Scaling Up versus Scaling Out: Mythbusting Database Deployment Options for Big Data

CONTENTS

OVERVIEW

In the world of Big Data, scaling out the database is the norm. As a result, Big Data deployments are often awash in clusters of small nodes, which bring with them many hidden costs. For example, scaling out usually comes at the cost of resource efficiency, since it can lead to low resource utilization. So why is scaling out so common? Scaled out deployment architectures are based on several assumptions that we'll examine in this white paper. We'll demonstrate that these assumptions prove unfounded for database infrastructure that is able to take advantage of the rich computing resources available on large nodes. In the process, we'll show that Scylla is uniquely positioned to leverage the multi-core architecture that modern cloud platforms offer, making it not only possible, but also preferable, to use smaller clusters of large nodes for Big Data deployments.

THE NEW AGE OF NOSQL AND COMMODITY HARDWARE

The NoSQL revolution in database management systems kicked off a full decade ago. Since then, organizations of all sizes have benefitted from a key feature that the NoSQL architecture introduced: massive scale using relatively inexpensive commodity hardware. Thanks to this innovation, organizations have been able to deploy architectures that would have been prohibitively expensive and impossible to scale using traditional relational database systems.

As businesses across industries embrace digital transformation, their ability to align scale with consumer demand has proven to be a key competitive advantage. As a result, IT groups constantly strive to balance competing demands for higher performance and lower costs. Since NoSQL makes it easy to scale out and back in rapidly on cheaper hardware, it offers the best of both worlds: performance along with cost savings. This need for flexible and cost-effective scale will only intensify as companies accelerate digital transformation.

Over the same decade, 'commodity hardware' itself has undergone a transformation. However, most modern software doesn't take advantage of modern computing resources. Most Big Data frameworks that scale out, don't scale up. They aren't able to take advantage of the resources offered by large nodes, such as the added CPU, memory, and solid-state drives (SSDs), nor can they store large amounts of data on disk efficiently. Managed runtimes, like Java, are further constrained by heap size. Multi-threaded code, with its locking overhead and lack of attention for Non-Uniform Memory Architecture (NUMA), imposes a significant performance penalty against modern hardware architectures.

Software's inability to keep up with hardware advancements has led to the widespread belief that running database infrastructure on many small nodes is the optimal architecture for scaling massive workloads. The alternative, using small clusters of large nodes, is often treated with skepticism.

These assumptions are largely based on experience with NoSQL databases like Apache Cassandra. Cassandra's architecture prevents it from efficiently exploiting modern computing resources—in particular, multi-core CPUs. Even as cloud platforms offer bigger and bigger machine instances, with massive amounts of memory, Cassandra is constrained by many factors, primarily its use of Java but also its caching model and its lack of an asynchronous architecture. As a result, Cassandra is often deployed on clusters of small instances that run at low-levels of utilization.

ScyllaDB set out to fix this with its close-to-the-hardware Scylla database. Written in C++ instead of Java, Scylla implements a number of low-level architectural techniques, such as thread-per-core and sharding. Those techniques combined with a shared-nothing architecture enable Scylla to scale linearly with the available resources. This means that Scylla can run massive workloads on smaller clusters of larger nodes, an approach that vastly reduces all of the inputs to TCO.

A few common concerns are that large nodes won't be fully utilized, that they have a hard time streaming data when scaling out and, finally, they might have a catastrophic effect on recovery times. Scylla breaks these assumptions, resulting in improved TCO and reduced maintenance.

In light of these concerns, we ran real-world scenarios against Scylla to demonstrate that the skepticism towards big nodes is misplaced. In fact, with Scylla, big nodes are often best for Big Data.

WHAT IS A LARGE NODE?

Before diving in, it's important to understand what we mean by 'node size' in the context of today's cloud hardware. Since Amazon EC2 instances are regularly used to run database infrastructure, we will use their offerings for reference. The image below shows the Amazon EC2 i3 family, with increasingly large nodes, or 'instances.' As you can see, the number of virtual CPUs spans from 2 to 64. The amount of memory of those machines grows proportionally, reaching almost half a terabyte with the i3.16xlarge instance. Similarly, the amount of storage grows from half a terabyte to more than fifteen terabytes.

Instance Name	vCPU Count	Memory	Instance Storage (NVMe SSD)	Price/ Hour
i3.large	2	15.25 GiB	0.475 TB	$0.15
i3.xlarge	4	30.5 GiB	0.950 TB	$0.31
i3.2xlarge	8	61GiB	1.9 TB	$0.62
i3.4xlarge	16	122 GiB	3.8 TB (2 disks)	$1.25
i3.8xlarge	32	244 GiB	7.6 TB (4 disks)	$2.50
i3.16xlarge	64	488 GiB	15.2 TB (8 disks)	$4.99

The Amazon EC2 i3 family

On the surface, the small nodes look inexpensive, but prices are actually consistent across node sizes. No matter how you organize your resources—the number of cores, the amount of memory, and the number of disks—the price to use those resources in the cloud will be roughly the same. Other cloud providers, such as Microsoft Azure and Google Cloud, have similar pricing structures, and therefore a comparable cost benefit to running on bigger nodes.

ADVANTAGES OF LARGE NODES

Now that we've established resource price parity among node sizes, let's examine some of the advantages that large nodes provide.

- Less Noisy Neighbors: On cloud platforms multi-tenancy is the norm. A cloud platform is, by definition, based on shared network bandwidth, I/O, memory, storage, and so on. As a result, a deployment of many small nodes is susceptible to the 'noisy neighbor' effect. This effect is experienced when one application or virtual machine consumes more than its fair share of available resources. As nodes increase in size, fewer and fewer resources are shared among tenants. In fact, beyond a certain size your applications are likely to be the only tenant on the physical machines on which your system is deployed. This isolates your system from potential degradation and outages. Large nodes shield your systems from noisy neighbors.
- Fewer Failures: Since nodes large and small fail at roughly the same rate, large nodes deliver a higher mean time between failures, or "MTBF" than small nodes. Failures in the data layer require operator intervention, and restoring a large node requires the same amount of human effort as a small one. In a cluster of a thousand nodes, you'll see failures every day. As a result, big clusters of small nodes magnify administrative costs.
- Datacenter Density: Many organizations with on-premises datacenters are seeking to increase density by consolidating servers into fewer, larger boxes with more computing resources per server. Small clusters of large nodes help this process by efficiently consuming denser resources, in turn decreasing energy and operating costs.

- Operational Simplicity: Big clusters of small instances demand more attention, and generate more alerts, than small clusters of large instances. All of those small nodes multiply the effort of real-time monitoring and periodic maintenance, such as rolling upgrades.

In spite of these significant benefits, systems awash in a sea of small instances remain commonplace. In the following sections, we'll run scenarios that investigate the assumptions that lead to the use of small nodes instead of big nodes, including performance, compaction, and streaming. We'll demonstrate that those assumptions don't always hold true when you crunch the numbers.

NODE SIZE AND PERFORMANCE

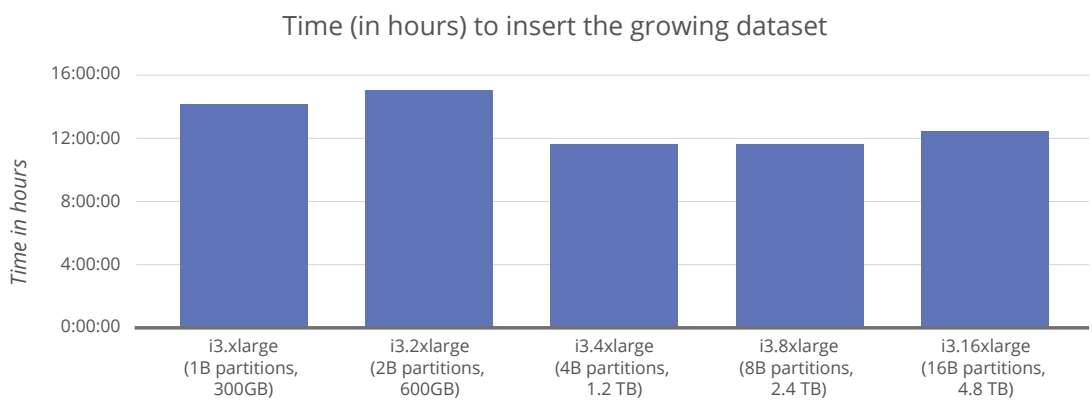
A key consideration in environment sizing is the performance characteristics of the software stack. As we've pointed out, Scylla is written on a framework that scales up as well as out, enabling it to double performance as resources double. To prove this, we ran a few scenarios against Scylla.

This scenario uses a single cluster of 3 machines from the i3 family. (Although we're using only 3 nodes here, out in the real-world Scylla runs in clusters of 100s of nodes, both big and small. In fact, one of our users runs a 1PB cluster with only 30 nodes). 3 nodes are configured as replicas (a replication factor of 3). Using a single loader machine, the reference workload inserts 1,000,000,000 partitions to the cluster as quickly as possible. We double resources available to the database at each iteration. We also increase the number of loaders, doubling the number of partitions and the dataset at each step.

As you can see in the chart below, ingest time of 300GB with 4 virtual CPU clusters is roughly equal to 600GB using 8 virtual CPUs, 1.2TB with 16 virtual CPUs, 24TB with 16 virtual CPUs and even when the total data size reached almost 5TB, ingest time for the workload remained relatively constant. These tests show that Scylla's linear scale-up characteristics make it advantageous to scale up before scaling out, since you can achieve the same results on fewer machines.

NODE SIZE AND STREAMING

Some architects are concerned that putting more data on fewer nodes increases the risks associated with outages and data loss. You can think of this as the 'Big Basket' problem. It may seem intuitive that storing all of your data on a few large nodes makes them more vulnerable to outages, like putting all of your eggs in one basket. Current thinking in distributed systems tends to agree, arguing that amortizing failures over many tiny and ostensibly inexpensive instances is best. But this doesn't necessarily hold true. Scylla uses a number of innovative techniques to ensure availability while also accelerating recovery from failures, making big nodes both safer and more economical.



The time to ingest the workload in Scylla remains constant as the dataset grows from 300GB to 4.8TB.

Restarting or replacing a node requires the entire dataset to be replicated to the new node via streaming. On the surface, replacing a node by transferring 1TB seems preferable to transferring 16TB. If larger nodes with more power and faster disks speed up compaction, can they also speed up replication? The most obvious potential bottleneck to streaming is network bandwidth.

But the network isn't the bottleneck for streaming. AWS's i3.16xlarge instances have 25 Gbps network links—enough bandwidth to transfer 5TB of data in thirty minutes. In practice, that speed proves impossible to achieve because data streaming depends not only on bandwidth, but also on CPU and disk. A node that dedicates significant resources to streaming will impact the latency of customer-facing queries.

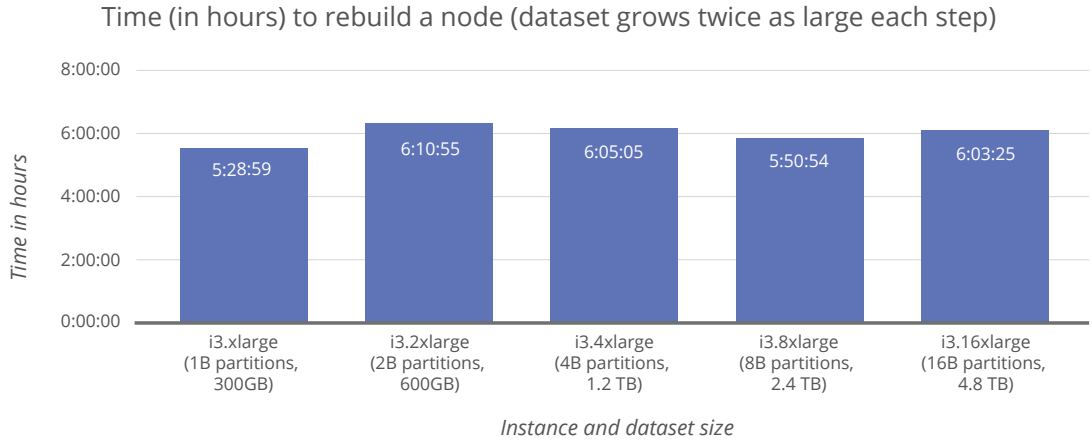
Streaming operations benefit from Scylla's autonomous and auto-tuning capabilities. Scylla queues all requests in the database and then uses an I/O scheduler and a CPU scheduler to execute them in priority order. The schedulers use algorithms derived from control theory to determine the optimal pace at which to run operations, based on available resources.

These autonomous capabilities result in rapid replication with minimal impact on latency. Since this happens dynamically, added resources automatically factor into scheduling decisions. Nodes can be brought up in a cluster very quickly, further lowering the risk of running smaller clusters with large nodes.

To demonstrate this, we ran a streaming scenario against a Scylla cluster. Using the same clusters as in the previous demonstration, we destroyed the node and rebuilt it from the other two nodes in the cluster. Based on these results, we can see that it takes about the same amount of time to rebuild a node from the other two, even for large datasets. It doesn't matter if the dataset is 300GB or almost 5TB. Our experiment shows that the Scylla database busts the myth that big nodes result in big problems when a node goes down.

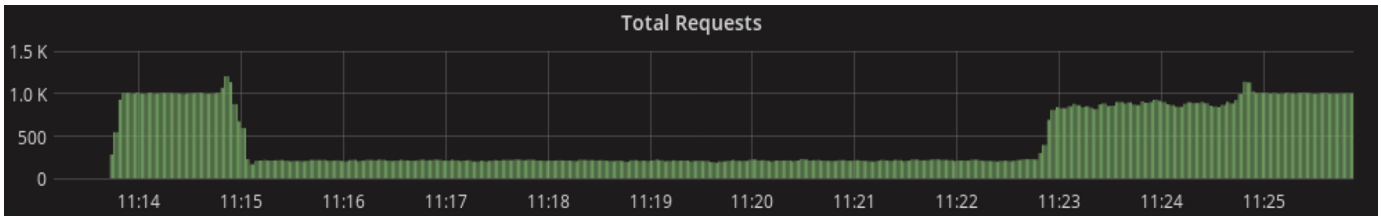
Calculating the real cost of failures involves more factors than than the time to replicate via streaming. If the software stack scales properly, as it does with Scylla, the cost to restore a large node is the same as for a small one. Since the mean time between failures tends to remain constant regardless of node size, fewer nodes also means fewer failures, less complexity, and lower maintenance overhead.

Efficient streaming enables nodes to be rebuilt or scaled out with minimal impact. But a second problem, cold caches, can also impact restarts. What happens when a node is restarted, or a new node is added to the cluster during rolling upgrades or following hardware outages? The typical database simply routes requests randomly across replicas in the cluster. Nodes with cold caches cannot sustain the same throughput with the same latency.

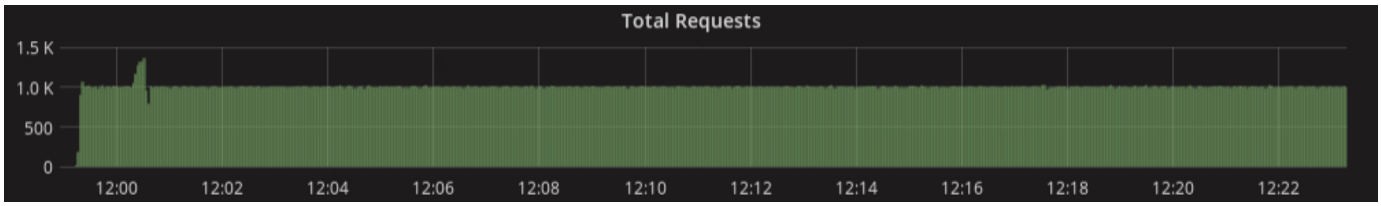


The time to rebuild a Scylla node remains constant with larger datasets.

BEFORE:



AFTER:



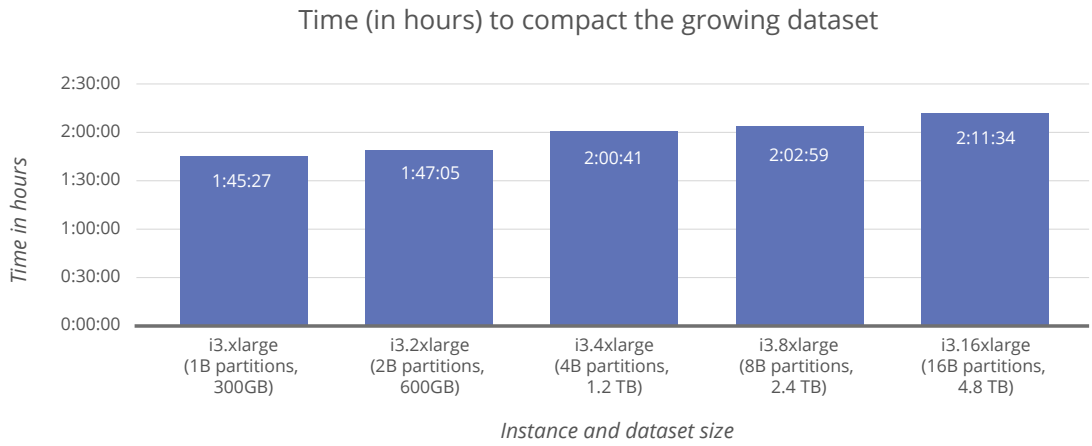
Scylla prevents latency spikes when nodes are restarted.

Scylla takes a different approach. Our patent-pending technology, heat-weighted load balancing, intelligently routes requests across the cluster to ensure that node restarts don't impact throughput or latency. Heat-weighted load balancing implements an algorithm that optimizes the ratio between a node's cache hit rate and the proportion of requests it receives. Over time, the node serving as request coordinator sends a smaller proportion of reads to the nodes that were up, while sending more reads to the restored node. The feature enables operators to restore or add nodes to clusters efficiently, since rebooting individual nodes no longer affects the cluster's read latency.

NODE SIZE AND COMPACTION

Readers with real-world experience with Apache Cassandra might react to large nodes with a healthy dose of skepticism. In Apache Cassandra, a node with just 5TB can take a long time to compact. (Compaction is a background process similar to garbage collection in the Java programming language, except that it works with disk data, rather than objects in memory).

Since compaction time increases along with the size of the dataset, it's often cited as a reason to avoid big nodes for data infrastructure. If compactions happen too quickly, foreground workloads are destroyed because all of the CPU



Compaction times in Scylla remained level as the dataset doubled with each successive iteration.

and disk resources are soaked up. If compaction is too slow, then reads suffer. A database like Cassandra requires operators to find the proper settings through trial and error, and once set, they are static. With those databases, resource consumption remains fixed regardless of shifting workloads.

As noted in the context of streaming, Scylla takes a very different approach, providing autonomous operations and auto-tuning that give priority to foreground operations over background tasks like compaction—even as workloads fluctuate. The end result is rapid compaction with no measurable effect on end user queries.

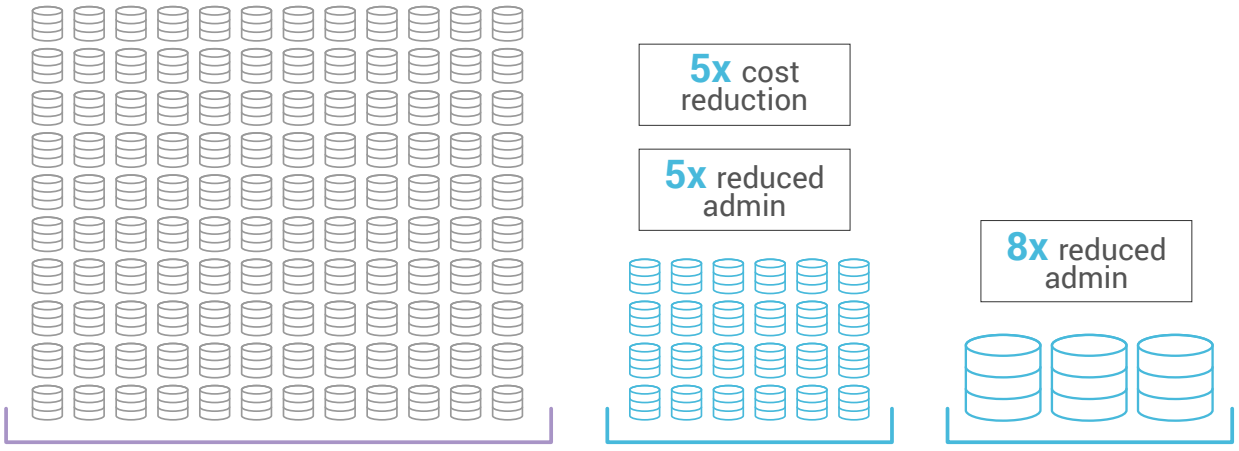
To prove this point, we forced compaction on a node using Scylla's tool, which is compatible with Cassandra. We called `nodetool compact` in one of the nodes in same cluster as in the previous experiment. The tool forces the node to compact all SSTables into a single one.

Remarkably, these results show that compaction time on Scylla remains constant for a dataset of 300GB versus 5TB, as long as all instances in the cluster scale resources proportionately. This demonstrates that one of the primary arguments against using large nodes for Big Data hinges on the false assumption that compaction of huge datasets must necessarily be slow. Scylla's linear scale-up characteristics show that the opposite is true; compaction of large datasets can also benefit from the resources of large nodes.

THE DATABASE MAKES A DIFFERENCE

As our scenarios demonstrate, running on fewer nodes is inherently advantageous. The highly performant Scylla database reduces the number of nodes required for Big Data applications. It's scale-up capabilities enable organizations to further reduce node count by moving to larger machines. By increasing the size of nodes and reducing their number, IT organizations can recognize benefits at all levels, especially in TCO. Moving from a large cluster of small nodes to small clusters of large nodes has a significant impact on TCO, while vastly reducing the mean-time between failures (MTBF) and administrative overhead.

The benefits of scaling up that we have demonstrated in this paper have been proven in the field. The diagram below illustrates the server costs and administration benefits experienced by a ScyllaDB customer. This customer migrated a Cassandra installation, distributed across 120 i3.2xl AWS instances



Scylla reduces server cost by 5X and improves MTBF by 40X

with 8 virtual CPUs each, to Scylla. Using the same node sizes, Scylla achieved the customer's performance targets with a much smaller cluster of only 24 nodes. The initial reduction in node sprawl produced a 5X reduction in server costs, from $651,744 to $130,348 annually. It also achieved a 5X reduction in administrative overhead, encompassing failures, upgrades, monitoring, etc.

Given Scylla' s scale-up capabilities, the customer then moved to the biggest nodes available, the i3.16xl instance with 64 virtual CPUs each. While the cost remained the same, those 3 large nodes were capable of maintaining the customer's SLAs. Scaling up resulted in reducing complexity (and administration, failures, etc.) by a factor of 40 compared with where the customer began.

As we've seen, there are many advantages to scaling up a distributed database environment before scaling out. If you've already scaled out, scaling up offers an escape from the costs and overhead of node sprawl.

Scylla gives users both the scale out benefits of NoSQL along with scale up characteristics that enable database consolidation and improved TCO.

Ask yourself...

- Are you concerned your database isn't efficiently utilizing all of its CPUs and disks?
- Are you investing too much time in managing excessively large clusters?
- Does your team have to replace dead nodes at odd hours on the weekends and holidays?
- Are you struggling to scale with your business as you grow 2X, 4X, 10X?
- Are you unable to meet service level agreements during failures and maintenance operations?

If the answer to even one of these questions is 'yes," we'd suggest evaluating your distributed database options. Open source Scylla is available for download from our website.
Or start by running a cloud-hosted Scylla Test Drive, which lets you spin-up a hosted Scylla cluster in minutes.

ABOUT SCYLLADB

Scylla is the real-time big data database. A drop-in alternative to Apache Cassandra and Amazon DynamoDB, Scylla embraces a shared-nothing approach that increases throughput and storage capacity as much as 10X that of Cassandra. AdGear, AppNexus, Comcast, Fanatics, FireEye, Grab, IBM Compose, MediaMath, Ola Cabs, Samsung, Starbucks and many more leading companies have adopted Scylla to realize order-of-magnitude performance improvements and reduce hardware costs. Scylla is available in Open Source, Enterprise and fully managed Cloud editions. ScyllaDB was founded by the team responsible for the KVM hypervisor and is backed by Bessemer Venture Partners, Eight Roads Ventures, Innovation Endeavors, Magma Venture Partners, Qualcomm Ventures, Samsung Ventures, TLV Partners, Western Digital Capital and Wing Venture Capital.

For more information: ScyllaDB.com

SCYLLADB.COM

SCYLLA

United States Headquarters
2445 Faber Place, Suite 200
Palo Alto, CA 94303 U.S.A.
Email: info@scylladb.com

Israel Headquarters
11 Galgalei Haplada
Herzelia, Israel