an I3 instance with Ephemeral Storage will give you far better performance with a much lower monthly cost to operate.

Table 7: Monthly cost of instance types (with I3)

Instance	CPU Generation	Storage Size	Storage Type	Monthly Cost
i2.4xlarge	Ivy Bridge	3.2 TB	Ephemeral	$2745.74
r4.4xlarge	Broadwell	3.2 TB	EBS gp2	$1205.44
r4.4xlarge	Broadwell	3.2 TB	EBS 3k IOPS io1	$1511.24
r4.4xlarge	Broadwell	3.2 TB	EBS 10k IOPS io1	$2011.74
r4.4xlarge	Broadwell	3.2 TB	EBS 20k IOPS io1	$2726.74
r4.4xlarge	Broadwell	3.2 TB	EBS gp2 (x2)	$2410.88
i3.4xlarge	**Broadwell**	**3.8 TB**	**Ephemeral**	**$911.04**

Conclusion

With the aid of the YCSB stress tool, and the management and monitoring tools provided by DataStax Enterprise (DSE), we were able to test DSE performance in a variety of EC2 instance types and storage configurations. The results show that there are a number of factors to keep in mind when setting up and scaling a DSE cluster on AWS.

When possible, choose instances based on the latest generations of Intel Xeon CPUs. Newer Xeon v3 (Haswell) and Xeon v4 (Broadwell) processors improve DSE throughput by offering a higher number of operations per second and by increasing concurrent compactors.

Instance memory is a key factor to consider when benchmarking DSE. If the test payload is small enough to fit within the instance's available RAM, then subsequent reads after the first will be accessed from the Page Cache rather than from disk, resulting in greatly increased performance.

Real-world data may exceed the size of the Page Cache, but that doesn't mean you can't benefit from Linux's native caching capabilities. Increasing memory will reduce disk I/O in general, and more memory increases the data ownership per node. If data doesn't fit in memory, you can simply add more instances.

The important takeaway here is that the data that is most frequently accessed should fit in memory. To achieve this, infrequently-read records can be organized in a way that places them in a separate data file that will fall out of the cache, making room for more frequently-accessed data files. DSE has configurable options per table, such as compaction strategy, allowing

customization of how the data is organized on disk. This demonstrates the importance of choosing the right compaction strategy for the access patterns of each table.

Disk performance becomes a factor when data doesn't fit in memory. Elastic Block Store (EBS) offers both utility and resiliency. Read performance can be increased by choosing EBS with the provisioned IOPS option. Ephemeral Storage offers the highest disk performance. While data on Ephemeral volumes does not persist if an instance is shut down, this is generally not an issue if the DSE cluster is properly configured in sets of three instances, as data is replicated across the cluster.

While EBS has historically been the lower-cost storage option, the landscape has changed with the recent introduction of the I3 instance type. I3 instances offer high-speed NVMe Ephemeral Storage at a lower monthly operating cost compared to EBS-backed instances or older I2 Ephemeral-backed instances.

While AWS makes available a wide variety of instance types, hardware configurations, and storage options, you can narrow down the choices based on the factors that are most critical for your data. By carefully designing benchmark tests, and using DSE to manage and monitor your cluster, you can find out what works best for your use cases, as well as for your AWS budget.

About the Author: Darren Bathgate

Darren Bathgate is a technical architect based out of Kenzan's Providence, Rhode Island location. During his six years at Kenzan, Darren has designed data models for relational SQL databases, including MySQL and Oracle, and has optimized query performance on legacy databases. He has also built reactive pipelines using Hadoop, Spark, and Cassandra. Darren is a graduate of the New England Institute of Technology, where he studied software engineering and received his Master's degree in information technology. In his free time he enjoys rock climbing, gaming, and keeping up with the latest computer hardware.