SCYLLA

3 Technical Shifts that Reduce Distributed Database Costs



For teams working on data-intensive applications, the database can be a low-hanging fruit for significant cost reduction. If you're working with data-intensive applications, the total cost of operating a highly-available database can be formidable – whether you're working with open source on-premises, fully-managed database-as-a-service or anything in between. That "total cost" goes beyond the price of the database and the infrastructure it runs on; there's also all the operational expenses related to database tuning, monitoring, admin, and so on to consider.

This paper looks at how teams can reduce that total cost of ownership by making 3 technical shifts:

- Reduce node sprawl by moving to fewer, larger nodes
- Reduce manual tuning and configuration by letting the database manage itself
- Reduce infrastructure by consolidating workloads under a single cluster

REDUCE CLUSTER SPRAWL BY MOVING TO FEWER, LARGER NODES

For over a decade, NoSQL's promise has been enabling massive horizontal scalability with relatively inexpensive commodity hardware. This has allowed organizations to deploy architectures that would have been prohibitively expensive and impossible to scale using traditional relational database systems.

But a focus on horizontal scaling results in system sprawl, which equates to operational overhead, with a far larger footprint to keep managed and secure. Big clusters of small instances demand more attention, are more likely to experience failures, 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.

Let's look at a real-world example. The diagram below illustrates the server costs and administration benefits experienced by someone who moved from Apache Cassandra to ScyllaDB, a database that is able to dramatically reduce cluster size as a result of its unique close-to-the metal architecture. This customer migrated a Cassandra installation, distributed across 120 i3.2xl AWS instances with 8 virtual CPUs each, to ScyllaDB. Using the same node sizes, ScyllaDB 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.

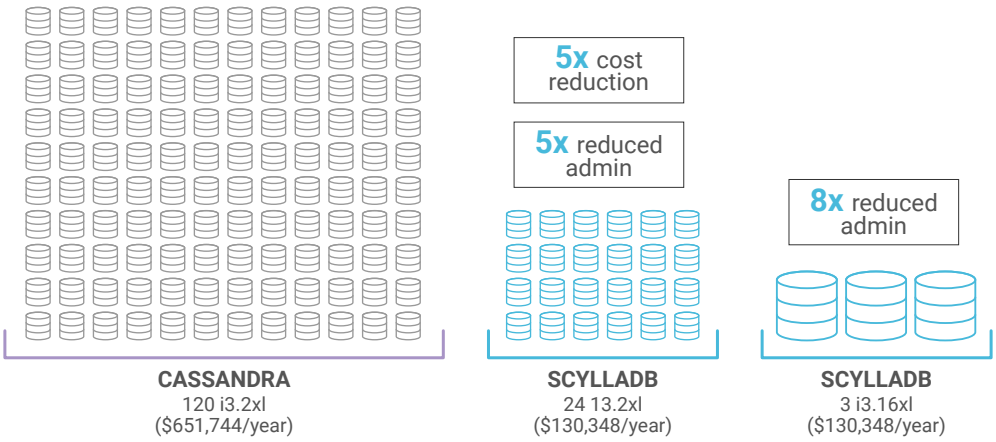


Figure 1: ScyllaDB reduces server cost by 5X and improves MTBF by 40X

ScyllaDB is built with a unique close to the metal architecture, which takes full advantage of modern infrastructure. This allowed the customer to move to the biggest nodes available at the time, 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.

Reducing the size of your cluster also offers additional advantages beyond reducing the administrative burden:

- **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 likely 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.

REDUCE MANUAL TUNING AND CONFIGURATION BY LETTING THE DATABASE MANAGE ITSELF

Distributed database tuning often requires specialized resources and consumes much more time than any team really would like to allocate to it.

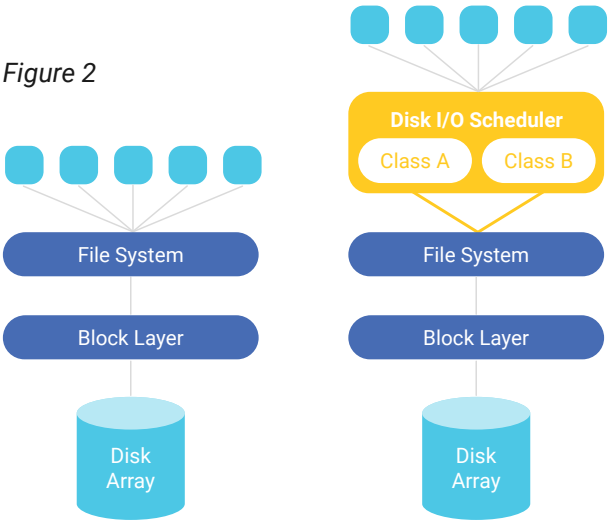
In any distributed database, many actors compete for available disk I/O bandwidth. Databases like Cassandra and Redis control I/O submission by statically capping background operations such as compaction, streaming, and repair. Getting that cap right requires painstaking, trial-and-error tuning combined with expert knowledge of specific database internals. Spiky workloads, with wide variance between reads and writes, or unpredictable end-user demand, are risky. Set the cap too high and foreground operations will be starved of bandwidth, creating erratic latency and bad customer experiences. Set it too low and it might take days to stream data between nodes, making auto-scaling a nightmare. What works today may be disastrous tomorrow.

Users are largely relieved of tuning when the database can automatically prioritize its own activities due to real-time, real-world conditions. For instance, ScyllaDB achieves this with several embedded schedulers/actuators, each of which is responsible for resources such as network bandwidth, disk I/O, and CPU. For example, ScyllaDB can enable application-level throttling, prioritize requests and process them selectively, and cancel requests before they impact a lower layer's software stack. It achieves a certain level of concurrency inside the disk to get maximum bandwidth from it, but not to make this concurrency too high in order to prevent the disk from queueing requests internally for longer than needed.

This means that even under the most intense workloads, ScyllaDB runs smoothly without requiring frequent administrator supervision and intervention. Users don't have to worry about tuning the underlying Linux kernel for better performance, or play with any Garbage Collector setting whatsoever (ScyllaDB is implemented in C++, so there is no Garbage Collection).

Figure 2 below shows this architecture at a high level. In ScyllaDB, requests bypass the kernel for processing and are sent directly to ScyllaDB's user space disk I/O scheduler. The I/O scheduler applies rich processing to simultaneously maintain system stability and meet SLAs.

Figure 2



On the left, requests generated by the userspace process are thrown directly into the kernel and lower layers. On the right, ScyllaDB's disk I/O scheduler intermediates requests. ScyllaDB classifies requests into semantically meaningful classes (A and B), then tracks and prioritizes them – guaranteeing balance while ensuring that lower layers are never overloaded.

The tedium of managing compaction can also be reduced. Configuring compaction manually requires intimate knowledge of both expected end user consumption patterns along with low-level database-specific configuration parameters. That can be avoided if the database uses algorithms to let the system self-correct and find the optimal compaction rate under varying loads. This radically lowers the risk of catastrophic failures, and also improves the interplay between maintenance and customer facing latency.

The impact can be seen in the following performance graphs.

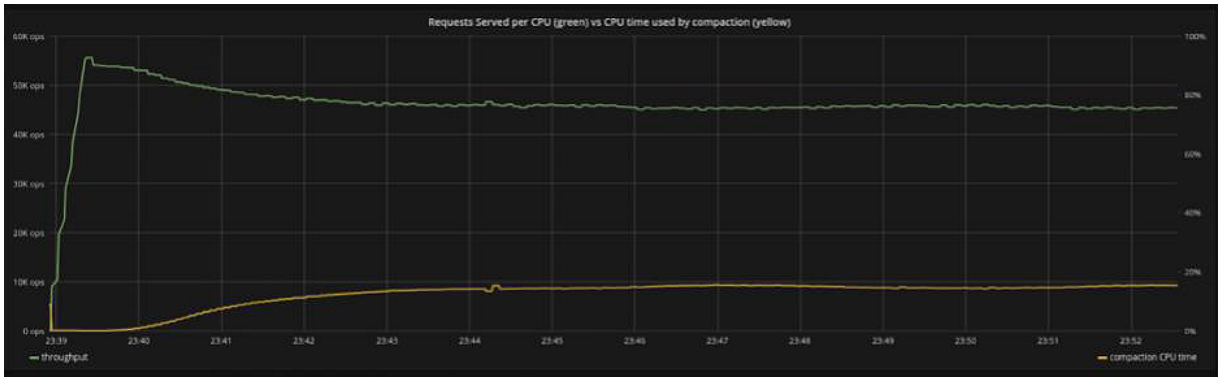


Figure 3: Throughput of a CPU in the system (green), versus percentage of CPU time used by compactions (yellow). In the beginning, there are no compactions. As time progresses the system reaches steady state as the throughput steadily drops.

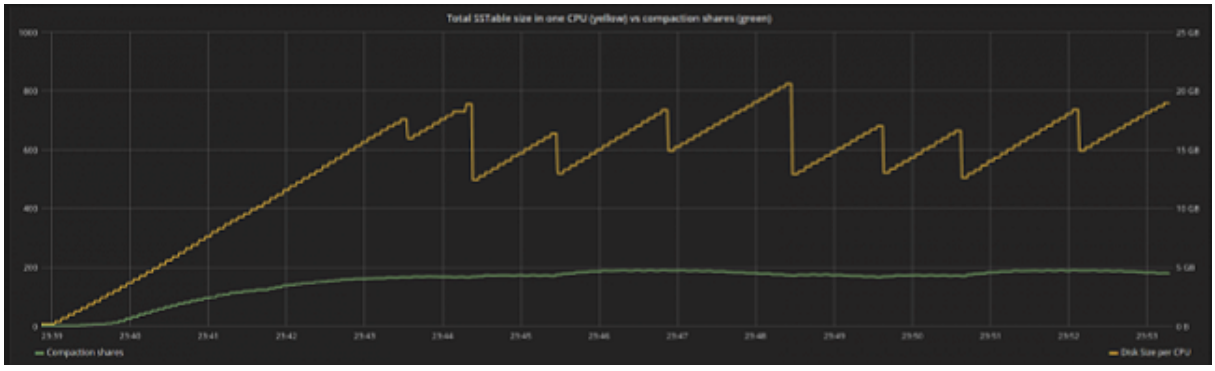


Figure 4: Disk space assigned to a particular CPU in the system (yellow) versus shares assigned to compaction (green). Shares are proportional to the backlog, which at some point will reach steady state.



Figure 5: 95th, 99th, and 99.9th percentile latencies (represented by green, yellow and blue lines respectively).

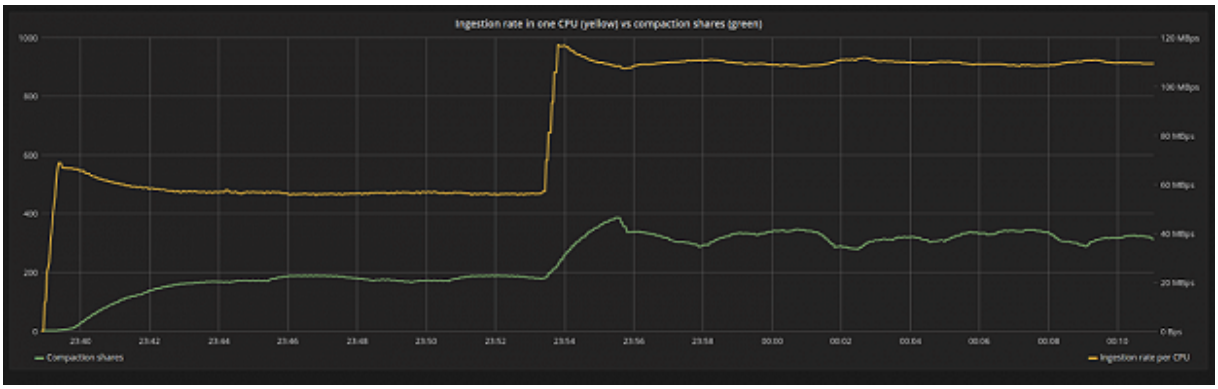


Figure 6: The ingestion rate (yellow line) suddenly increases from 55MB/s to 110MB/s, as the payload of each request increases in size. The system is disturbed from its steady state position but finds equilibrium for the backlog (green line).

A sudden increase in requests requires the system to ingest data faster. As the ingestion bandwidth increases and data gets persisted into disk more aggressively, the compaction backlog also increases. The effects are shown on figure 6. With the new ingestion rate, the system is disturbed and the backlog grows faster than before. However, the compaction controller will automatically increase shares of the internal compaction process and the system in turn achieves a new state of equilibrium

REDUCE INFRASTRUCTURE BY RUNNING MULTIPLE WORKLOADS ON A SINGLE CLUSTER

When looking to save costs, teams might consider running multiple different use cases against the database. It is often compelling to aggregate different use cases under a single cluster, especially when those use cases need to work on the exact same data set. Keeping several use cases together under a single cluster can also reduce costs. But, it's essential to avoid resource contention when implementing latency-critical use cases. Failure to do so may introduce hard-to-diagnose performance situations, where one misbehaving use case ends up dragging down the entire cluster's performance.

There are many ways to accomplish workload isolation to minimize the resource contention that could occur when running multiple workloads on a single cluster. Here are a few that we have seen work well. Keep in mind that the approach to choose will depend on your existing database available options, as well as your use case's requirements:

- **Physical Isolation:** This setup is often used to entirely isolate one workload from another. It involves essentially extending your deployment to an additional region (which may be physically the same as your existing one, but logically different on the database side). As a result, the use cases are split to replicate data to another region, but queries are executed only within a particular region – in such a way that a performance bottleneck in one use case won't degrade or bottleneck the other. Note that a downside of this solution is that your infrastructure costs double.
- **Logical Isolation:** Some databases or deployment options allow you to logically isolate workloads without needing to increase your infrastructure resources. For example, ScyllaDB has a Workload Prioritization feature where you can assign different weights for specific workloads to help the database understand which workload you want it to prioritize in the event of system contention. If your database does not offer such a feature, you may still be able to run two or more workloads in parallel, but watch out against potential contentions on your database.
- **Scheduled Isolation:** Many times, you might need to simply run batched scheduled jobs at specified intervals in order to support other business-related activities, such as extracting analytics reports. In those cases, consider running the workload in question at low-peak periods (if any exist), and experiment with different concurrency settings in order to avoid impairing the latency of the primary workload that's running alongside it.

Let's look at ScyllaDB's Workload Prioritization in a bit more detail. This capability is often used to balance OLAP and OLTP workloads. The purpose of workload balancing is to provide a mechanism that ensures each defined task has a fair share of system resources such that no single job monopolizes system resources, starving other jobs of their needed minimums to continue operations.

On Figure 7, note that latency for both loads nearly converges. OLTP processing began at or below 2 ms for p99 up until the OLAP job began at 12:15. When OLAP was enabled OLTP p99 latencies shot up to 8 ms, then further degraded, plateauing around 11 –12 ms until the OLAP job terminates after 12:26. These latencies are approximately 6x greater than

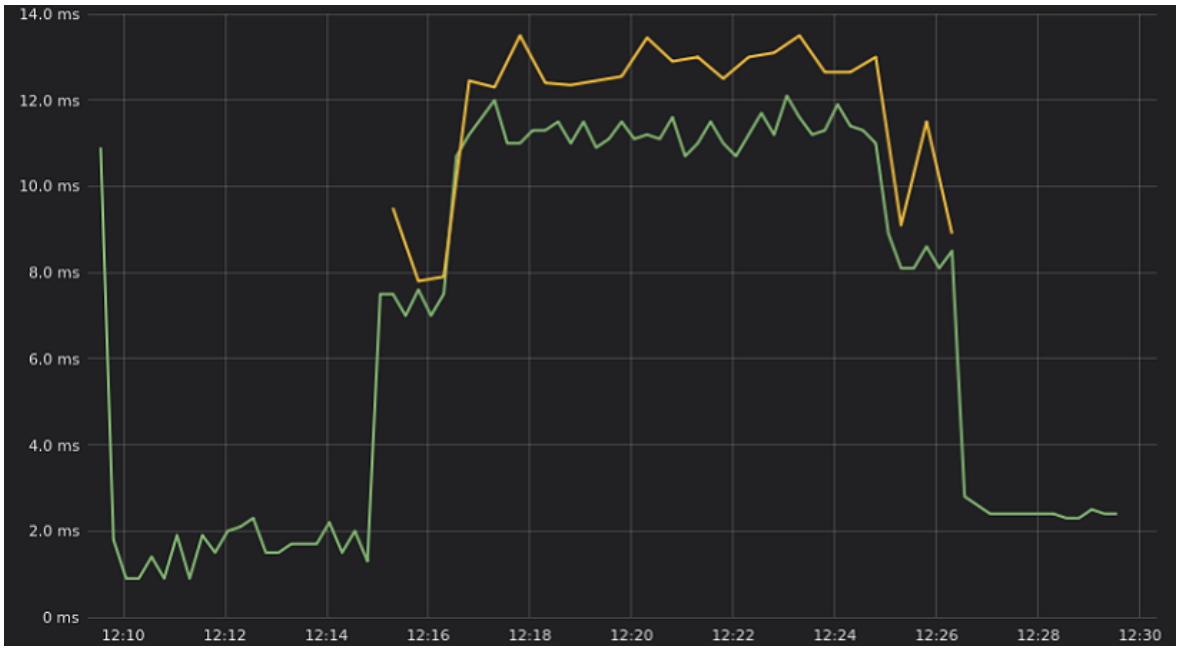


Figure 7: Latency between OLTP and OLAP on the same cluster before enabling workload prioritization

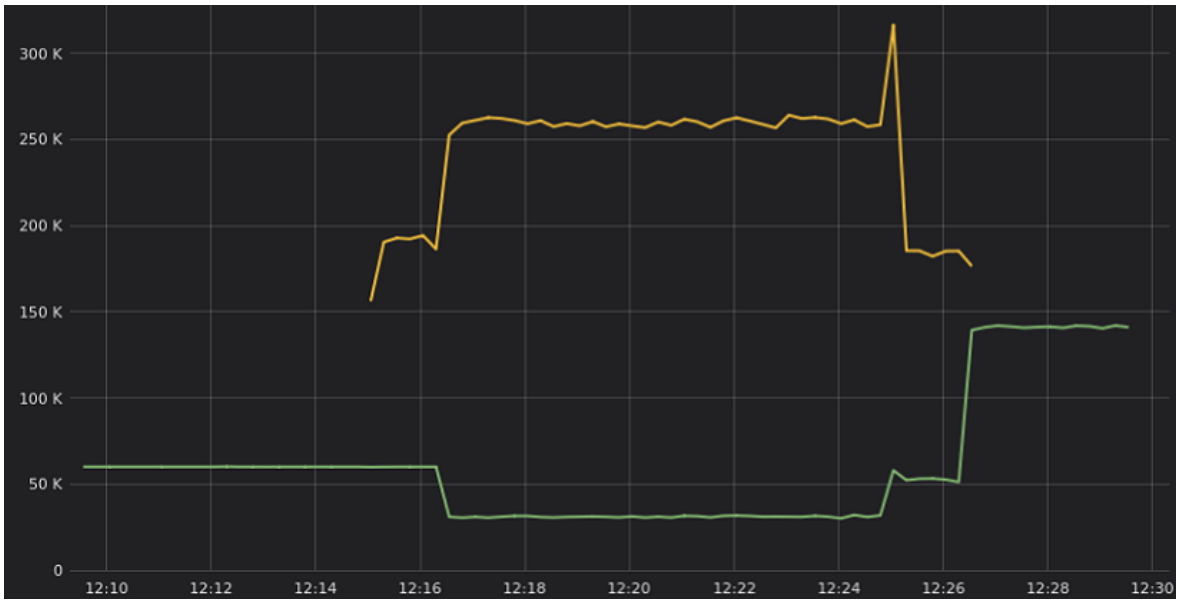


Figure 8: Comparative throughput results for OLTP and OLAP on the same cluster without workload prioritization enabled

when OLTP ran by itself. (OLAP latencies hover between 12 – 14 ms, but, again, OLAP is not latency-sensitive.).

On figure 8, note that throughput on OLTP sinks from around 60,000 ops to half that — 30,000 ops. You can see the reason why. OLAP, being throughput hungry, is maintaining roughly 260,000 ops.

The bottom line is that OLTP suffers with respect to both latency and throughput, and users experience slow response times. In many real-world conditions, such OLTP responses would violate a customer's SLA.

After workload prioritization is enabled, the OLTP workload similarly starts out at sub-millisecond to 2 ms p99 latencies. Once an OLAP workload is added performance degrades on OLTP processing, but with p99 latencies hovering between 4 – 7 ms (about half of the 11-12 ms p99 latencies when workload prioritization was not enabled). It is important to note that once system contention kicks in,

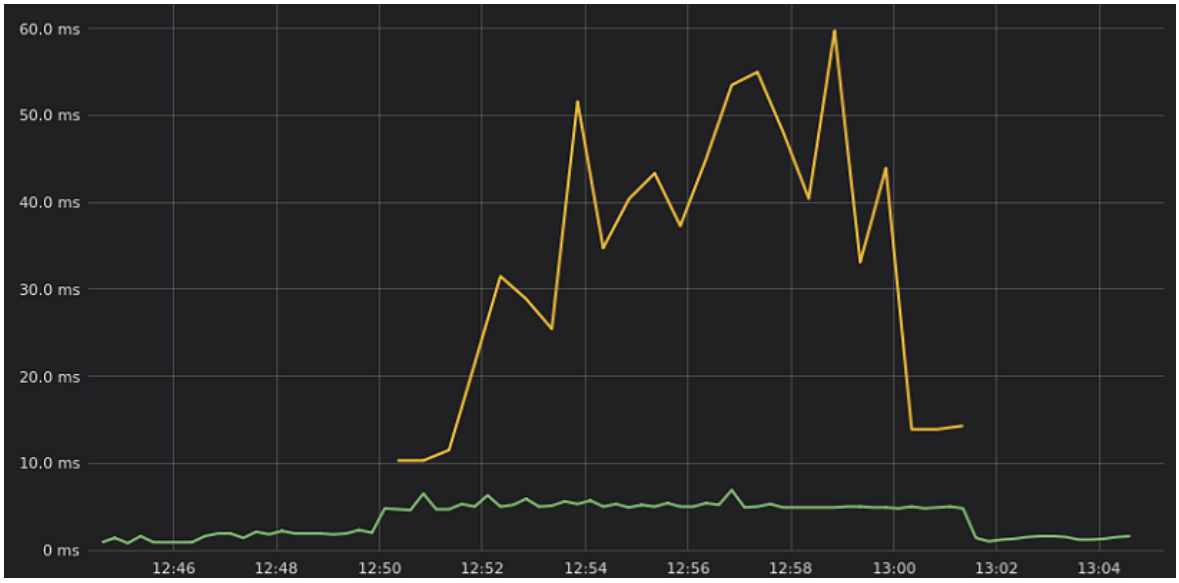


Figure 9: OLTP and OLAP latencies with workload prioritization enabled

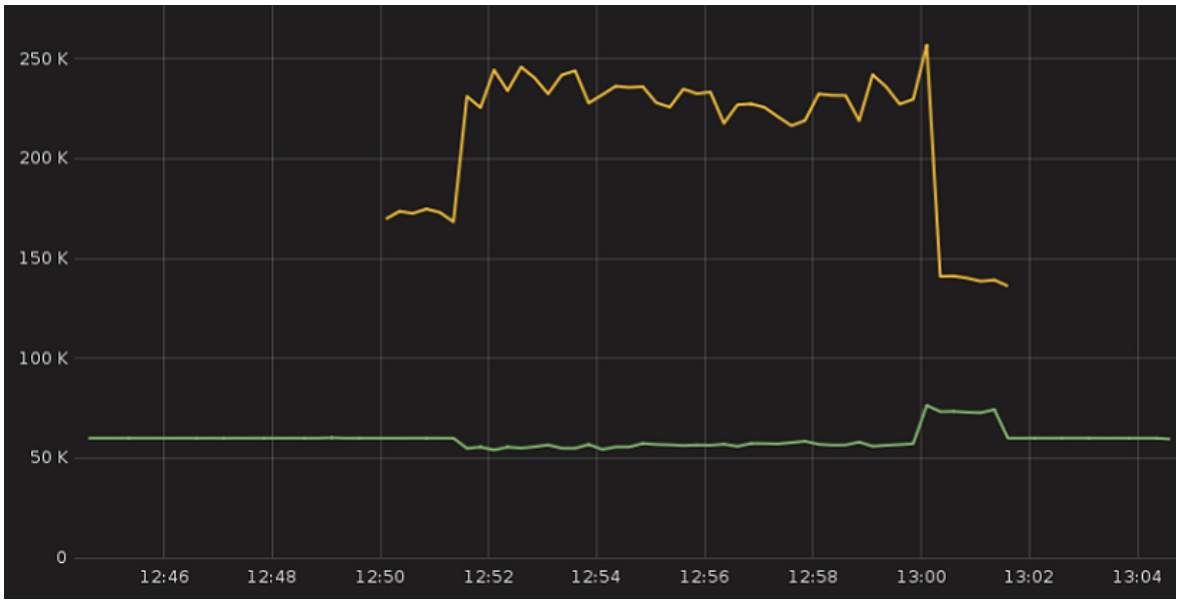


Figure 10: OLTP and OLAP latencies with workload prioritization enabled

the OLTP latencies are still somewhat impacted – just not to the same extent they were prior to workload prioritization. If your real-time workload requires ultra-constant single digit or less p99 millisecond latencies, then we strongly recommend that you avoid introducing any form of contention.

The OLAP workload, not being as latency-sensitive, has p99 latencies that hover between 25 – 65 ms. These are much higher latencies than before – the tradeoff for keeping the OLTP latencies lower.

On Figure 10, OLTP traffic is a smooth 60,000 ops until the OLAP load is also enabled. Thereafter it does dip in performance, but only slightly, hovering between 54,000 to 58,000 ops. That is only a 3% – 10% drop in throughput. The OLAP workload, for its part, hovers between 215,000 – 250,000 ops. That is a drop of 4% – 18%, which means an OLAP workload would take longer to complete. Both workloads suffer degradation, as would be expected for an overloaded cluster, but neither to a crippling degree.

Using this capability, many teams can avoid having one datacenter for OLTP real-time workloads and another for analytics workloads. That means less admin burden as well as lower infrastructure costs.

WRAP UP

These 3 technical shifts are just the start. Here are some additional ways to reduce your database spend:

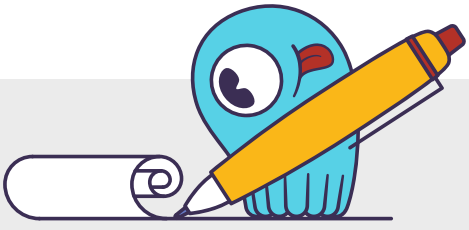
- Improve your price-performance with more powerful hardware and a database that's built to squeeze every ounce of power out of it, such as ScyllaDB's highly efficient shard-per-core architecture and custom compaction strategy.
- Consider Database as a Service options that allow flexibility in your cloud spend rather than lock you into one vendor's ecosystem.
- Move to a DBaaS provider with pricing better aligned with your workload, data set size and budget (compare multiple vendors' prices with this DBaaS pricing calculator).

And if you'd like advice on which, if any, of these options might be a good fit for your team's particular workload, use case and ecosystem, the architects at ScyllaDB would be happy to provide a technical consultation.

ABOUT SCYLLADB

ScyllaDB is the database for data-intensive apps that require high performance and low latency. It enables teams to harness the ever-increasing computing power of modern infrastructures – eliminating barriers to scale as data grows. Unlike any other database, ScyllaDB is built with deep architectural advancements that enable exceptional end-user experiences at radically lower costs. Over 400 game-changing companies like Disney+ Hotstar, Expedia, FireEye, Discord, Crypto.com, Zillow, Starbucks, Comcast, and Samsung use ScyllaDB for their toughest database challenges. ScyllaDB is available as free open source software, a fully-supported enterprise product, and a fully managed service on multiple cloud providers. For more information: ScyllaDB.com

SCYLLADB.COM



SCYLLA

United States Headquarters
1309 S Mary Ave
Sunnyvale, CA 94087 U.S.A.
Email: info@scylladb.com

Israel Headquarters
11 Galgalei Haplada
Herzelia, Israel

Copyright © 2022 ScyllaDB Inc. All rights reserved. All trademarks or registered trademarks used herein are property of their respective owners.