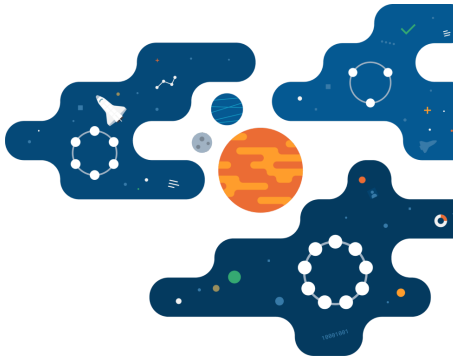
WHITEPAPER



Apache Cassandra 4.0

The Most Reliable, Elastic, and Fastest Version of Apache Cassandra™

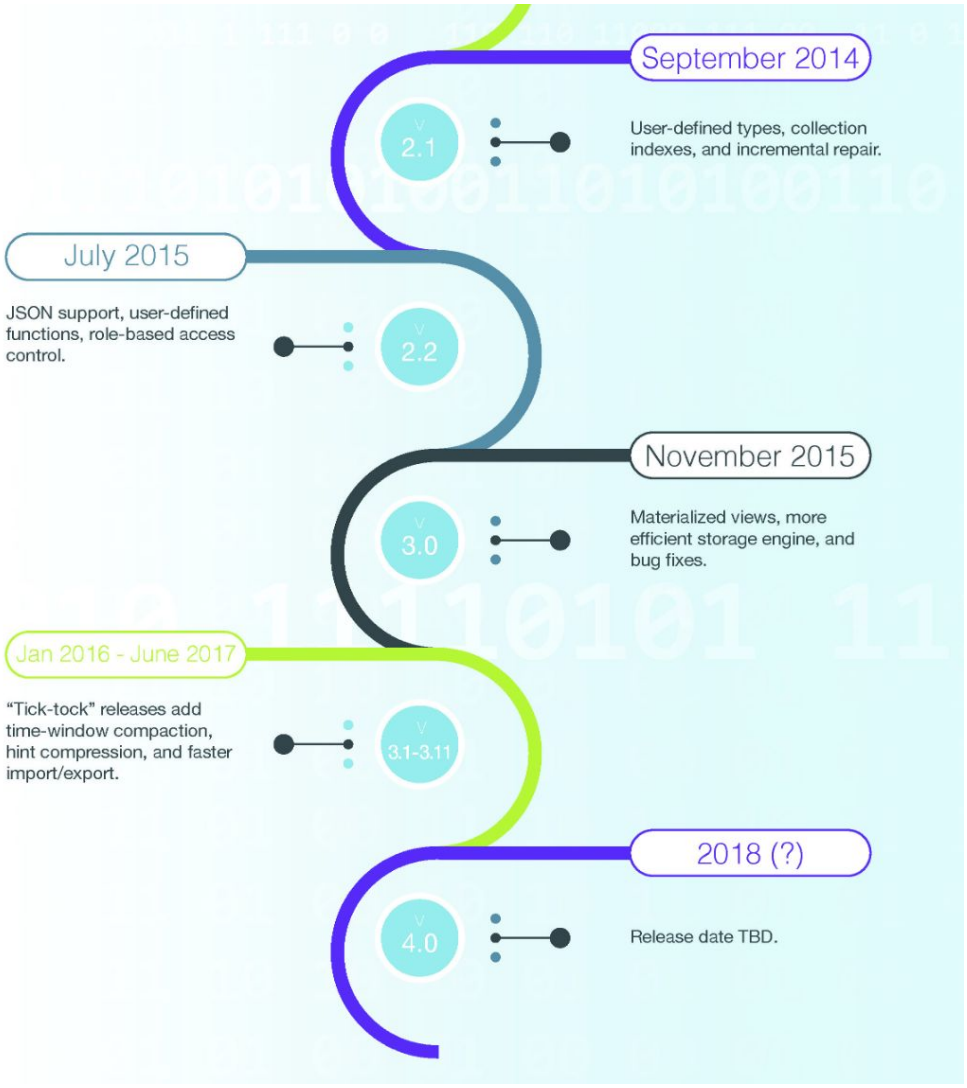
Apache Cassandra 4.0 is the most stable, battle-tested major release of Cassandra ever. It is tested and trusted by Apple, DataStax, Instagram, Netflix, and dozens more.

Before we go deep into Apache Cassandra 4.0, let's take a quick glance at Apache Cassandra, and the journey to Apache Cassandra 4.0.

Apache Cassandra was first developed by Facebook, and became a top-level Apache Foundation project in 2010. Apache Cassandra is a ***free and open-source, distributed, wide column store, NoSQL database*** management system designed to handle large amounts of data across many ***commodity servers,*** providing high availability with no ***single point of failure.*** Apache Cassandra offers robust support for ***clusters*** spanning multiple datacenters or cloud providers with asynchronous masterless replication allowing low latency operations for all clients.

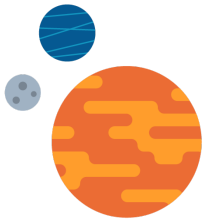
Apache Cassandra was widely adopted because of its tunable consistency, partitioned row store, elastic scalability, high performance, fault tolerant and a promise to provide zero downtime.

Apache Cassandra has been a driving force for applications that scale for over 10 years. This open-source database now powers 30% of the Fortune 100. Apache Cassandra Beta Preview for 4.0 was released on 11/04 and the GA is around the corner.


September 2014
2.1
User-defined types, collection indexes, and incremental repair.
July 2015
JSON support, user-defined functions, role-based access control.
2.2
November 2015
3.0
Materialized views, more efficient storage engine, and bug fixes.
Jan 2016 - June 2017
"Tick-tock" releases add time-window compaction, hint compression, and faster import/export.
3.1-3.11
2018 (?)
4.0
Release date TBD.

01

Apache Cassandra 4.0 Benefits



Note, this document only describes advantages of Cassandra 4.0 features over the previous major version of Cassandra 3.11. The document does not make comparisons between 4.0 and versions earlier than 3.11. The documentation provides insight into Cassandra 4.0 cost optimization, new feature benefits like; Auditing, elasticity, and throughput. The document also provides insight into aspects of Security and CDC (Change Data Capture).

Apache Cassandra is the fastest, most elastic, and most available database in the world, and with Cassandra 4.0 it is getting better. Apache Cassandra 3.11.6 maxed out at 41k ops/s while Apache Cassandra 4.0 running on the same hardware goes up to 51k ops/s, which is a nice 25% improvement.

It is important to call out that OSS partnership with DataStax (including 4.0 and older versions of Apache Cassandra) resulted in 20% infra cost savings. More on this in the later sections of this paper.

With Apache Cassandra 4.0 you get:

- Faster, Denser nodes
- Backpressure and improved resiliency in the system
- Realtime audit logging and workload capture and replay
- The most stable, battle-tested major release of Cassandra Ever (Tested by Apple, DataStax, Instagram, Netflix, and dozens more)

Cassandra 4.0 provides improvements everywhere:

Better ROI: Performance and density improvements

- Significant compute improvements
- Support for zstd compression

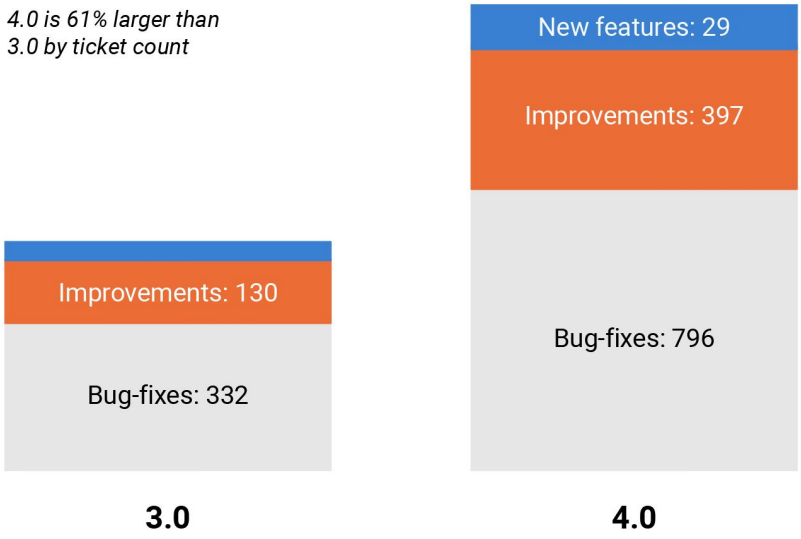
Operator improvements

- Compaction: performance improvements (CPU and memory)
- Networking: major improvements in performance and client backpressure
- New feature: Realtime audit logging
- New feature: Workload capture and replay (QA, root cause analysis)
- Incremental repair hardening

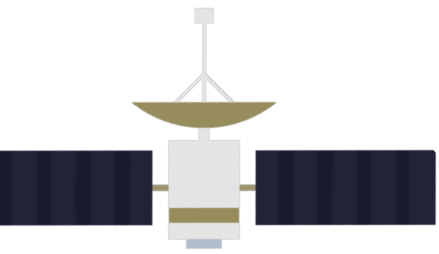
Bootstrapping and Recovery improvements: 5x faster with zero-copy streaming

Cassandra 3.0 vs. 4.0

4.0 is 61% larger than 3.0 by ticket count

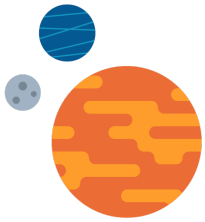


Read ***TLP Blog*** for more information on performance testing.



02

Release Themes and Objectives—Apache Cassandra 4.0



The Apache Cassandra community has been working tirelessly on the quality of the core database to build an indestructible foundation that you can trust to handle your most demanding data challenges.

The release of the Cassandra 4.0 beta marks a major achievement for the project that has poured over 1000 bug fixes, improvements, and new features into this release; see the community's ***blog post*** for details.

Join the community on the path to GA by running your workloads against the 4.0 beta in your test environments today and kick the tires in a few clicks on your laptop by downloading ***DataStax Desktop.***

Why is the Cassandra 4.0 release such an important milestone?

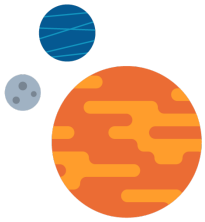
Cloud computing and Kubernetes have transformed the way that we build, deploy, and run stateless applications. This new breed of application scales as traffic fluctuates, is resilient to a range of failure scenarios, and is containerized, allowing it to predictably run across different environments. Stacks that are fully cloud-native have these characteristics at every layer; from the application, to the database, to the infrastructure that it runs on. Cassandra fits perfectly as the database of choice in this cloud-native stack because of its peer-to-peer architecture, extreme elasticity, and incredible resilience.

That said, there is groundwork to be done, both in Cassandra and Kubernetes to serve these cloud-native applications with a data tier that matches the same attributes as the stateless components. Cassandra 4.0 contains many pieces of this foundational work to hammer out rare edge cases of unavailability during topology changes and to scale elastically when adding capacity to the cluster.

Also, with Cassandra 4.0 you can now anticipate infra cost savings from Open Source DataStax Partnership close to 20% cost reduction, and there are even further benefits in a multi-cloud setup.

03

Apache Cassandra 4.0 New Features



Each release of Cassandra adds improvements, enhancements, fixes, and new features. This section of the document runs through the major features and whether their use is recommended.

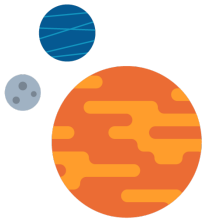
Beyond this information, the NEWS.txt file in each release lists major features, operational and functional changes, and any tasks that must be carried out during the upgrade. This includes a Maximum TTL expiration notice that should be read prior to doing this upgrade. The CHANGES.txt file in each release goes into further detail, listing the tickets that are included in each release.

This document will explore in detail the following new and improved major features:

- Auditing
- Full Query Logging
- Data Center Authorization
- SSL Certificate Reloading
- Zero Copy Streams
- Java 11
- Transient Replication
- Incremental Repairs Improvements
- Repair Streaming Preview and Validation Options
- Change Data Capture Improvements

04

Auditing



Background

Auditing is a long awaited feature in Apache Cassandra. From operational policies to regulatory compliance, database audit logging is an industry standard for security in the enterprise.

Regulatory compliance with laws such as SOX, PCI and GDPR et al. are critical for many companies, for example those that are traded on public stock exchanges, hold payment information such as credit cards, or retain private user information. Most enterprises today are expected to have sound security practices in place, with strict rules for what data can be accessed by which employees, to protect the privacy of users and limit the probability of a data breach.

Implemented in ***CASSANDRA-12151***

Key Concepts

Use Auditing for security, business accounting, monetary transaction, and privacy regulation, compliance. Use Full Query Logging (FQL) for correctness testing, and Diagnostics Events for debugging. Use Change Data Capture (CDC) for runtime duplication/exporting/streaming of the data.

Auditing and Full Query Logging have been implemented on a shared design. This uses the OpenHFT libraries. By default the chronicle queue library of OpenHFT is used as a BinLog, making the default behaviour file-based.

Performance

The implementation is file (binlog) based. Different OpenHFT plug-in libraries can be used, the choice of which will influence operability and performance.

There are performance implications involved when enabling Auditing. This depends on the auditing setup and configuration. Testing and benchmarking therefore is required to evaluate production implications.

Bugs

There have been reported ~10 bugs against the new Auditing feature, even before it has become Generally Available (GA) to the public. This gives some indication to a medium level of complexity, and the amount of existing code touched, to implement the feature. Nearly all of these reported bugs have been fixed.

Final Thoughts

Auditing is designed to meet regulatory and security requirements, and is able to audit more types of requests and interactions with the database than Full Query Logging (FQL) or Change Data Capture (CDC) can.

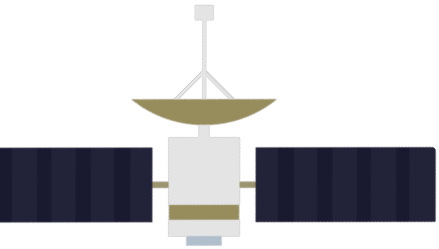
By default we do not recommend using new features released in their first major version. It should be regarded as an experimental feature, and use of the feature should be accompanied with involvement and resources, internally or externally, to ensure a required degree of ownership to the open sourced code behind that feature.

Because the use of Auditing may be a hard requirement for many, our recommendation is to use it with thorough testing and benchmarking prior to deployment in production.

Auditing will still have a performance impact on production clusters, and file-based logging needs to be configured to be operationally safe and sustainable. Ensure appropriate metrics monitoring and alerts are in place around the log files.

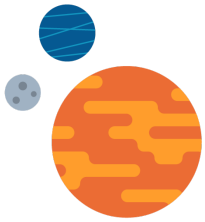
For more detailed information on Auditing read this blog post
https://cassandra.apache.org/blog/2018/10/29/audit_logging_cassandra.html

Recommendation: When Auditing is a requirement, be prepared to thoroughly test and benchmark it before any production deployments. Otherwise we recommend waiting until the next release. Reach out for support if you would like to use this feature.



05

Full Query Logging (FQL)



Background

Full Query Logging (FQL) logs all requests to the CQL interface. FQL can be used for debugging, performance benchmarking, testing and also for auditing CQL queries. In comparison to audit logging, FQL is dedicated to CQL requests only and comes with features such as the FQL Replay and FQL Compare that are unavailable in audit logging.

Key Concepts

Full Query Logging's primary use is for correctness testing. Use Auditing for security, business accounting, and monetary transaction compliance. Use Diagnostics Events for debugging. Use Change Data Capture (CDC) for runtime duplication/exporting/streaming of the data. FQL only logs the queries that successfully complete.

Performance

Existing benchmarking has shown that FQL appears to have little or no overhead in WRITE only workloads, and a minor overhead in MIXED workload.

The implementation is file (binlog) based. Different OpenHFT plug-in libraries can be used, which will influence operability and performance.

Minor performance changes are to be expected, and will depend on the setup and configuration. Testing and benchmarking therefore is required to evaluate production implications.

Bugs

There have been reported ~10 bugs against the new Auditing feature, even before it has become GA to the public. This gives some indication to a medium level of complexity, and the amount of existing code touched, to implement the feature. Nearly all of these reported bugs have been fixed.

Final Thoughts

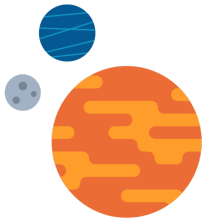
By default we do not recommend using new features released in their first major version. For FQL this recommendation would only apply to any use of FQL at the application runtime. Since FQL is intended for correctness testing its use will add quality assurance to existing clusters, something we strongly recommend.

FQL will still have a performance impact on production clusters, and file-based logging needs to be configured to be operational safe and sustainable. Ensure appropriate metrics monitoring and alerts are in place around the log files.

Recommendation: We recommend Full Query Logging as an important addition to testing Cassandra platforms and data. It is recommended to benchmark the performance impact and operational overhead of using it before deploying to production. Reach out for support if you would like to use this feature.

06

Authorization by Data Center



Background

Authorization on a per Data Center basis has been implemented in Cassandra 4.0.

Previously authorization, via user or role, was all or nothing with no concept of data centers. With version 4.0 and via the use of roles it will be possible to permit and restrict access to specific data centers.

Key Concepts

Allow Cassandra operators to enforce connections from client applications to data centers. This is done using the existing Authorization feature in Cassandra, and its concept of Roles. This only prevents the connections to coordinators in disallowed data centers. Data from remote data centers can still be accessed through a local coordinator. Server-side control of connections to coordinators is important to prevent saturation of network and available connections. This can lead to client applications being unable to connect to the cluster. This is a common problem with analytics data centers where clients, like Apache Spark, can spawn hundreds or thousands of connections.

Final Thoughts

This is a safe feature to use immediately, as it touches very few components of the codebase. Where there is benefit in using this feature, we recommend its immediate use.

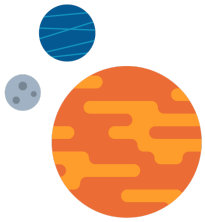
Do not rely upon this feature to restrict access to data only replicated to certain data centers.

For more information read our blog post at *https://thelastpickle.com/blog/2018/05/08/cassandra-4.0-datacentre-security-improvements.html*

Recommendation: The use of data center authorization is recommended when this is a need to control and limit connections to coordinators in specific data centers.

07

SSL Certificate Reloading



Background

From the docs:

Beginning with Cassandra 4, Cassandra supports hot reloading of SSL Certificates. If SSL/TLS support is enabled in Cassandra, the node periodically polls the Trust and Key Stores specified in cassandra.yaml. When the files are updated, Cassandra will reload them and use them for subsequent connections. Please note that the Trust & Key Store passwords are part of the yaml so the updated files should also use the same passwords. The default polling interval is 10 minutes.

Certificate Hot reloading may also be triggered using the nodetool reloadssl command. Use this if you want Cassandra to immediately notice the changed certificates.

Key Concepts

SSL certificates have expiring dates forcing Cassandra nodes prior to version 4.0 to have to be restarted to reload replacement certificates. Cassandra version 4.0 removes the need for restarting nodes, automatically checking for new SSL certificates every ten minutes.

SSL Certificate Reloading is enabled by default. The feature was implemented in CASSANDRA-14222.

Final Thoughts

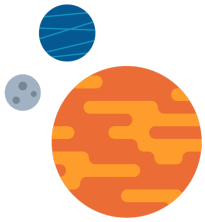
Many environments with good security hygiene inplace, for example certificate and keystore rotation automated, will want to shorten the expiry period on SSL certificates. Automatic reloading of SSL certificates by Cassandra nodes removes a manual step from that operational process.

This is a valuable advantage for operations and should, and can safely, be taken advantage of. If more complicated security operations are required, for example Automatic certificate management using Vault, reach out to DataStax.

Recommendation: It is recommended to use SSL Certificate Reloading. It is enabled by default.

08

Zero Copy Streams



Background

In Cassandra 4.0 hardware-bound Zero Copy Streaming (ZCS) is introduced. Streaming is a fundamental operation to many of Cassandra's internal operations: bootstrapping, decommissioning, host replacement, repairs, rebuilds, etc. Prior to this feature, all data was streamed through the JVMs of the source and destination nodes, making it a slow and GC intensive process. With hardware-bound streaming, data can be transferred directly from disk to network avoiding the JVM.

Key Concepts

Enabled via the cassandra.yaml setting stream_entire_sstables: true Throttling still occurs according to the stream_throughput_outbound_megabits_per_sec setting.

ZCS will only be used when an SSTable contains only partitions that have been requested to be streamed. In practice this means it only works when vnodes are disabled (i.e. num_tokens=1), or RangeAwareCompaction (not yet implemented). That is, if an SSTable candidate contains partitions that are additional to those of the requested streaming operation, it will be streamed via the traditional JVM streaming implementation.

ZCS is also unable to be used for any SSTables that contain legacy counters (from Cassandra versions before 2.1).

To summarize, for ZCS to work on a particular SSTable, the following checks must pass:

1. The stream_entire_sstables setting is true.
2. No counter legacy shards in SSTable.
3. The SSTable must contain data belonging to only the token ranges specified in the streaming request.

Performance

Activation and frequency of ZCS can be observed in the debug.log file, if it is enabled. The output will be from the CassandraEntireSSTableStreamReader logger.

Bugs

ZCS has yet to be deployed to any production environments, as it is unreleased. It involves new code paths inside Cassandra that will get little production verification, as many clusters use vnodes. A number of shortcomings to the feature have already been addressed, even before its initial release. A number of bugs are still being worked on leading up to the first 4.0 beta version. ZCS should be seen as an experiment feature in the 4.0 release.

Final Thoughts

ZCS provides improved availability and elasticity by reducing the time it takes to stream token range replicas between nodes during bootstrapping, decommissioning, and host replacement operations. ZCS provides improved performance by reducing the load and latency impact streaming large amounts of data through each JVM causes. Benchmarks show that ZCS provides five times faster streaming. But ZCS only works for clusters not using vnodes, i.e. where num_tokens=1.

The big improvements from ZCS will be more widely available when RangeAwareCompaction is finished implementation (RAC). With RAC the time to bootstrap nodes will be one fifth of the time, with an unthrottled streaming rate.

An alternative implementation of ZCS is found in DSE 6.8, and is being proposed to the open sourced Cassandra, see CEP. This implementation does zero-copy streaming for all streaming, regardless of setup or what is requested in the stream. In benchmarks a 1TB node can be bootstrapped in 30 minutes, while in practice on a running cluster with latency requirements the streaming may be reduced to being a factor four to five times faster than current streaming rates. This means bootstrap times would be one-quarter to one-fifth of their current times.

Conclusion: Bootstrapping 1TB nodes in under one hour is possible, but is waiting on either RangeAwareCompaction or the DSE ZCS implementations. Reach out to DataStax for heap or more information.

Other Streaming Improvements in 4.0: There are a number of other important streaming improvements in version 4.0: Netty non-blocking NIO messaging, parallel keyspace streams, better stream balancing during bootstrap. These improvements will improve bootstrapping times and overall elasticity of a Cassandra cluster.

To enable parallel streams, increase the value in cassandra.yaml for the streaming_connections_per_host setting.

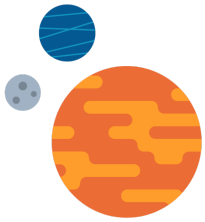
Recommendation: ZCS provides no benefits on vnode clusters. There is no negative impact, so it is recommended to leave enabled.

Recommendation: Increase streaming_connections_per_host when streams are receiver CPU bound.

Noted for Reference: Expect faster streaming and bootstrapping times in 4.0 anyway.

09

Java 11



Background

Apache Cassandra 4.0 is the first version that will support JDK 11 and onwards. Beyond the need for running on supported versions of Java, an obvious cost concern for Apache Cassandra users is latency and throughput performance. Making available the use of JDK 11 and JDK 14 introduces the new low latency garbage collectors available: Shenandoah and ZGC.

Cassandra version 4.0 brings impressive boosts on its own, but by allowing the use of the new generation garbage collectors further reductions in latencies and increases in throughput can be expected. Developers and Operators time is also saved, as in the same tradition of G1, both Shenandoah and ZGC are much easier to tune than CMS.

Key Concepts

Java 8, for personal use, is supported up to at least December 2020. The last commercial update for Java 8 was in January 2019.

Java 14 General Availability occurred on March 17, 2020.

Performance

Comparisons between Cassandra 3.11 and 4.0 show potential improvements of 35% in throughput and 85% reductions in p99 latencies.

The G1GC brought improvements in Cassandra's usability by removing the need to fine tune generation sizes at the expense of some performance. The new GCs: ZGC and Shenandoah; provide the same ease of use but also bring significant performance improvements over CMS and G1.

Bugs

With Cassandra, Java 11 and above has not been used and tested in production systems. With the GA release of Cassandra version 4.0 live testing of production clusters with Java 11 will begin in earnest.

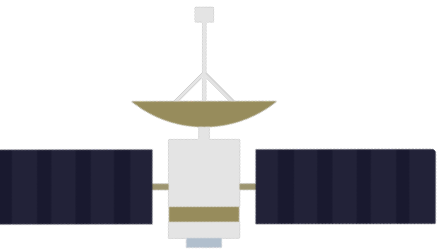
Running Cassandra with Java 11 on Windows is unsupported. See CASSANDRA-14608 for further details. We recommend avoiding running Cassandra on Windows in production. Otherwise there are no known unresolved issues.

Final Thoughts

On smaller heaps of 16G or less, we recommend using CMS and putting some effort into tuning it. On heaps sizes from 20G and upwards our preliminary recommendations are to use G1 on JDK8, and Shenandoah on JDK11. On JDK14 both Shenandoah and ZGC outperformed G1. When using Shenandoah or ZGC we recommend some effort into tuning before considering it safe in production. Both the Shenandoah and ZGC collectors are quicker to tune than CMS.

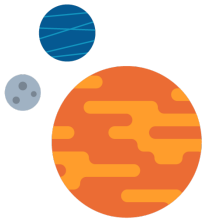
Because Cassandra version 4.0 brings about significant performance improvements on its own, we recommend that trying new GC collectors and tuning be postponed until a period of time after the upgrade to 4.0 has occurred. This is to ensure a reliable performance baseline is being compared against when comparing and tuning GC collectors.

Recommendation: It is recommended to switch from JDK 8 to JDK 11. Changing GC collectors, and using JDK 14 can be done, but later on and piecewise, only after incremental testing and observations.



10

Incremental Repairs



Background

Incremental Repairs were introduced in Cassandra 2.1 and made default in Cassandra 3.0. The concept to Incremental Repairs is to keep the data on disk in separate sets of SSTables, one set for repaired data and another for unrepaired. Based on the idea that once some data has been repaired, it can be marked as such and never needed to be repaired again. This then allows repairs to focus on the smaller set of unrepaired data, at the expense of compactions divided into the two sets (and anti-compaction).

Despite having been made the default in Cassandra 2.2, Incremental Repairs remain with serious flaws and bugs. We recommend to avoid using incremental before Cassandra 4.0. An example of the flaws with this feature is CASSANDRA-9143 which leads to large amounts of over-streaming during repairs. Another example is CASSANDRA-14763.

Also note that Incremental Repairs do not work yet with Materialized Views (MV) or Change Data Capture (CDC). See CASSANDRA-12888.

Key Concepts

Entropy of consistency of replicas at rest is an unavoidable aspect of running Cassandra clusters. Running normal repairs has been a required additional operation on clusters. Incremental repairs help reduce the load repairs generate, permitting more frequent repair runs.

Performance

The repair process: calculation of merkle trees and streaming on unrepaired data; is minimised with the use of incremental repairs, at the expense of some additional compactions. Merkle trees are only generated over unrepaired data, and finer granularity of unrepaired data can be streamed.

Incremental repairs are significantly faster than normal repairs, allowing them to be run at higher frequency. An important consequence of this is it allows gc_grace_seconds to be reduced from its default of ten days to a value on par with the incremental repair frequency. For tables with tombstones this can have a significant benefit on storage costs.

The use of incremental repairs is also an important predecessor for the use of Transient Replicas.

Bugs

Incremental Repairs are yet to work with Materialized Views (MV) or Change Data Capture (CDC). See CASSANDRA-12888.

If incremental repair was being used prior to upgrading to 4.0, it is important that a full repair is run after the upgrade, before any further incremental repairs. This is to resolve any inconsistencies possible from a number of earlier bugs now fixed. To protect against disk rot, corruption, and operator error, it is still recommended to run full repairs occasionally. That exact frequency of full repairs required depends on the data model and use, but once a month is the starting suggestion. The frequency of incremental repairs also depends on the data model, but daily is a safe starting suggestion.

Final Thoughts

Incremental Repairs will provide performance improvements, and can reduce disk usage by allowing lower gc_grace_seconds values. It also permits a higher frequency of running repairs which, with less inconsistency of data on disk, improves the results of eventual consistency.

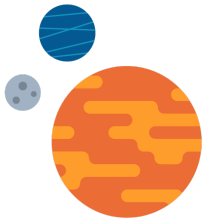
As we recommend to avoid using either MVs or CDC, we do therefore recommend switching to use incremental repairs.

For more information read our blog post on using Incremental Repairs in Cassandra 4.0 at ***https://thelastpickle.com/blog/2018/09/10/incremental-repair-improvements-in-cassandra-4.html***

Recommendation: It is recommended to switch to use Incremental Repair (on clusters not using MVs or CDC). The use of incremental repairs should be accompanied with the use of full repairs.

11

Streaming Previews



Background

Two new options have been added to the `nodetool repair` command: preview and validate. The stabilisation of incremental repairs means the recommendation is now to use both incremental and full repairs. This presents some new scenarios and challenges for operators and these two new options help to address them.

The `--preview` option estimates the amount of streaming that would occur for the given repair command. This builds and compares the merkle trees, and prints the expected streaming activity, but does not actually do any streaming. Note, building merkle trees is still an expensive component to the repair process. Applies to both incremental and full repairs.

The `--validate` option verifies that the repaired data is the same across all nodes. Similar to the `--preview` option, this builds and compares merkle trees of repaired data and doesn't do any streaming. This is useful for troubleshooting. This option will highlight that the repaired data is out of sync, indicating that a full repair should be run. Applies to incremental repairs.

Implemented in CASSANDRA-13257.

Performance

Both the `--preview` and `--validate` options involve having to calculate merkle trees on all replica nodes. Using the `--preview` option implies, by default, incremental repair so merkle trees calculated are limited to only the unrepaired data sets. Using `--preview --full` involves calculating merkle trees on all data of all replicas. Using `--validate` involves calculating merkle trees on the repaired data set on all replicas. All these approaches impact a cluster's performance and latency numbers, but used correctly can be used to optimize performance by avoiding unnecessary streaming and compactions.

Bugs

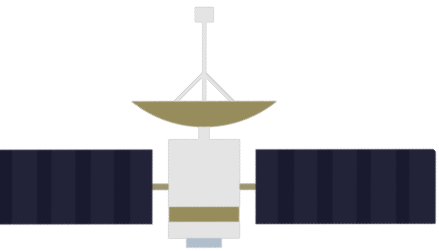
There has been a handful of small bugs and improvements to both these new options to nodetool repair. While they are intended for troubleshooting use-cases, this indicates that they may evolve a little more during the 4.0 releases.

Final Thoughts

Preview the expected streaming required for a repair with `nodetool repair --preview (--full)`. Use this to address concerns around the amount of streaming that will occur. Streaming that takes too long has a number of potential negative effects. Such negative effects include reducing the availability of a cluster, unacceptably increasing the latency of requests, causing the repair to fail, and causing the repair to take so long it has no value. Where the cost of compactions: specifically validation compactions, i.e. calculating merkle trees, a read-only IO operation; is relatively smaller than the cost of streaming and normal compactions, getting a preview of the streaming helps operators understand the subsequent streaming and normal compaction costs.

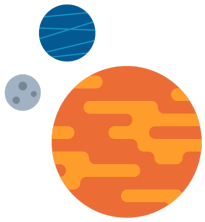
Validate the consistency of repaired data between nodes with nodetool repair --validate. Use this to optimise full repairs when the cost of compactions: specifically validation compactions, i.e. calculating merkle trees, a read-only IO operation; is relatively smaller than the cost of streaming and normal compactions. That is, full repairs can be avoided by first validating that all repaired data is in sync. The trade-off is when a full repair is required the merkle tree calculations end up being done twice. The full repair can not optimize this situation itself as it will also repair the unrepaired datasets, which the validate option does not include. When a full repair can be avoided then the incremental repair can be used to just repair those unrepaired datasets.

Recommendation: Both the preview and validate options to nodetool repair should be considered only for manual troubleshooting use. A well defined repair schedule of incremental and full repairs should take precedence, as well as a cluster properly capacity planned.



12

Transient Replication



Background

Transient Replication takes advantage of incremental repairs, allowing consistency levels and storage requirements to be reduced when data is marked as repaired.

Transient Replication introduces new terminology for each replica, replicas are now either full replicas or transient replicas. Full replicas, like replicas previously, store all data. Transient replicas store only unrepaired data.

Key Concepts

Explicitly announced as an experimental feature.

Currently cannot be used for:

- Monotonic Reads
- Lightweight Transactions (LWTs)
- Logged Batches
- Counters
- Keyspaces using materialized views
- Secondary indexes (2i)
- Tables with CDC enabled

Introduces operational overhead when it comes to changing replica factor (RF).

- RF cannot be altered while some endpoints are not in a normal state (no range movements).
- You can't add full replicas if there are any transient replicas. You must first remove all transient replicas, then change the # of full replicas, then add back the transient replicas.
- You can only safely increase the number of transients one at a time with incremental repair run in between each time.

Final Thoughts

Transient Replication is released in version 4.0 as an experiment feature. By default, we do not recommend using new features released in their first major version. Any use of such features should be accompanied with committed involvement and resources, internally or externally, to ensure the required degree of ownership to the open sourced code and the feature.

Eventual use of Transient Replication in subsequent versions is an exciting promise of reducing network traffic and storage requirements: node volume sizes and/or number of nodes; by up to 66%. More information can be found here.

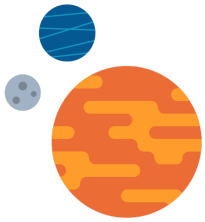
Some example configurations could be

- 3/2 where durability and availability are not required by Cassandra (i.e. you are happy to rely on backups, and use a CP model), providing up to 66% storage saving,
- 3/1 where quorum and availability together is not required (i.e. for eventually consistent AP models), providing up to 33% storage saving,
- 5/2 for quorum with increased availability and performance, without any storage savings.

Recommendation: We do not recommend the use of Transient Replication yet. Reach out for support if you would like to use this feature.

13

Change Data Capture



Background

From the docs:

Change data capture (CDC) provides a mechanism to flag specific tables for archival as well as rejecting writes to those tables once a configurable size-on-disk for the combined flushed and unflushed CDC-log is reached. An operator can enable CDC on a table by setting the table property cdc=true (either when creating the table or altering it), after which any CommitLogSegments containing data for a CDC-enabled table are moved to the directory specified in cassandra.yaml on segment discard. A threshold of total disk space allowed is specified in the yaml at which time newly allocated CommitLogSegments will not allow CDC data until a consumer parses and removes data from the destination archival directory.

Change Data Capture (CDC) was first introduced in Cassandra 3.8, implemented using separate commit log files. In Cassandra version 4.0 some improvements have been made based on feedback and production experiences.

indexes on the commit logs have been added for improved performance and reliability. The feature can be globally disabled in the cassandra.yaml configuration file, and enabled per-table in the CQL schema.

Key Concepts

Use Change Data Capture (CDC) for runtime duplication/exporting/streaming of data.
Use Auditing for security, business accounting, and monetary transaction compliance.
Use Full Query Logging (FQL) for correctness testing, and Diagnostics Events for debugging.

CDC requires there to be consumers off all nodes to guarantee all changes are captured. This means de-duplication efforts are on the CDC consumers.

Performance

The speed and durability of CDC in Cassandra 4.0 has improved due to the index files on the commit logs. Being able to enable CDC only on specifically needed tables also improves durability and load on the cluster.

When using CDC it is crucial to configure cdc_total_space_in_mb and cdc_free_space_check_interval_ms so disks don't fill and crash nodes, for example when the consumer stops working or can not keep up.

Cassandra 4.0 and - CDC does increase the system's IO, the commit logs must be written to and then read from by the consumer. Testing and monitoring of production deployments must be put in place. There exists performance overhead and availability risks in using CDC.

Bugs

The current CDC implementation has several shortcomings. Multiple records of the same write operation are created (one per replica) with no consistency guarantees, so deduplication is required on the client side. Various failure handling is not supported. And the implementation is file (commitlog) based.

CDC also does not yet work with Incremental Repairs. Incremental Repairs must not be run on MV or CDC enabled tables, as repaired data goes through the mutation path resulting in it then landing in the unrepaired sstables on disk. Care must be taken to avoid ever running incremental repairs on any tables with CDC enabled.

Final Thoughts

CDC is not a popular feature, though there are users and no significant issues raised. It does include some variation in the write code path that doesn't get tested as extensively. It also does not yet work with Incremental Repairs.

For a more advanced CDC implementation there is a new version being proposed. To follow its development observe its Cassandra Enhancement Proposal page at ***https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=95652201#CassandraEnhancementPropos als(CEP)-CEPsindraft***

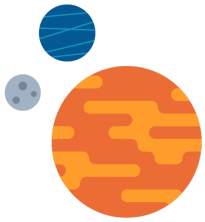
We generally do not recommend the use of CDC. System designs that have existing streaming infrastructure (e.g. Apache Kafka) should use that instead. For situations where that is not possible, CDC is then a recommended solution. There exists performance overhead and availability risks in using CDC. Testing and monitoring of production deployments must be put in place.

Recommendation: When CDC is a requirement, be prepared to thoroughly test and benchmark it before any production deployments. Reach out for support if you would like to use this feature.

Recommendation: Never run Incremental Repairs on CDC-enabled tables. Ensure repairedAt timestamps in each SSTable's metadata is always 0.

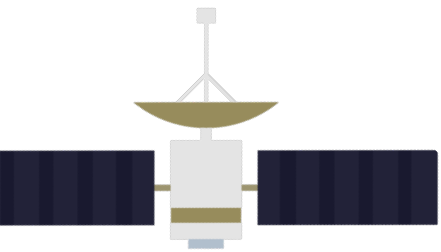
14

Application of Recommended New Features



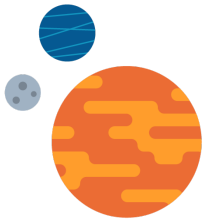
The following section runs through the application of each new feature and our recommendation of it.

- Applying Authorization by Data Center
- Enforcing a 90-day Password Rotation
- Using Java 11
- Switching to Incremental Repairs
- New cassandra.yaml configurations and schema options



15

Applying Authorization by Data Center



Update cassandra.yaml to have the settings:

```
authenticator: PasswordAuthenticator

network_authCassandraNetworkAuthorizer
```

A simple example of validating this, using the ccm tool, follows.

1. Create the ccm cluster, using 4.0-alpha4 and two data centers. Test normal cqlsh works.

   ```
   $ ccm create test_dc_auth -v 4.0-alpha4 -n 1:1 $ ccm status $ ccm start $ ccm status $ ccm node1 nodetool status $ ccm node1 cqlsh cqlsh> describe keyspaces; cqlsh> exit;
   ```

2. Switch to use the PasswordAuthenticator.

   ```
   $ sed -i '' 's/authenticator: AllowAllAuthenticator/authenticator: PasswordAuthenticator/' ~/.ccm/test_dc_auth/node1/conf/cassandra.yaml $ sed -i '' 's/authenticator: AllowAllAuthenticator/authenticator: PasswordAuthenticator/' ~/.ccm/test_dc_auth/node2/conf/cassandra.yaml
   ```

3. Perform a rolling restart of the (two) nodes. If you like you can validate the authenticator working (or not) at each step through this process. This demonstrates the need for all clients to be providing the authentication credentials before this rolling restart initiates.

   ```
   # at this point you could test cqlsh still works without requiring authentication $ ccm node1 stop $ ccm node1 start # at this point you could test cqlsh requires authentication on node1 but not node2 $ ccm node2 stop $ ccm node2 start # at this point you could test cqlsh requires authentication on all nodes
   ```

4. Now introduce the CassandraNetworkAuthorizer.

   ```
   $ sed -i '' 's/network_authorizer: AllowAllNetworkAuthorizer/network_authorizer: CassandraNetworkAuthorizer/' ~/.ccm/test_dc_auth/node1/conf/cassandra.yaml $ sed -i '' 's/network_authorizer: AllowAllNetworkAuthorizer/network_authorizer: CassandraNetworkAuthorizer/' ~/.ccm/test_dc_auth/node2/conf/cassandra.yaml
   ```

5. Perform another rolling restart.

```
$ ccm node1 stop $
ccm node1 start $
ccm node2 stop $
ccm node2 start
```

6. Ensure that the system_auth keyspace is replicated to all nodes. This should have been the case already. If it was not, then it is important to perform a repair on all nodes.

```
$ ccm node1 cqlsh -u cassandra -p cassandra # you always want system_auth
replicated to every node cassandra@cqlsh> ALTER KEYSPACE system_auth WITH
REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 1, 'dc2' : 1};
cassandra@cqlsh> exit

ccm node1 nodetool repair system_auth
ccm node2 nodetool repair system_auth
```

7. Create two roles: foo and bar; that each have access to different datacenters. Also create a keyspace and table, with some data, that only is replicated in one datacenter.

```
$ ccm node1 cqlsh -u cassandra -p cassandra cassandra@cqlsh> CREATE ROLE foo
WITH PASSWORD = 'foo' AND LOGIN = true AND ACCESS TO DATACENTERS {'dc1'};
cassandra@cqlsh> CREATE ROLE bar WITH PASSWORD = 'bar' AND LOGIN = true AND
ACCESS TO DATACENTERS {'dc2'}; cassandra@cqlsh> SELECT * FROM
system_auth.network_permissions;

role | dcs
-----------+---------
roles/foo | {'dc1'}
roles/bar | {'dc2'}

(2 rows) cassandra@cqlsh> create keyspace test WITH REPLICATION = {'class' :
'NetworkTopologyStrategy', 'dc1' : 1, 'dc2' : 0}; cassandra@cqlsh> create
table test.test ( one text, two int, three text, PRIMARY KEY (one,two)
```

8. Validate that the foo role can connect to the first datacenter, and can access the local dc data.

```
$ ccm node1 cqlsh -u foo -p foo Connected to dc-security-demo at
127.0.0.1:9042. [cqlsh 5.0.1 | Cassandra 4.0-alpha4-SNAPSHOT | CQL spec 3.4.5
| Native protocol v4] Use HELP for help. foo@cqlsh> select * from test.test;
-- query data with local dc replica
```

9. Validate that the foo role can not connect to the second datacenter.

```
$ ccm node2 cqlsh -u foo -p foo Connection error: ('Unable to connect to any
servers', {'127.0.0.2': Unauthorized('Error from server: code=2100
[Unauthorized] message="You do not have access to this datacenter"',)})
```

10. Validate that the bar role can not connect to the first datacenter.

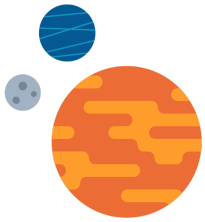
```
$ ccm node1 cqlsh -u bar -p bar Connection error: ('Unable to connect to any
servers', {'127.0.0.1': Unauthorized('Error from server: code=2100
[Unauthorized] message="You do not have access to this datacenter"',)})
```

11. Validate that the bar role can connect to the second datacenter, and can still read data that resides only in the first datacenter.

```
$ ccm node2 cqlsh -u bar -p bar Connected to dc-security-demo at
127.0.0.2:9042. [cqlsh 5.0.1 | Cassandra 4.0-alpha4-SNAPSHOT | CQL spec 3.4.5
| Native protocol v4] Use HELP for help. bar@cqlsh> select * from test.test;
-- query data in remote dc replica
```

16

Enforcing a 90-day Password Rotation



The following example demonstrates how to check users that have not updated their password in the last 90 days.

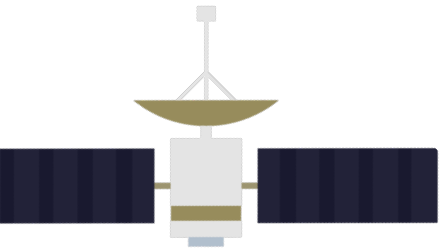
1. To list timestamps in microseconds of passwords of all users.

```
$ ccm node1 cqlsh -u cassandra -p cassandra Connected to dc-security-demo at
127.0.0.2:9042. [cqlsh 5.0.1 | Cassandra 4.0-alpha4-SNAPSHOT | CQL spec 3.4.5
| Native protocol v4] Use HELP for help. bar@cqlsh> select
role,WRITETIME(salted_hash) from system_auth.roles where can_login = True
ALLOW FILTERING;
```

2. The same list, now formatted, and from the command line so it becomes easy to script against.

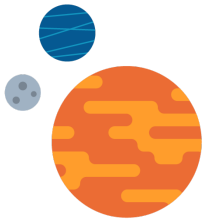
```
$ cqlsh -u cassandra -p cassandra -e "select role,writetime(salted_hash) from
system_auth.roles;"\

| grep " | " | grep -v "writetime(salted_hash)" \ | gawk '{print $1 " : "
strftime("%c",$3/1000000)}'
```



17

Using Java 11



Switching to Java 11 is recommended for - because of performance improvements - has known issues with timeouts which Java 11 and the new garbage collectors can address. Performance improvements on compaction and streaming will also improve elasticity (i.e. bootstrapping new nodes).

Switching to Java 11 involves

- Changing default java to JDK 11 (or setting JAVA_HOME)
- Configure and test conf/jvm11-server.options
- Restart one node and wait 3 hours to gather GC metrics
- Perform rolling restart on remaining nodes

Cassandra version 4.0 ships with separate jvm.options files for Java 8 and Java 11. These are the files:

- conf/jvm-server.options
- conf/jvm8-server.options
- conf/jvm11-server.options

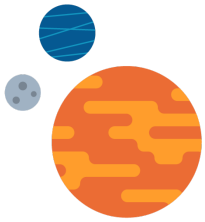
Upgrading to version 4.0 will work with an existing jvm.options file from version 3.11, so long as it is renamed to jvm-server.options and the jvm8-server.options and jvm11-server.options files are removed. This is not the recommended approach.

The recommended approach is to re-apply the settings found in the previous jvm.options file to the new jvm-server.options and jvm8-server.options files. The Java specific option files are mostly related to the garbage collection flags. Once these two files are updated and in place, it then becomes easier to configure the jvm11-server.options file, and simpler to switch from JDK 8 to JDK 11.

It is recommended to stick with the same GC when switching from JDK 8 to JDK 11. Even after the complete rolling restart it is recommended to wait a number of days running JDK 11 without incidents, before trialing alternative GC or JVM flags.

18

Switching to Incremental Repairs



Switching to incremental repairs means moving from only a schedule of full repairs, to a schedule of both incremental and full repairs. Incremental repairs should be performed on roughly a daily schedule. Full repairs on roughly a monthly schedule.

- The benefit of making this switch to - is
- A significant reduction in load, also reducing latency numbers,
- Smaller storage footprint by reducing gc_grace_seconds,
- More consistent data after bootstrapping new nodes, making bootstrapping per rack more reliable,
- Preparing for the use of transient replicas.

Switching to the new mixed schedules involves

- First, performing a full repair on all nodes after the upgrade to version 4.0,
- Initiating a planned schedule of incremental and full repairs for each keyspace and table,
- Reducing gc_grace_seconds on tables.

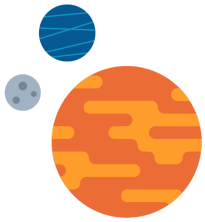
After upgrading to version 4.0 is complete it is important that a full repair is run across the cluster for all keyspaces/tables. Once this has been done a mixed schedule of incremental and full repairs for each table can be initiated.

Each keyspace and table may have different requirements for the frequency of both incremental and full repairs. This will depend upon the consistency model used, the data model, whether TTL or deletes occur, and the compaction strategy used. Both eventually consistent applications, and data models with TTL or manual deletes, have a higher dependency on repairs. Large tables using TTL will benefit more from shorter gc_grace_seconds configurations, made possible by more frequent incremental repairs. Full repairs against tables using TimeWindowCompactionStrategy is generally not recommended, but the introduction of incremental repairs at a frequency higher than the configured time_window is recommended and has potential consequence for the choice of consistency levels used.

It is recommended to use the Cassandra Reaper tool to schedule and run both full and incremental repairs. Reaper provides a number of advantages over the direct (or scripted) use of `nodetool repair`. Some of these advantages relate to improved cluster availability.

19

New cassandra.yaml Configurations and Schema Options



Changes in cassandra.yaml

`allocate_tokens_for_local_replication_factor`

Simpler and safer approach than allocate_tokens_for_keyspace. Important to see when using num_tokens <256.

`network_authorizer`

See Authorization by Data Center section above.

`commitlog_sync: group`

Adds a group mode where writes don't trigger commitlog flushes, but still waits for acknowledgement by the commitlog flush which happens at minimum every commitlog_sync_group_window_in_ms.

`periodic_commitlog_sync_lag_block_in_ms`

How often the commitlog in periodic mode flushes to disk.

`flush_compression`

The compression algorithm to apply to flushed SSTables. Flushes need to happen fast, to not block the flush writer and memtables, and write smaller SSTables than compactions so choosing a fast algorithm has value. Choices are none (no compression), fast, and table (as configured by the table's schema).

`repair_session_space_in_mb replaces`

Replaces repair_session_max_tree_depth, providing a more intuitive configuration to define.

`native_transport_frame_block_size_in_kb`

When checksumming is enabled, the chunk size to create checksums on.

`native_transport_allow_older_protocols`

Whether older protocol versions are allowed. It defaults to false but must be set to true when upgrading a cluster.

`native_transport_idle_timeout_in_ms`

Controls when idle client connections are close.

`concurrent_validations`

Number of simultaneous repair validations to allow. Useful to ensure concurrent_compactors are reserved for non-repair compactions.

`concurrent_materialized_view_builders`

Number of simultaneous materialized view builder tasks to allow.

`stream_entire_sstables`

See Zero Copy Streams section above.

`internode_tcp_* and internode_application_*`

Defensive settings for protecting Cassandra from network partitions. Part of the new messaging system.

`streaming_connections_per_host`

See Zero Copy Streams section above.

`enable_legacy_ssl_storage_port`

If enabled, will open up an encrypted listening socket on ssl_storage_port. Must be set to true when upgrading to 4.0.

`ideal_consistency_level`

Define the ideal consistency_level that writes are expected to work within the timeout window. This can include the asynchronous writes above the request statement's specified consistency level. Metrics are provided on how many writes meet this ideal_consistency_level.

`automatic_sstable_upgrade and max_concurrent_automatic_sstable_upgrades`

Enable the automatic upgrade of sstables, and limit the number of concurrent sstable upgrades.

`audit_logging_options`

See Auditing section above.

`full_query_logging_options`

See Full Query Logging (FQL) section above.

`corrupted_tombstone_strategy`

Validate tombstones on reads and compaction, can be either "disabled", "warn" or "exception".

`diagnostic_events_enabled`

Use to enable diagnostic_events. Diagnostic events are for observation purposes, but can introduce performance penalties if not used accurately and wisely.

`native_transport_flush_in_batches_legacy`

Use native transport TCP message coalescing. Worth testing on older kernels with fewer client connections available.

`repaired_data_tracking_for_* and report_unconfirmed_repaired_data_mismatches`

Enable tracking of repaired state of data during reads.

`cross_node_timeouts`

Now defaults to true. It is expected that everyone knows today the importance of keeping servers in sync with NTP.

`enable_materialized_views`

Now default to false. SASI is an experimental feature that requires some developer understanding to safely use.

`enable_sasi_indexes`

Now default to false. SASI is an experimental feature that requires some developer understanding to safely use.

`enable_transient_replication`

See Transient Replication above.

New CQL Data Types

POSITIVE AND NEGATIVE NAN AND INFINITY

The existing NAN and INFINITY data types are replaced with both positive and negative variants.

New CQL Table Options

`additional_write_policy`

How transient replication writes handle the transient replica.

`read_repair`

Introduces a configurable trade-off between monotonic quorum reads and partition level write atomicity.

`compression_level`

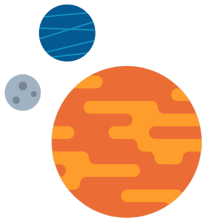
Allows defining compression level to the compressor. 0: fastest; 9: smallest.

`hybrid speculative_retry`

Allows specifying MIN() and MAX() around two types of speculative_retry values, providing bounds that ensure useful values even with one down or misbehaving node.

20

Conclusion



This document has described relevant new 4.0 features for -.

Some features that are recommended and uncontroversial are then described in further detail, contextual to - and going into more detail on how to apply them. Those features are Applying Authorization by Data Center, Using Java 11, and Incremental Repairs.

An upgrade to Cassandra 4.0 will address -'s needs around Auditing, Elasticity, and in addressing throughput and timeout issues. For both Auditing and Elasticity, further coordination with DataStax is recommended.

Auditing is feature complete, but additional coordination with DataStax is recommended to ensure a safe sustainable configuration is developed and deployed into production. The application of Auditing was suggested to provide a comparison or combination to the current query-tracer java agent in use. Full Query Logging is a relevant choice in this application too. The Full Query Logging feature is built upon the same code as Auditing so recommendations apply equally to both.

Elasticity, to -'s desire to bootstrap nodes within an hour, is a complex concern with no ready-to-go or single solution yet available. Cassandra 4.0 provides a number of improvements that will reduce bootstrap times. For bootstrapping 1TB nodes in under one hour further work is required, either on testing and merging in the RangeAwareCompaction work, or the Zero Copy Streaming implementation from DSE.

-'s uber_item cluster is currently running Cassandra version 2.1. The performance improvements from that to 3.11.6 are significant, and will already be doing a lot to address a number of -'s stated concerns and issues.

On the issue of Elasticity, it is worth noting that various shortcuts or smarts can be implemented. For example, elasticity can be constrained to be either the need for CPU elasticity (handling variations in traffic) or Storage elasticity (handling variations in storage load). If elasticity can be constrained to one of these problems then actions can be taken that provide results faster than bootstrapping a new node. For CPU elasticity it is possible to run coordinator-only Cassandra nodes. And for Storage elasticity volumes can be increased in size. With longer-term elasticity in mind, both these shortcuts can buy time while bootstrapping new nodes into the cluster. Work on improving elasticity for - is also being done by looking into Cassandra deployed in a Kubernetes environment, see the separate JIRA epic ticket for more information on that.

Cassandra 4.0 provides better restrictions, via the authorization mechanism, in preventing client connections to remote, or unwanted, data centers. This should be taken advantage of, but does not address -'s concern with enforcing a 90 day password rotation. To implement a 90 day password rotation policy it is recommended to use Cassandra's authorization Roles. Non-login roles can be used to provide the grants/restrictions, and login roles added to these non-login roles. A cqlsh based script can be executed daily that checks the cell timestamp value of the login role's password, alerting when it's older than 90 days.

Implementing CDC should be reserved when data flow through existing streaming solutions is not available, e.g. the use of Apache Kafka. Care needs to be taken to configure it correctly to minimize its impact on the cluster's performance and availability, and to correctly implement CDC consumers over all nodes with deduplication of data. Reach out to DataStax for more help.

As promised this document has provided answers on Auditing, Elasticity, Security, CDC, and addressed throughput and timeout issues.