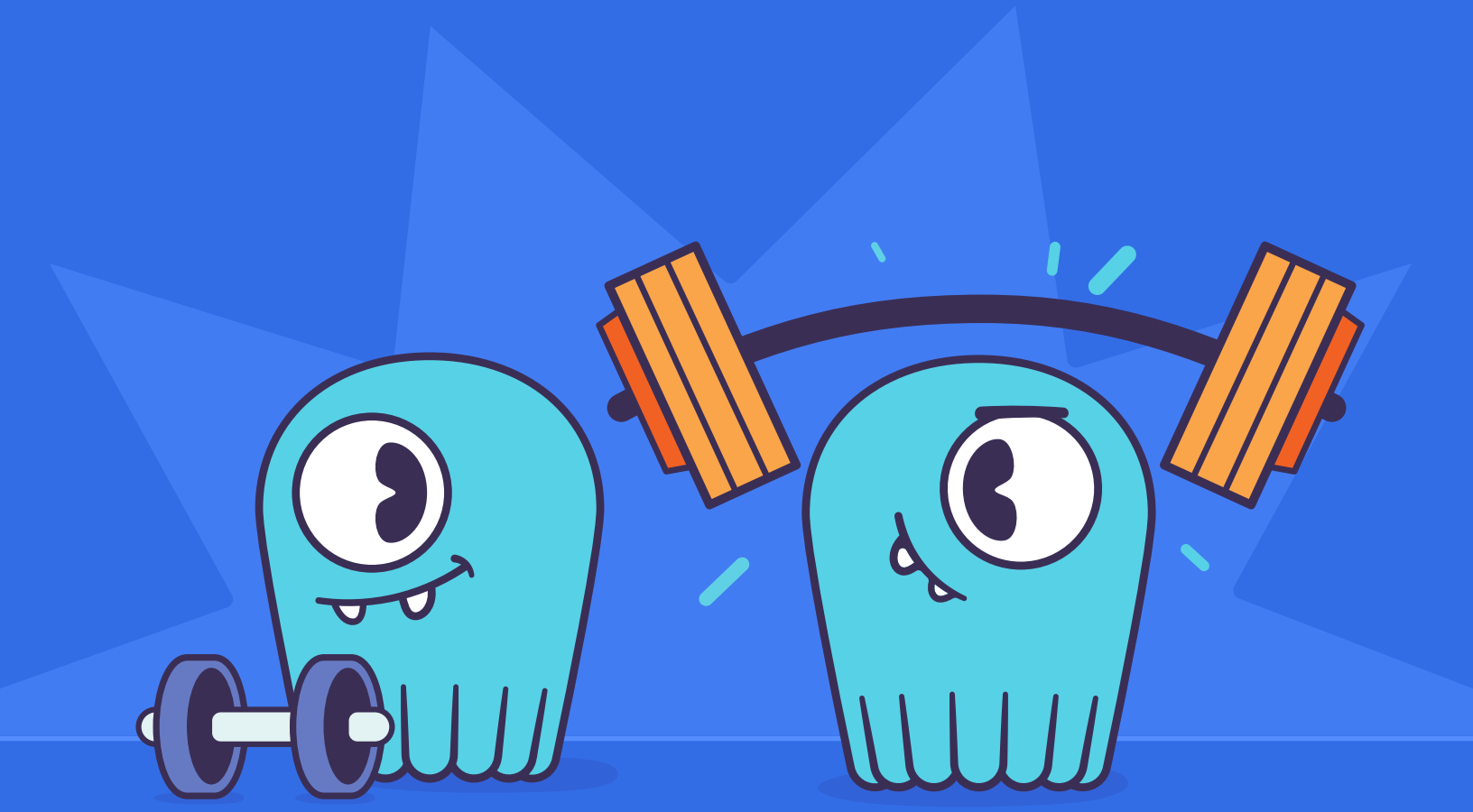
SCYLLA

BEST PRACTICES GUIDE

Maximizing Scylla Performance

A Guide to Getting the Most from Your Scylla Database



The purpose of this guide is to provide an overview of the best practices for maximizing the performance of Scylla, the next-generation NoSQL database. Even though Scylla auto-tunes itself for optimal performance, users still need to apply best practices in order to get the most out of their Scylla deployments.

Get me up and running

In case you are not able to read this document in full, here are the most important things to remember:

- **use the best hardware you can reasonably afford**
- **install Scylla Monitoring Stack**
- **run scylla_setup script**
- **use Cassandra stress test**
- **expect to get at least 12.5K operations per second (OPS) per physical core for simple operations on selected hardware**

Why should I read this? I already know how to execute a benchmark

Scylla is different from any other NoSQL database. It achieves the highest levels of performance and takes full control of the hardware by utilizing all of the server cores in order to provide strict SLAs for low-latency operations. If you run Scylla in an over-committed environment, performance won't just be linearly slower — it will tank completely.

This is because Scylla has a reactor design that runs on all the (configured) cores and a scheduler that assumes a 0.5 ms tick. Scylla does everything it can to control queues in userspace and not in the OS/drives. Thus it assumes the bandwidth that was measured by scylla_setup.

However, it is not difficult to get the best performance out of Scylla. It primarily tunes itself automatically. Just make sure you don't work against the system.

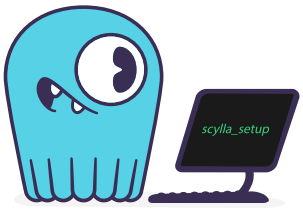
Install Scylla Monitoring Stack

Install and use the Scylla Monitoring Stack, which provides excellent additional value above and beyond performance optimization. If you cannot pinpoint a performance bottleneck, you likely have not configured the system correctly. Scylla Monitoring Stack will help to sort this out.

With the recent addition of the Scylla Advisor to the Scylla Monitoring Stack, it is now even easier to find potential issues.

Install Scylla Manager

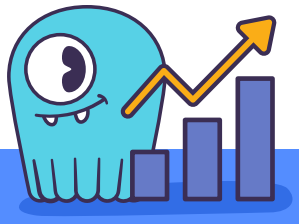
Install and use Scylla Manager together with the Scylla Monitoring Stack. Scylla Manager provides automated backups, and repairs of your database. Scylla Manager can manage multiple Scylla clusters and run cluster-wide tasks in a controlled and predictable way.



Run scylla_setup

Before running Scylla, it is critical that the scylla_setup script has been executed. Scylla doesn't require manual optimization – it is the task of the scylla_setup script to determine the optimal configuration. If scylla_setup has not run, the system won't be configured optimally.

▶ Read more here.



Benchmarking best practices

Use a representative environment

Execute benchmarks on an environment that reflects your production environment. Benchmarking on the wrong environment can easily lead to an order-of-magnitude performance difference. For example, on a laptop you might see 20K OPS while on a dedicated server you could easily achieve 200K OPS. Unless you have your production system running on a laptop, do not benchmark on a laptop.

We recommend automating your benchmarking with tools like Terraform/Ansible so you can more easily repeat the benchmark test.

If you are using shared hardware in a containerized/virtualized environment, be aware that one guest can increase latency in other guests.

Also, make sure you do not underprovision load generators, otherwise the load generators themselves will be the bottleneck.

Use a representative data model

Tools such as cassandra-stress use a default data model that does not completely reflect what actions you will perform in production. For example, the cassandra-stress default data model has a replication factor set to 1 and uses the LOCAL_ONE as a consistency level.

Although cassandra_stress is a convenient way to get some initial performance impressions, it is critical to benchmark the same/similar data model that you will use in production. We therefore recommend that you use a custom data model. For more information refer to the user mode section in our documentation.

Use representative datasets

If you run the benchmark with a dataset that is smaller than your production data, you may have misleading or incorrect results due to the reduced number of I/O operations. Therefore, it is critical to configure the size of the dataset to reflect your production dataset size.

Use a representative load

Run the benchmark using a load that represents, as closely as possible, the load you anticipate having in production. This includes the queries submitted by the load generator. When you use the right type of queries, they are distributed over the partitions and the ratio between read/write remains relatively constant. The read/write ratio is important due to the overhead of compaction and finding the right data on disk.

Proper warmup & duration

When benchmarking, it is important to give the system time to warm up. This allows the database to fill the cache. In addition, it is critical to run the benchmarks long enough so that at least one compaction is triggered.

Latency test vs throughput test

When performing a load test you will need to differentiate between a latency test and a throughput test. With a throughput test, you measure the maximum throughput by sending a new request as soon as the previous request completes. With a latency test, you pin the throughput at a fixed rate. In both cases, latency is measured.

Most engineers will start with a throughput test, but often a latency test is a better choice because they know the desired throughput, e.g. 1M op/s. This is especially the case if your production system must meet a specific SLA. For example, the 99.99 percentile should have a latency less than 10ms.

Coordinated omission

A common problem when measuring latencies is the coordinated omission problem, which causes the worst latencies to be omitted from the measurements and, as a consequence, renders the higher percentiles useless. A tool like cassandra-stress prevents coordinated omission from occurring.

▶ Read more here.

Don't average percentiles

Another typical problem with benchmarks is that when a load is generated by multiple load generators, the percentiles are averaged. The correct way to determine the percentiles over multiple load generators is to merge the latency distribution of each load generator and then to determine the percentiles.

If this isn't an option, then the next best alternative is to take the maximum (the p99, for example) of each of the load generators.
The actual p99 will be equal to or less than the maximum p99.

▶ Read more here.

Use proven benchmark tools

Instead of rolling out custom benchmarks, use proven tools like cassandra-stress. Cassandra-stress is very flexible and takes care of coordinated omission. Yahoo! Cloud Serving Benchmark(YCSB) is also an option, but needs to be configured correctly to prevent coordinated omission. TLP-stress is not recommended because it suffers from coordinated omission.

When benchmarking make sure to use the cassandra-stress that is part of the Scylla distribution because it contains the shard-aware drivers.

Use the same benchmarking tool

When benchmarking with different tools, it is very easy to run into an apples vs oranges comparison. When comparing products, use the same benchmark tool, if possible.

Reproducible results

Make sure that the outcomes of your benchmark are reproducible, so execute your tests at least twice. If the outcomes are different, then the benchmark results are unreliable. One potential cause could be that the data set of a previous benchmark has not been cleaned, which can lead to a performance difference for writes.

Query recommendations

Correct data modeling

The key to a well performing system is using the properly defined data model. A poorly structured data model can easily lead to an order-of-magnitude performance difference compared to a proper model.

A few of the most important tips:

- Choose the right partition key and clustering keys. Reduce or even eliminate the amount of data that needs to be scanned.
- Add indexes where appropriate.
- Partitions that are accessed more than others (hot partitions) should be avoided because they cause load imbalances between CPUs and nodes.
- Large partitions, large rows and large cells should be avoided because they can cause high latencies.

Use prepared statements

Prepared statements provide better performance because: parsing is done once, token/shard aware routing and less data is sent. Apart from performance improvements, prepared statements also increase security because they prevent CQL injection.

▶ Read more here.

Use paged queries

It is best to run queries that return a small number of rows. But if a query could return many rows, then an unpaged query can lead to a huge memory bubble and Scylla could eventually decide to kill the query. With a paged query, the execution collects a page's worth of data and new pages are retrieved on demand, leading to smaller memory bubbles.

▶ Read more here.

Don't use reverse queries

When using a query with an ORDER BY clause, you need to make sure that the order is the same as in the data model. Otherwise you run into a problem called reverse queries, which can cause unbound memory usage and killed queries.

Use workload prioritization

In a typical application there are operational workloads that require low latency. Sometimes these run in parallel with analytic workloads that process high volumes of data and do not require low latency. With workload prioritization, one can prevent the analytic workloads from negatively impacting the latency-sensitive operational workload.

▶ Read more here.

Bypass cache

There are certain workloads, e.g. analytical workloads, that scan through all the data. By default Scylla will try to use cache, but since the data won't be used again, it leads to cache pollution — good data is pushed out of the cache and replaced by useless data.

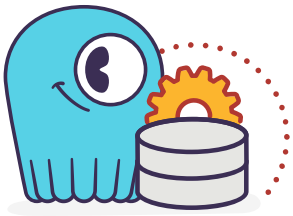
This can result in bad latency on operational workloads due to increased rate of cache misses. To prevent this problem, queries from analytical workloads can bypass the cache using the 'bypass cache' option.

▶ Read more here.

Batching

Multiple CQL queries to the same partition can be batched into a single call. Imagine the round trip time being 0.9 ms and the service time time 0.1 ms. Without batching the total latency would be 10x(0.9+0.1)=10.0 ms. But if you would create a batch of 10 instructions, the total time would be 0.9+10*0.1=1.9 ms. That is 19% of the latency compared to no batching.

▶ Read more here.



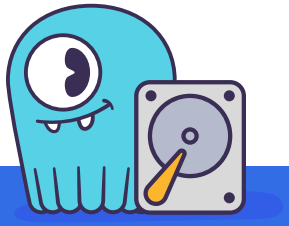
Driver guidelines

Use the Scylla drivers that are available for Java/Python/Go. They provide much better performance than third-party drivers because they are shard aware – they can route requests to the right CPU core (shard). When the driver starts, it gets the topology of the cluster and therefore it knows exactly which CPU core should get a request.

If Scylla drivers are not an option, make sure that at least a token-aware driver is used so one round trip is removed.

Check if there are sufficient connections created by the client, otherwise performance could suffer. The general rule is between 1-3 connections per Scylla CPU per node.

▶ Read more here.



CPU cores count guidelines

By default Scylla will make use of all CPU cores and is designed to perform well on powerful machines. As a result, it requires fewer machines. The recommended minimum number of CPU cores per node for operational workloads is 20.

The rule of thumb is that a single physical CPU can process 12.5 K queries per second with a payload of up to 1 KB. If a single node should process 400K queries per second, at least 32 physical CPUs or 64 hyper-threaded cores are required. In cloud environments hyper-threaded cores are often called virtual CPUs (vCPUs) or just cores. So it is important to determine if a virtual CPU is the same as a physical CPU or if it is a hyper-threaded CPU.

Scylla relies on temporarily spinning the CPU instead of blocking and waiting for data to arrive. This is done to lower latency due to reduced context switching.

The drawback is that the CPUs are 100% utilized and you might falsely conclude that Scylla can't keep up with the load.

▶ Read more here.

Memory guidelines

During the startup, Scylla will claim nearly all memory for itself. This is done for caching purposes to reduce the number of I/O operations. The more memory, the better the performance.

Scylla recommends at least 2 GB of memory per core and a minimum of 16 GB of memory for a system (pick the highest value). For example, if you have a 64-core system, you should have at least 2x64=128 GB of memory.

The max recommended ratio of storage/memory for good performance is 30:1. So for a system with 128 GB of memory, the recommended upper bound on the storage capacity is 3.8 TB per node of data. To store 6 TB of data per node, the minimum recommended amount of memory is 200 GB.

▶ Read more here and here.

Storage guidelines

Scylla can utilize the full potential of modern NVMe SSDs. The faster the drive, the better the Scylla performance. If there is more than one SSD,
it is recommended to use them as RAID 0 forbest performance. This is configured during the scylla_setup and Scylla will create the RAID
device automatically. If there is limited SSD capacity, the commit log should be placed on the SSD.

The recommended file system is XFS because
of its support for asynchronous appending writes and because it is the primary file system with which ScyllaDB is tested.

Because SSDs wear out over time, it is recommended to rerun the iotune tool every few months. This will help Scylla's IO scheduler make the best performing choices.

▶ Read more here.

Networking guidelines

For operational workloads the minimum recommended network bandwidth is 10 Gbps. The scylla_setup script takes care of optimizing the kernel parameters, IRQ handling, etc.

▶ Read more here.

Cloud compute instance recommendations

Scylla is designed to utilize all hardware resources. Bare metal instances are preferred for best performance.

- Read more here.

Image guidelines

Use Scylla provided AMI on AWS EC2, if possible. They have already been correctly configured.

AWS

AWS EC2 i3, i3en and cd5 bare metal instances are highly recommended because they are optimized for high I/O.

- Read more here.

If bare metal isn't possible, use Nitro-based instances and run with 'host' as tenancy policy. This will prevent the instance being shared with other VMs.

If the recommendation above isn't possible, we recommend instance storage over EBS. If instance store is not an option, use an io2 IOPS provisioned SSD for best performance. If there is limited support for instance storage, place the commit log there. There is a new instance type available called r5b that has high EBS performance.

- Read more here.

GCP

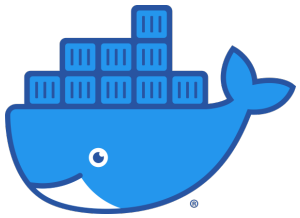
For GCP we recommend n1/n2-highmem with local SSDs.

- Read more here.

Azure

For Azure we recommend the Lsv2 series. They feature high throughput and low latency and have local NVMe storage.

- Read more here.



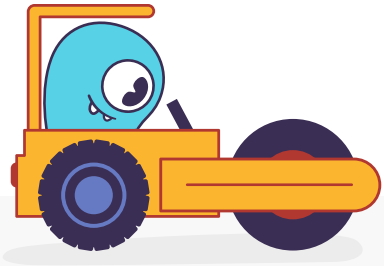
Docker

When running in the Docker platform, please use CPU pinning and host networking for the best performance.

- Read more here.

Kubernetes

As with Docker, CPU pinning should be used on Kubernetes environments as well. In this case the pod should be pinned to a CPU so that no sharing takes place.



Data Compaction

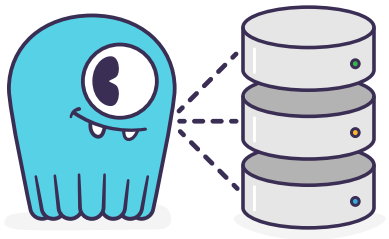
When records are updated or deleted, the old data eventually needs to be deleted. This is done using compaction. The compaction settings can make a huge difference. Check the following matrix to understand how to configure compaction for your use case:

Incremental Compaction Strategy (ICS) is a feature exclusively available in Scylla Enterprise. It combines the low space amplification of LCS with the low write amplification of STCS. ICS is the default strategy for Scylla Enterprise.

If you have time series data, the TWCS should be used.

- Read more here.

Workload/Compaction Strategy	STCS	LCS	ICS	TWCS
Write only	+	-	+	-
Overwrite	+	-	+	-
Read mostly, with few updates	-	+	-	-
Read-mostly with many updates	+	-	+	-
Time series	-	-	-	+



Consistency Level

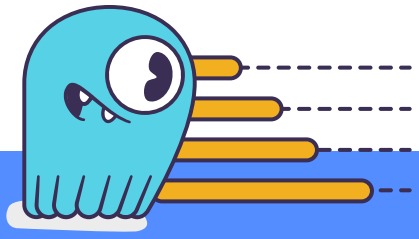
The consistency level determines how many nodes the coordinator should wait for in order for the read or write to be considered a success. The consistency level is determined by the application based on requirements for consistency, availability and performance. The higher the consistency, the lower the availability and the performance.

For single data center setups a frequently used consistency level for both reads and writes is QUORUM. It gives a nice balance between consistency and availability/ performance. For multi-datacenter setups it is best to use LOCAL_QUORUM.

▶ Read more here.

Replication Factor

The recommended replication factor is set to 3, and in most cases this is a sensible default because it provides a good balance between performance and availability. Keep in mind that a write will always be sent to all replicas, no matter the consistency level.



Asynchronous Requests

Using asynchronous requests can help to increase the throughput of the system. If the latency would be 1 ms, then 1 thread at most could do 1000 QPS. But if an operation takes a service time of 100 us, with pipelining the throughput could increase to 10.000 QPS.

To prevent overload due to asynchronous requests, the drivers limit the number of pending requests to prevent overloading
the server.

▶ Read more here.

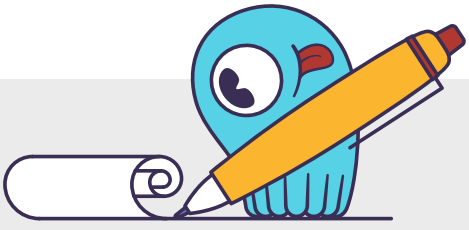
Conclusion

Scylla has excellent performance out of the box. Following the best practices described in this paper will prevent mistakes that might diminish the performance of your Scylla deployment.

ABOUT SCYLLADB

Scylla is the real-time big data database. API-compatible with Apache Cassandra and Amazon DynamoDB, Scylla embraces a shared-nothing approach that increases throughput and storage capacity as much as 10X. Comcast, Discord, Disney+ Hotstar, Grab, Medium, Starbucks, Ola Cabs, Samsung, IBM, Investing.com and many more leading companies have adopted Scylla to realize order-of-magnitude performance improvements and reduce hardware costs. Scylla's database is available as an open source project, an enterprise edition and a fully managed database as a service. ScyllaDB was founded by the team responsible for the KVM hypervisor. 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