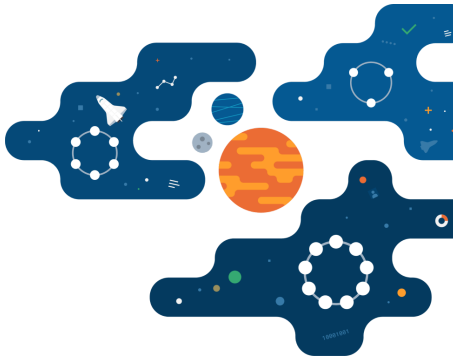
WHITEPAPER



Get on the Apache Cassandra™ 4.0 Bandwagon

Latest Cassandra Version Doubles Down on Performance and Reliability to Create Engaging, Fast Data Customer Experience

Innovation and enhancements in Apache Cassandra™ version 4.0 have resulted in improved performance and reduced costs.

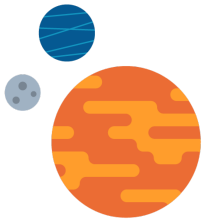
Cassandra 4.0 has increased the number of requests it can process per second by 25% to 70% while reducing the "tail latency" for those requests by a factor of 7; and with the addition of backpressure management, applications can now verifiably achieve peak performance. These improvements are complemented by enhancements to the compaction and repair maintenance process that reduce memory, disk IO, and CPU usage dramatically, resulting in greater system stability. Put all these improvements together and Cassandra 4.0 is faster and cheaper to operate than any previous version.

The amazing performance boost and lower TCO of Apache Cassandra 4.0 position it as the default platform for Enterprises to build fast data experiences for customers that deliver reactive engagement at the point of interaction.

DataStax

01

Introduction



Apache Cassandra™ is an ***open source distributed database*** trusted by thousands of companies for scalability, high availability, and outstanding performance.

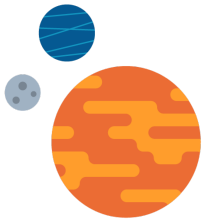
Linear scalability and proven fault-tolerance make it the perfect platform for mission-critical data, whether on commodity hardware, self-managed in the cloud, or ***as a managed service.***

As the latest release in the 12 year history of the project, Cassandra 4.0 is the most tested and hardened version ever. Compared to version 3.0 it has more than 3 times the number of improvements, and 2 times the number of bug fixes. All this was driven by real-world workload capture and replay, fuzz testing, and a dramatic expansion in test coverage and quality assurance. The result? Starting with the beta release 4.0 has been used in production at some of your favorite internet giants.

This release comes at a time when the growth of containers and Kubernetes as the dominant orchestration layer has created massive demand for a "kube" native data tier. Containers and Kubernetes have dramatically simplified the deployment and management of stateless, cloud-native, microservices. Enterprises are now looking for a data tier that can be deployed alongside their stateless services on Kubernetes. As a shared nothing, distributed database that supports scale out and scale in, Cassandra is the perfect match for Kubernetes. In-fact it is fairer to say, Cassandra has been waiting for Kubernetes to be invented.

02

Enhancements for the Enterprise CTO



Easier Integration with Improved Change Data Capture

Change Data Capture (CDC) allows databases to easily integrate into the wider data ecosystem. These integrations drive the "east-west" data movements in an enterprise that unlock modern digital products and experiences, by sharing data between silos. Examples include integrating "fast data" changes with "smart data" services such as data lakes, AI training, Apache Spark, Snowflake, or other operational data service such as Elastic Search. In these situations CDC is the feature that turns data into knowledge by enabling it to be shared in the Enterprise.

In previous Cassandra versions CDC was essentially a commit log archive service. Consumers would need to read, parse, and deduplicate the entries once the commit log segment was archived. Version 4.0 creates a real time CDC service by indexing the commit logs as they are written, improving performance and creating an abstraction from the core log implementation. These enhancements create a significantly more user-friendly scaffolding for CDC tools and consumers.

No Noisy Neighbors with Rogue Client Throttling

Misbehaving clients are a fact of life for databases, and historically they have been able to have an adverse impact on a Cassandra cluster. Previously all client requests were treated as equal, and assumed to be well intentioned and balanced. This worked for small, and tightly integrated, development and SRE (Site Reliability Engineering) teams but is harder when operating large teams and a large number of clients. Cassandra 4.0 introduces per client throttling as part of the Backpressure feature.

In Cassandra 4.0 the memory used to execute commands from each client (identified by an IP address) is managed and limited[1]. For most clients the memory usage grows and shrinks as its workload changes. However rogue clients can flood the cluster with 1,000s of commands without waiting for a response and dramatically impact all other clients.

[1] The limit can be set via the native_transport_max_concurrent_requests_in_bytes_per_ip configuration setting, which defaults to 1/40th of the available memory per client.

When a client hits the memory limits, one of two approaches is used. The default approach, which works with any client, is to use TCP backpressure to slow it down. The other approach is for the client to enable the cluster to send backpressure errors to it, this allows well behaved clients to perform at peak performance without adverse impacts on the cluster or other clients.

See the ***Backpressure*** section below for more details.

Expanded Use Cases with Compliance Audit Logging

The addition of robust Audit Logging to Cassandra makes it possible to deploy workloads that require a high level of regulatory compliance. The audit logging implementation in Cassandra 4.0 was donated by Netflix and designed to meet three goals: Performance, Accuracy, Usability & Extensibility. Above all the audit logging was implemented to meet the requirements of regulations such as SOX, PCI, GDPR, and internal company compliance policies.

The audit logging is highly configurable to meet any compliance policy. Configuration covers log size and location, format, included or excluded users and keyspaces, and the category of commands to audit. Command categories cover events such as logins, errors, user management, schema changes, data modification, and data reading. Each audit log entry includes information that allows the event to be traced back to the source client.

In common with the Full Query Logging, discussed below, the audit logging uses the ***OpenHFT Chronicle Queue*** from the high frequency trading sector and has no major impact on performance.

Lower Computing TCO with Improved Throughput

Improvements to latency and throughput, discussed below, can deliver between 25% and 70% better performance on the same hardware (depending on workload type). This level of improvement can result in a significant reduction in TCO as the same workload can be delivered using fewer nodes or servers with less computing resources. Enterprises with existing workloads should test and plan for deploying upgraded clusters with reduced hardware.

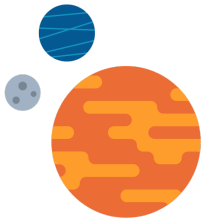
Long Term Support with Java 11

While Java 11 has been available since 2018, Cassandra has not been able to take advantage of the enhancements it provides. Aside from the performance implications, the lack of support also introduced problems with compliance and support when Oracle removed commercial support from Java 8 in 2019. Cassandra 4.0 requires Java 11, and benefits greatly from the improvements in Garbage Collection.

Java 11 is the current Long Term Support version of Java until Java 17 is released.

03

Enhancements for the Developer



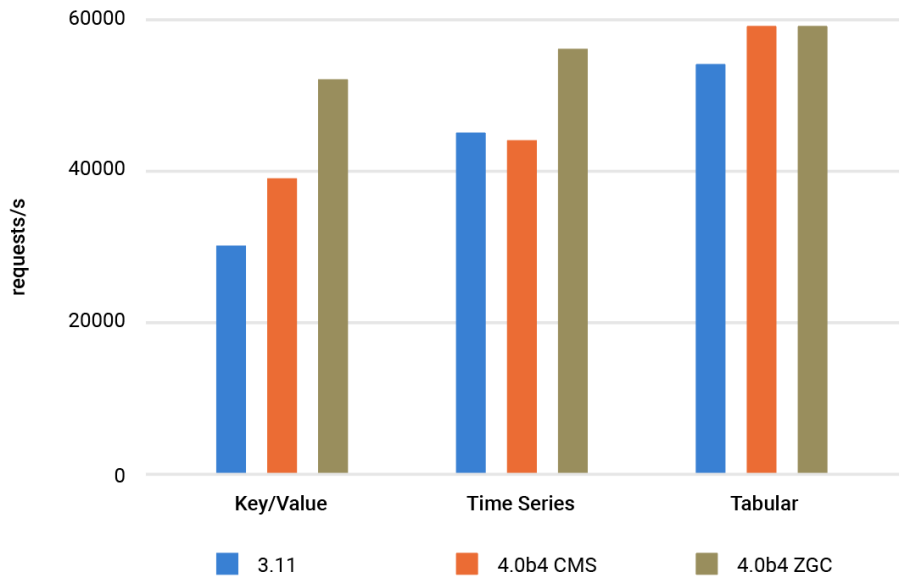
25% to 70% Better Throughput

Cassandra 4.0 can deliver between 25% and 70% better throughput than version 3.11. This dramatic improvement in performance comes from a combination of enhancements to compaction that reduce memory allocations by up to 50%, support for Java 11 with new garbage collectors, and other refinements to memory management. While the results vary with workload, one thing is constant: operators can do more with less hardware.

Using the CMS garbage collector, Cassandra 4.0 (bea 4) on Java 11 reached 51K op/sec while Cassandra 3.11 on JDK 8 maxed out at ~41K ops/sec. (Cassandra 4.0 results are shaded in the table below).

	Throughput Load (ops/sec)				
C*3 and C*4 (shaded), GCs and Java versions	**25k ops/s**	**40k ops/s**	**45k ops/s**	**50k ops/s**	**55k ops/s**
CMS 3.11.6 jdk8	24,222.80	38,519.94	40,722.02	0	0
G1 C* 3.11.6 jdk8	24,222.61	38,736.61	0	0	0
Shenandoah C* 3.11.6 jdk8	24,211.57	35,778.40	0	0	0
ZGC jdk11	24,222.72	38,674.99	41,557.88	0	0
ZGC jdk14	24,222.80	38,519.94	40,722.02	0	0
Shenandoah jdk11	24,222.61	38,736.61	43,649.18	48,438.28	49,689.06
CMS jdk11	24,211.57	35,778.40	43,385.94	47,901.46	51,512.48
G1 jdk11	24,220.08	38,730.57	43,603.88	47,590.90	y0
G1 jdk14	24,211.71	38,721.09	43,561.66	48,030.46	50,833.91

Further testing using Java 16 showed even more increases for some workloads.



Cassandra 4.0 (beta4) has more throughput than Cassandra 3.11, when run on Java 16 combined with CMS and ZGC garbage collectors. Note: Java 16 is not officially supported by Cassandra at the time of this test.

Up to 7 Times Lower Tail Latency

Cassandra 4.0 can reduce tail latency by up to 7 times when compared to Cassandra 3.11. Tail latency is the term used to refer to the latency a very small percentage of requests experience. It is expressed as the response time for the 98th, 99th, or 99.9th percentile, commonly referred to as p98, p99, and p999, and is seen as vitality important when creating compelling "fast data" user experiences. Distributed systems architect and author Roberto Vitillo ***explains***: "Even though a small fraction of requests experience these extreme latencies, it tends to affect your most profitable users. These users tend to be the ones that make the highest number of requests and thus have a higher chance of experiencing tail latencies."

An article in Computer Week goes further, citing statistics that show how high tail latency negatively affects revenue:

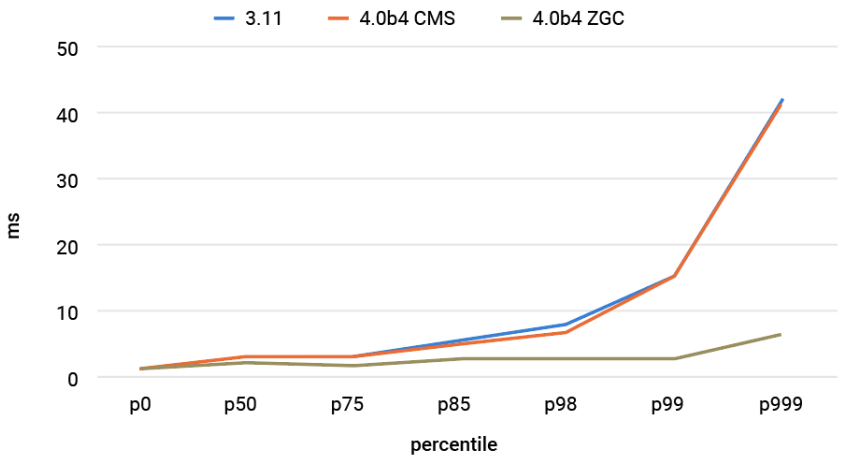
- At Amazon, every 100 milliseconds of latency causes a 1% decrease in sale
- At Bing, a two-second slowdown was found to reduce revenue per user by 4.3%
- At Shopzilla, reducing latency from seven seconds to two seconds increased page views by 25%, and revenue by 7%

Research comparing the latency of Cassandra 3.11.6 and a pre-release version of 4.0 shows that, when running on Java 11 with the ***Shenandoah*** garbage collector Cassandra 4.0 has up to 6 times less latency at the 99th percentile when compared to Cassandra 3.11.6 using Java 8 and the CMS collector.

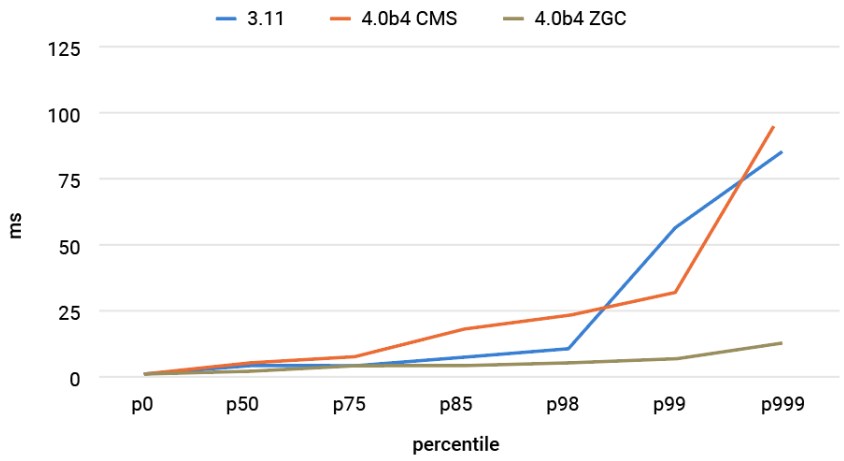
	Read Latency Average p99, Different Loads			
C*3 and C*4 (shaded), GCs and Java versions	**25k ops/s**	**40k ops/s**	**45k ops/s**	**50k ops/s**
CMS jdk8 3.11.6	17.01	24.18	0	0
G1 jdk8 3.11.6	25.21	66.48	0	0
Shenandoah jdk8 3.11.6	9.16	49.64	0	0
ZGC jdk11 4.0~alpha4	4.12	32.70	0	0
ZGC jdk14 4.0~alpha4	2.81	14.83	0	0
Shenandoah jdk11 4.0~alpha4	2.64	9.28	17.37	28.10
CMS jdk11 4.0~alpha4	11.54	20.15	24.87	31.53
G1 jdk11 4.0~alpha4	19.77	26.35	48.38	42.41
G1 jdk14 4.0~alpha4	12.51	22.36	25.20	39.38

Further research testing Cassandra 3.11 and 4.0 with Java 16 and the ***Z Garbage Collector*** (ZGC) shows up to 7 times lower latency.

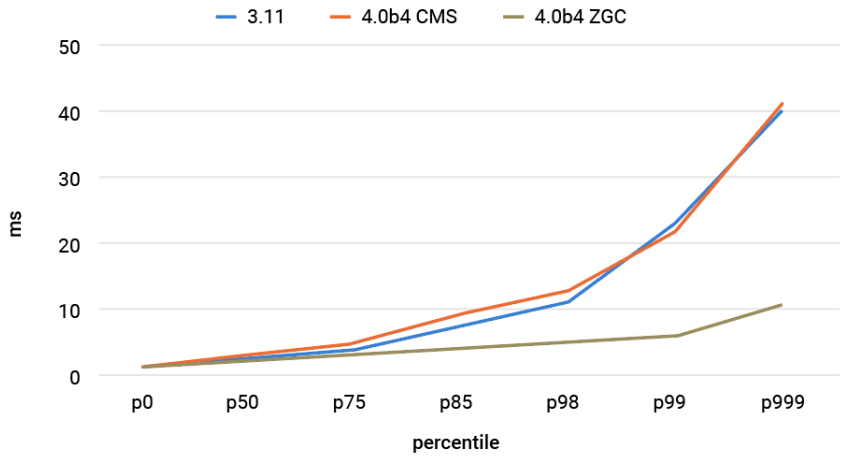
Key/Value workload latencies



Timeseries workload latencies



Tabular workload latencies



Tests done by DataStax indicate that Cassandra 4.0 running on Java 16 with Zero Garbage Collector delivers up to 7x less p98, p99, and p999 latencies than Cassandra 3.11 across different workloads (key-value, time series, and tabular). Note: Java 16 is not officially supported by Cassandra at the time of this test.

Taming tail latencies is a significant breakthrough for all Cassandra users around the world. If you are serious about improving user experience in your customer-facing applications and adding to your top line, it is time to consider Cassandra 4.

Verifiable Peak Performance with Backpressure

In addition to protecting from rogue clients, backpressure messages from a Cassandra cluster allow clients to dynamically, and verifiably, push a cluster to peak performance. Prior to version 4.0 misbehaving clients could have a major impact on the performance of a cluster, and the performance experienced by other clients. It did not matter if the errant behavior was an accident, or a client trying to achieve peak performance. Degrading performance and sending timeouts was the only way to stop errant clients or to let well intentioned code know it had passed peak performance.

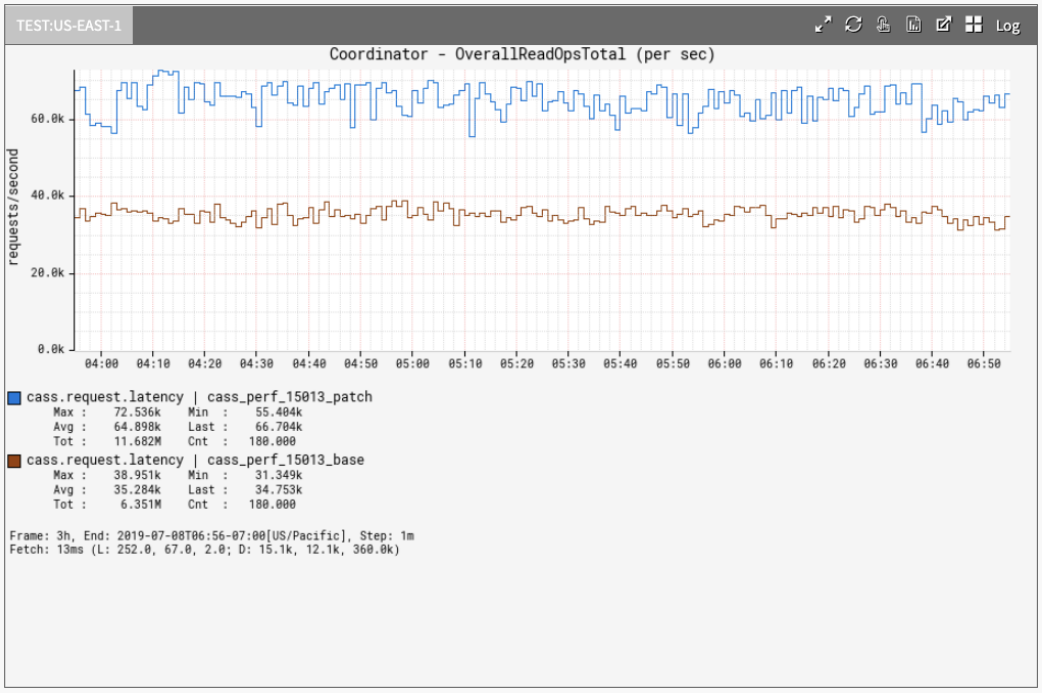
Cassandra 4.0 adds a ***backpressure system*** that will push back to clients when a single client is overloading the system or when the entire node is overloaded. Implementation details are in the linked ASF ***blog post,*** at a high level the node monitors the memory in use to process the commands from a single client (identified by IP address) and the memory used to process all commands. This is done in conjunction with changes to connection handling so that an overload of inbound messages does not block the event loop from sending outbound messages. Rather than limit the number of inbound messages (and blocking when the queue was full), Cassandra limits the amount of memory used to process the messages and applies back pressure when limits are reached.

Backpressure takes two forms: TCP backpressure and OverloadedException response. TCP back pressure is the default and is implemented by no longer automatically reading the bytes from the client. This eventually backs up the TCP buffers on the client and slows it down. Clients can also enable an exception by setting the THROW_ON_OVERLOAD connecting parameter to true. Receiving the OverloadedException allows smart clients to intelligently push the cluster to peak performance.

For workloads that want to achieve peak throughput and get the maximum return for the hardware investment, the steps are:

1. Design your code to ingest source data as fast as possible with the ability to dynamically throttle throughput.
2. Connect to the Cassandra cluster with the THROW_ON_OVERLOAD option enabled.
3. Send data to Cassandra as fast as possible.
4. Handle the OverloadedException exception if it happens, reduce throughput until the errors no longer occur.

In testing the enhancements for the backpressure system resulted in an 80% improvement in read throughput as seen below.

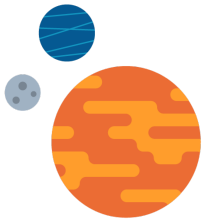


Performance graph showing much better throughput based on a performance analysis.

Source: ***Ticket tracking the improvement (CASSANDRA-15013)***

04

Enhancements for the Operator / Site Reliability Engineer



Greater Stability with More Efficient Compaction

The stability of Cassandra nodes has been greatly improved by reducing the memory allocations during compaction by up to 80%. The ***Log Structured Merge Tree*** storage engine used by Cassandra provides outstanding write performance by running some housekeeping processes asynchronously. The most important of these is compacting together multiple files that may contain deleted, overwritten, or partial data from multiple updates. While often an automatic and often innocuous process for most tables in a database, there have always been edge cases where one or two tables could cause instability for a whole node.

The main cause of compaction related instability was excessive memory allocations leading to increased Java garbage collection resulting in higher latency and reduced throughput. The compaction enhancements (***CASSANDRA-14654***) in Cassandra 4.0 target workloads that deal with a very high number of small partitions. In one benchmark during development the memory allocated during compaction was reduced from 483 MB/second to 262 MB/second with 128 milliseconds (76%) less time spent in garbage collection.

The reduction in garbage collection caused by compaction will improve the stability of nodes as it will reduce the amount of time Java pauses the Cassandra process. This will be particularly true for IoT style workloads that can have a burst of write traffic that must be later compacted.

Reliably Faster Repair with Incremental Repair

Improvements to incremental repair make it a reliable choice for this essential operations task that will dramatically reduce the disk and CPU time spent running repair. Repair is the process Cassandra uses to resolve inconsistency across the entire cluster, it checks all data rather than just data that is being read. Incremental repair was designed to be more intelligent and only repair data that has not been previously repaired, rather than repair all data in a specified token range each time it is run. However it had ***issues*** with failure modes that made it unreliable, Cassandra 4.0 ***resolves those issues*** and allows operators to use incremental repair and further reduce the disk IO and CPU needs of Cassandra.

Repair still needs to be scheduled, executed, and managed through an external service. The ***Reaper*** community project makes running repair a simple task and is now included by DataStax as a component of ***k8ssandra***, an open-source project contributed by DataStax that makes it easy to run Cassandra on Kubernetes.

Better Observability with Virtual Tables

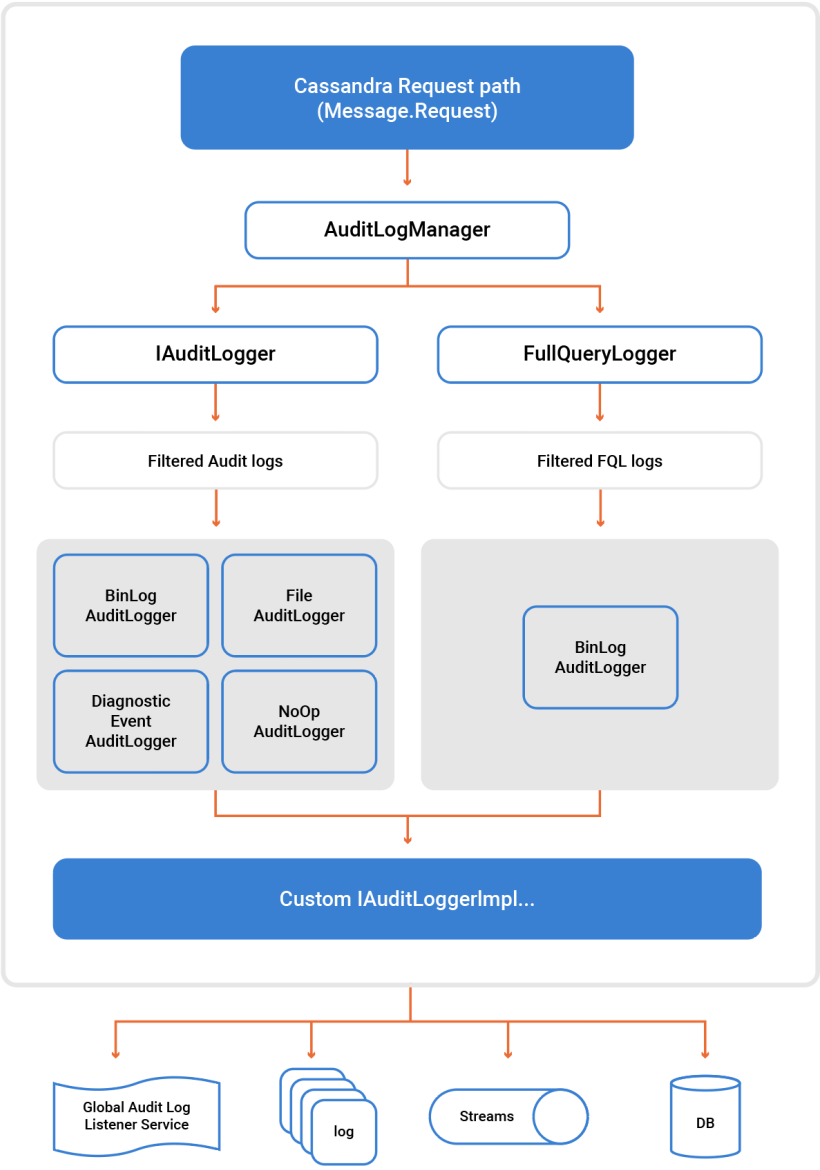
The ***expansion of Virtual Tables*** to include schema, configuration, and metrics creates a single way to interact with and manage Cassandra and deprecates the need for Java JMX access. The expanded virtual tables are similar to "V$" views in Oracle that provide information on system performance. The information provided in Cassandra Virtual Tables was always available through other techniques, such as JMX endpoints or metrics. Making the information available via the CQL interface makes it much easier to access and reduces the need to provide JMX access to Cassandra nodes.

Although the ***Java Management Extensions (JMX)*** endpoints in Cassandra provide a powerful set of management and observation features, they are often difficult or impossible to be approved for use in an Enterprise environment. The heart of the problem is the JMX protocol was not originally designed with security in mind. This difficulty has made it harder for some teams to observe and manage their clusters. With virtual tables and CQL access, an operator can easily look up a schema, execute a select command, and see what's going on in the cluster. This opens up a whole set of new instrumentation opportunities to better manage the cluster.

Workload Replay with Full Query Logging

Workload capture is an essential tool for debugging production problems, complex ***migrations***, and testing new systems. Workload capture involves capturing the commands sent to the cluster at a level of resolution that allows them to be replayed, typically into a different system for testing and comparison. It is no coincidence that Cassandra 4.0 includes Full Query Logging and is the most tested version ever. Entire production workloads were captured and replayed to ensure Cassandra performed flawlessly.

While ***Full Query Logging (FQL)*** reuses some of the same code paths as Audit Logging (see chart below), its configuration, data, and behaviour are different. FQL does not have the granularity to select or exclude users, keyspaces, or command categories; it is designed to collect all the commands on the node. The data captured by FQL does not contain the same information about the client that sent the command; it does contain all the information needed to replay the such as the CQL statement and query options. And lastly, FQL can be configured to drop log records rather than block requests if the logging cannot keep up.

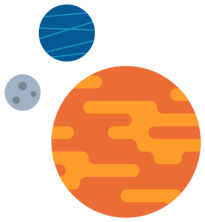


Audit Logging and Full Query Logging (FQL) mechanism

One thing Full Query Logging and Audit Logging have in common: little to no impact on performance. Both rely on the high performance ***OpenHFT Chronicle Queue*** and have been tested during development. Full Query Logging has little to no impact on write only workloads, and a minor overhead on mixed read and write workloads.

05

Next Steps



Cassandra 4.0 is the most performant and tested version of Cassandra ever. Its release will also be the starting pistol for innovation on this stable platform through the *Cassandra Enhancement Proposals (CEP)* program.

For enterprises and developers upgrading will bring TCO reductions and help build more compelling fast data experiences for customers that increase revenue. For any existing Cassandra users, it is time to hop on the Cassandra 4.0 bandwagon.

If you need help with the upgrade process, consider ***DataStax professional services.*** If you need help with ongoing technical support for your Cassandra operators and developers, consider subscribing to ***DataStax Luna,*** brought to you by the experts who have been delivering the majority of the code in Cassandra OSS.

About DataStax

The performance gains and innovations described in this paper are not the only ones happening with Cassandra. DataStax develops the commercial versions of and provides technical support for Cassandra OSS for organizations seeking to modernize their data platform and for developers seeking to develop modern applications.

- ***DataStax Astra*** is the Cassandra as-a-Service designed to simplify cloud-native application development. Perfect for organizations who want to develop modern apps but do not want to operate their own databases. Your developers can use it for free.
- ***DataStax Enterprise*** is the platform for enterprises that need to modernize their data from RDBMS or consolidate their NoSQL database sprawl. Perfect for organizations who prefer to operate their own databases. It can handle almost any workload, in private or public clouds, on Kubernetes.

References

1. ***Webinar given by DataStax in Dec 2020*** by Patrick McFadin, Aaron Morton, and Joshua McKenzie
2. ***Benchmarking tests*** done on Java 11 by The Last Pickle (now part of DataStax)
3. Internal benchmarking test done by DataStax on Java 16
4. ***Why You Should Measure Tail Latency*** by Roberto Vitillo
5. ***How Computer Latency Impacts Customer Facing Applications*** by Henry He, Computer Weekly, Aug 13, 2019
6. ***Manual Repair, DataStax Documentation***
7. ***Hardware bound Zero Copy Streaming in Apache Cassandra 4.0***
8. ***Reduce heap pressure during compactions*** ticket
9. ***Improving Apache Cassandra's Front Door and Backpressure***
10. ***Audit Logging in Apache Cassandra 4.0***
11. ***Best Practices for Migrating from Relational Data Platform to Apache Cassandra***
12. ***DataStax Astra is Managed Cassandra as a Service***
13. ***DataStax Enterprise is Self-Managed Cassandra with enterprise-grade capabilities***
14. ***DBA to SRE with K8ssandra*** blog