



# Cut Long-Tail Latencies with a New Approach to NoSQL



## CONTENTS

ZEROING IN ON LONG-TAIL LATENCY MEASUREMENT	3
CAUSES OF LONG-TAIL LATENCY	4
HARDWARE AND NETWORK BOTTLENECKS	4
SPEED MISMATCH	5
EXCEEDING THE SYSTEM'S LATENCY BUDGET	5
EFFECTS OF LONG-TAIL LATENCY	5
THE LIMITATIONS OF 1ST GEN NOSQL: REAL TIME, BUT ONLY PART TIME	5
WINNING THE BATTLE FOR LOW LONG-TAIL LATENCY IN THE REAL WORLD	7
DIGITAL MEDIA	7
INDUSTRIAL INTERNET OF THINGS (IIOT)	7
CYBERSECURITY	8
GUIDELINES FOR MASTERING LOW LONG-TAIL LATENCY	9
SCYLLADB REINS IN LONG-TAIL LATENCY	9

## ZEROING IN ON LONG-TAIL LATENCY MEASUREMENT

The explosion of real-time digital businesses such as digital media, ecommerce, and industrial IoT, has put the focus of infrastructure squarely on performance. In the end, all companies are impacted by this metric. Whereas peak performance was once commonly measured in milliseconds, storage performance is now below the [microsecond](#) threshold. That's driving a renewed focus on understanding and minimizing latency.

Whereas peak performance was once commonly measured in milliseconds, storage performance is now below the microsecond threshold.

Medium	Year Introduced	Latency	Transfer rate
7,200 RPM HDD	1992	4 ms	Up to 80-160 MB/s
15,000 RPM HDD	2000	2 ms	Up to 315 MB/s
SATA SSD	2004	30-100 $\mu$ seconds	Originally 40 MB/s Up to 550 MB/s
NVMe PCIe SSD	2013	5-10 $\mu$ seconds	Originally 1,000 MB/s Up to 3,500 MB/s
Persistent Memory App Direct mode	2019	100 - 340 ns	Up to 6,800 MB/s

Sources: [SSD Market History](#), [A Brief History of Disk Storage](#), [SATA vs. NVMe, Is it time for NVMe?](#), [Seagate Introduces New Generation of Enterprise Performance HDDs with NAND Caching](#), [Restoring the Balance Between Bandwidth and Latency](#)

Given that, it is surprising that many IT organizations fail to address latency in a systematic way. Teams that test and monitor applications often focus on average (mean or median, i.e., P50) performance measurements. This is not surprising; most of the tools developed for application performance monitoring are built to report averages by default. But averages have a weakness: they conceal outliers. Performance outliers are extreme deviations from the norm that may have a large and unexpected impact on overall system performance, and hence on user experience.

It is easy to dismiss performance outliers. In fact, monitoring systems are sometimes configured

in ways that don't even measure outliers. For example, if a monitoring system is calibrated to measure latency on a scale of 0 to 1000 ms, it is going to overlook any larger measurements — thus failing to detect the serious issues of query timeouts and retries. Outlier measurements are also sometimes dismissed or discounted as affecting only a small cohort of users, such as power users. This can be difficult to ascertain without extensive, careful, and (in the end) probably pointless testing. As we'll see, it is critical for IT to ruthlessly track down and address outlier latencies.

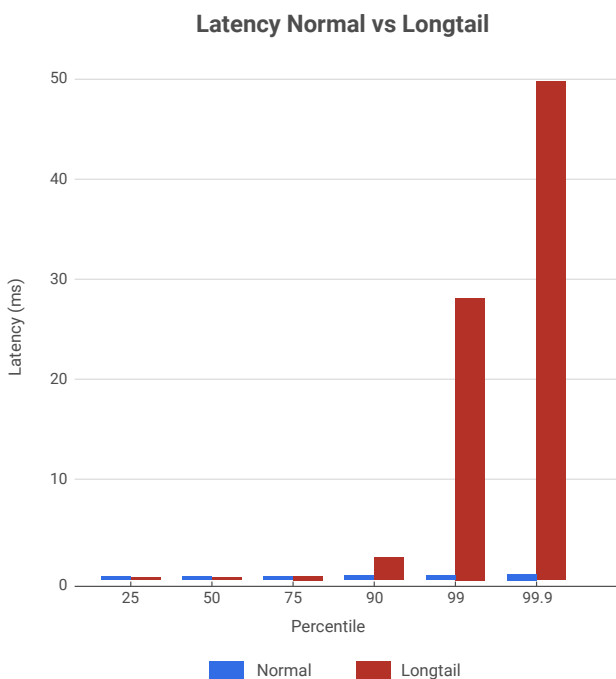
Furthermore, users never, or rarely, actually experience 'average performance.' Average

latency is a theoretical measurement that has little direct bearing on end user experience.

Latency is typically calculated in the median 50th, 90th, 95th, and 99th percentiles. These are commonly referred to as p50, p90, p95, and p99. They can go even higher, such as p999, which refers to the 99.9th percentile. Percentile-based metrics are used to expose the outliers that constitute the ‘long-tail’ of performance. Measurements of 99th (“p99”) and above are considered measurements of ‘long-tail latency,’ which is the focus of this paper.

A p50 measurement, for example, measures the median performance of a system. Imagine 10 latency measurements: 1, 2, 5, 5, 18, 25, 33, 36, 122, and 1000 milliseconds (ms). The p50 measurement is 18 ms. 50% of users experienced that latency or less. The p90 measurement is 122, meaning that 9 of the 10 latencies measured less than 122.

Yet, even percentile-based measurements can be misinterpreted. For example, it is a mistake to assume that “p99” means that only 1% of users experience that set of latencies. p99 (and higher) outlier latencies can actually have a disproportionate effect on the system as a whole.



Long-tail latency can have systemic causes; often caused by glitches in the underlying infrastructure.

For example, given the complexity of many modern websites, applications may need to fetch data from the database dozens or even hundreds of times to even paint a single web page for an end user. Just think about all the results you are offered from your favorite search engine, online shopping, social media, music or video site. As the complexity of websites and mobile applications continue to increase over time, the chances increase that any or every user starts experiencing these long-tail latencies on nearly every page they visit and every action they take.

## CAUSES OF LONG-TAIL LATENCY

Long-tail latency has many causes, but, significantly, it is not usually caused by application-specific problems or normal network lag. Instead, long-tail latency can have systemic causes; often caused by glitches in the underlying infrastructure. These glitches can include pauses due to Java Virtual Machine (JVM) ‘garbage collection,’ hypervisor interrupts, operating system context switches, database repair, cache flushes, and so on. This makes long-tail latency very tricky to diagnose and fix, as it’s often a ‘whack-a-mole’ exercise.

## HARDWARE AND NETWORK BOTTLENECKS

Long-tail latency can also be caused by hardware and network bottlenecks. IT commonly invests heavily in the best components available: great solid state drives (NVMe SSDs), CPUs, and memory. But they may choose software that was not designed with vertical scalability and full system utilization in mind — resulting in poor CPU utilization or poor storage I/O. Plus, they

often shoot themselves in the foot by configuring a 'narrow' network — throttling performance as they max out network I/O. For another example, it's common that SSD settings are misconfigured. Cloud SSDs have two associated throttles: burst and sustained. Background operations such as streaming and compaction can chew up the sustained throughput, which may be 1/10 of burst. While the backend process hogs I/O, there is no capacity left for bursts. The result is that theoretically highly performant hardware is bottlenecked by maximum CPU or network utilization.

## SPEED MISMATCH

As far as databases are concerned, there are two speeds that have to work together. The first is the combined speed of CPU and memory speed; the other is disk speed. CPU and memory tends to be faster (with performance measured in nanoseconds), while the disk tends to be slower (measured in microseconds to milliseconds). This equation is evolving as SSD manufacturers are catching up to CPU improvements, but it is still often the case.

You not only need to size your disks for basic data; you also need to calculate total storage needed based on your replication factor, as well as overhead to allow for compactions.

Speed mismatch is more commonly a problem with write workloads that are bounded by the disk. This can happen either because the disk is slow, or because the payloads are large. Large payloads shift the bottleneck to the disk. When the time comes to write to disk, any relative disk slowness will cause queries to back up in the buffer. This will result in outlier p99 latencies.

## EXCEEDING THE SYSTEM'S LATENCY BUDGET

Every system is designed with a 'latency budget'. In fact, you can view p95 and p99 latency as a function of the throughput that a given system is designed to support. The 'budget' itself is the ratio of latency to throughput.

Systems that handle massive amounts of data have primary and secondary considerations. These include storage, CPU cores, memory (RAM), and network interfaces. You not only need to size your disks for basic data; you also need to calculate total storage needed based on your replication factor, as well as overhead to allow for compactions.

Modern servers running on increasingly powerful multicore CPUs are common across AWS, Azure, and Google Cloud. Server needs are based upon how much throughput each of these multicore chips can support.

These are all components that define a system's latency budget. It's important to understand and stay within those limitations. When they are exceeded, outlier latencies are virtually assured to occur.

## EFFECTS OF LONG-TAIL LATENCY

Some web pages can generate well over 100 HTTP calls to various services. Many of those services run on multi-tier infrastructure. A hiccup at any layer can create a latency outlier, resulting in a poor experience for a user. A very active backend service that is affected by long-tail latencies can have a serious site-wide impact.

## THE LIMITATIONS OF 1ST GEN NOSQL: REAL TIME, BUT ONLY PART TIME

The basic premise of NoSQL is the ability to 'scale out' on cheap, commodity hardware. When more capacity is needed, administrators simply add servers to database clusters. This simplicity, however, evolved into a vulnerability. Clusters often grow out of control, resulting in a phenomenon called 'node sprawl.' The architecture of the first-gen NoSQL databases actively encourages larger clusters of less

powerful machines. From the perspective of latency, this architecture virtually ensures poor p99 performance due to more context switches, disk failures, and network hiccups – all of which create latency outliers.

For example, the widely-used MongoDB database is susceptible to poor long-tail performance due to its master/slave architecture. Once MongoDB is set up and the pipe is configured, throughput is essentially fixed. Once the database is running, the only remaining mechanism that is available for optimizing performance is sharding. Configuring *sharding* is a notoriously risky undertaking that often requires weeks of trial-and-error experimentation.

In contrast to MongoDB, Apache Cassandra has a peer-to-peer architecture. But even without a master/slave approach, Cassandra has proven to have its own weaknesses.

First, Cassandra is implemented in Java, making it vulnerable to the weaknesses of that platform. It is mainly susceptible to pauses caused by garbage collection (GC). Some monitoring teams even have a dedicated metric devoted to GC stall percentage. Compaction and repair operations, which become more onerous in larger clusters, add to the outliers that plague Cassandra deployments.

Second, Cassandra has limited ability to utilize modern hardware for compaction and streaming. Since Cassandra is an ‘append-only’ system, keeping data coherent for reads requires a compaction process. Servers with a high number of cores are available on IaaS and on-prem deployment. However, Cassandra is not able to scale the compaction process linearly with the increased number of available cores. This limitation results in higher read latency and a need for more servers to be deployed in the clusters.

During streaming, Cassandra limits the amount of data streamed from nodes to control the load on the sending and receiving servers. Modern servers utilizing 10GbE networks or higher, SSDs, and fast CPUs should not be limited by the software using these servers. Cassandra throttles the streaming process to 200 mbits/

sec. Limiting streaming has a direct impact on latency. When rebuilding a node, your application is using reduced capacity. While repairing your cluster, your application might wind up waiting for server-side reconciliation of data discrepancy, contributing to higher read latency.

Cassandra is not able to scale the compaction process linearly with the increased number of available cores. This limitation results in higher read latency and a need for more servers to be deployed in the clusters.

Lastly, some workloads have higher read characteristics that are specific to their use case. The system therefore must favor the read path of data, which involves external caches or waiting for media to fetch and deliver. Conversely, write workloads look for direct fast access to the persistent volume. After all, a database is built to persist data. Cassandra does not recommend that users leverage the built-in table caching system; this means reads are fetched from disk, regardless of whether the data was ever changed after the first insertion.

Since Cassandra has different settings for storage media, JVM heap cache, and concurrency on read and writes, the burden is placed on administrators to properly configure the system for specific workloads. When workloads are unpredictable and spiky, administrators must constantly tune the Cassandra cluster to match read and write latencies.

Many IT organizations turn to external caches as a band-aid to insulate end users from the poor long-tail performance of their databases. But the [limitations of external caches](#) are well-

The best approach is to identify a system that intrinsically combines cache and persistent storage in a way that optimizes performance and data integrity.

documented. Caches are often used to enhance read performance, but write latencies derive little or no benefit. Transactional systems, which depend on real-time performance, cannot drop mutations, so caches offer no help. Transactional systems depend on low tail latency for writes as well as reads. If spikes are too frequent, the client experience deteriorates, becoming very sluggish. In the worst case scenario, data loss occurs.

Still, front end cache for reads is often a necessary evil, to the point where it is treated as obligatory. However, maintaining coherence between cache and persistent storage is a well-documented hassle that can easily bog down operations and negatively impact customers. The best approach is to identify a system that intrinsically combines cache and persistent storage in a way that optimizes performance and data integrity.

## WINNING THE BATTLE FOR LOW LONG-TAIL LATENCY IN THE REAL WORLD

Companies across the spectrum have come to realize that real-time performance can provide a significant competitive advantage. In the digital economy, switching costs for users are generally very low. With so much competition for user attention, a slow app or service will be mercilessly abandoned. In this section, we present use cases from a variety of industries where long-tail latencies were tamed to unlock business value.

## DIGITAL MEDIA

A great example of the need for consistently low p99 latency is Comcast, one of America's leading providers of communications, entertainment, and cable products and service. Comcast's X1 Platform enables viewers to discover, play, pause, and save videos, and also to maintain a history so that viewers can easily access 'last watched' programs. As a provider of critical services that require real-time performance, the platform is highly sensitive to long-tail latency.

The X1 Platform has a massive user base. Over the course of seven years, the X1 platform has grown from 30,000 devices to 31 million set-top boxes. The X1 Scheduler processes a staggering 2 billion RESTful calls daily.

Comcast originally built their system against Cassandra. For the reasons that we have touched on in this paper, Cassandra was not meeting their needs for low long-tail latency, so the X1 team decided to research alternatives. Comcast evaluated their existing database, Cassandra, against ScyllaDB, the database for data-intensive apps that require high performance and low latency.

Comcast turned to ScyllaDB to achieve better long-tail latencies than with Cassandra. To compare the two databases, Comcast benchmarked the platform prior to deploying it in production. A key measurement for Comcast turned out to be p99, and even p99.9. As Comcast discovered, performance characteristics of different databases become even more starkly differentiated in these edge cases.

By paying close attention to long-tail performance, Comcast has been able to maximize real-time performance where it's most important: the user experience. As a side-effect, Comcast was able to reduce node counts, and hence lower the overall TCO of their system.

## INDUSTRIAL INTERNET OF THINGS (IIOT)

Companies working in IIoT have unique requirements when it comes to long-tail performance. First is the need to ingest huge

<b>Cassandra</b>	<b>Median</b>	<b>90%</b>	<b>99%</b>	<b>99.90%</b>
DeviceRecording-Read	3.959	22.338	84.318	218.15
DeviceRecording-Write	1.248	4.094	31.914	117.344
<b>Scylla</b>	<b>Median</b>	<b>90%</b>	<b>99%</b>	<b>99.90%</b>
DeviceRecording-Read	0.523	0.982	3.839	9.786
DeviceRecording-Write	0.609	0.913	2.556	7.152

amounts of data very rapidly. IIoT is based on sensor data and telemetry from industrial equipment and robots in the field and on the factory floor. This data is processed and analyzed to deliver real-time alerts as well as historical views over long time periods. The result is that IIoT companies often must deploy two fairly redundant data infrastructures: one for operational workloads (OLTP) and the other for analytics workloads (OLAP).

Augury, a leading AI-based Machine Health solution provider, provides a nice example of how low long-tail latency helps in IIoT use cases. Augury helps their customers perform timely maintenance by delivering real-time insights and historical analytics about the condition of machines on the manufacturing floor. Beyond alerting, Augury provides actionable suggestions on the root cause of faults, recommendations on which checks to perform, and where to best focus the maintenance that's needed to keep machines running smoothly and avert catastrophic failures. The results are impressive — averages of 75% fewer breakdowns, 30% lower asset costs, 45% higher uptime.

Augury initially built their predictive services against MongoDB. Yet as the company began to grow and the dataset reached the limits of MongoDB, they realized the need for data infrastructure that could scale horizontally. Augury turned to ScyllaDB to achieve low long-tail latencies and analytics capabilities from a unified database infrastructure.

By paying close attention to long-tail performance, Comcast has been able to maximize real-time performance where it's most important: the user experience. As a side-effect, Comcast was able to reduce node counts, and hence lower the overall TCO of their system.

## CYBERSECURITY

Cybersecurity firms are also highly sensitive to long-tail latency. Insights that can prevent breaches are derived in real-time from massive data sets that span devices, services, and endpoints. Low tail latency not only enhances the end user experience; it also helps to more quickly identify and prevent threats and hacks.

FireEye (now known as Trellix), a top cybersecurity company, provides a great example. FireEye's Threat Intelligence application centralizes, organizes, and processes threat intel data to support analysts. Threat data is organized into a graph representing relationships among actors, such as hackers and criminal organizations, and threat vectors represented by URLs, email, IP addresses, and malware signatures.

ScyllaDB is uniquely able to support petabyte-scale workloads, millions of IOPs, and P99 latencies <10 msec while also significantly reducing total cost of ownership.

FireEye discovered that a system is only as strong or as performant as its underlying database solution. By paying attention to long-tail latencies, FireEye was able to make the overall system 100x faster than before; a query that traverses 15,000 graph nodes now returns results in about 300ms.

As with Comcast, this is partially the result of a smaller infrastructure footprint. FireEye dramatically slashed the storage footprint, while preserving the 1000-2000% performance increase they had experienced by switching to ScyllaDB. Ultimately, they reduced AWS spend to 10% of the original cost.

Lookout is another cybersecurity provider whose mission is to protect data in today's privacy-focused world. Lookout's benchmark simulated 38 million devices generating 110,000 messages per second. The results were average latencies in milliseconds with the ScyllaDB cluster running at very high levels of utilization (between 75% and 90%). Best of all, however, Lookout found that ScyllaDB delivered consistent single-digit millisecond p99 latencies.

## GUIDELINES FOR MASTERING LOW LONG-TAIL LATENCY

With these issues in mind, there are a number of guidelines that your organization can follow to master long-tail latency.

- Design a system that ensures p99 latency based on your organization's throughput targets.

- Do not ignore performance outliers. Embrace them and use them to optimize system performance.
- Minimize the possibility of outliers by minimizing infrastructure. For a start, avoid external database caches. Shrink database clusters by scaling vertically on more powerful hardware.
- Adopt database infrastructure that can make optimal use of powerful hardware.
- Adopt a peer-to-peer database architecture.
- Minimize the vectors needed to resize the system. Ideally, limit vectors to system resources, such as CPU cores.
- Avoid databases that add complexity. Look for databases with a close-to-the hardware architecture.
- Take into account the price-performance of your database. You do not need to break your IT budget to achieve real-time long-tail performance.

## SCYLLADB REINS IN LONG-TAIL LATENCY

In summary, latency outliers will always exist, but the trick is limiting their magnitude.

ScyllaDB's ability to tame long-tail latencies has led to its adoption in industries such as media, cybersecurity, the industrial Internet of Things (IIoT), AdTech, retail, and social media.

ScyllaDB is built on an advanced, open-source C++ framework for high-performance server applications on modern hardware. The team that invented ScyllaDB has deep roots in low-level kernel programming. They created the KVM hypervisor, which now powers Google, AWS, OpenStack, and many other public clouds.

Modern hardware is capable of performing millions of I/O operations per second (IOPS). ScyllaDB's [asynchronous architecture](#) takes advantage of this capability to minimize p99 latencies. ScyllaDB is uniquely able to support petabyte-scale workloads, millions of IOPs, and P99 latencies <10 msec while also significantly reducing total cost of ownership.

# 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](https://scylladb.com)

SCYLLADB.COM



**United States Headquarters**  
2445 Faber Place, Suite 200  
Palo Alto, CA 94303 U.S.A.  
Email: [info@scylladb.com](mailto:info@scylladb.com)

**Israel Headquarters**  
11 Galgalei Haplada  
Herzeliya, Israel



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