DATASTAX

Apache Cassandra™ 2.0
Document

August 13, 2015

Contents

Contents

About Apache Cassandra

This guide provides information for developers and administrators on installing, configuring, and using the features and capabilities of Cassandra.

What is Apache Cassandra?

Apache Cassandra™ is a massively scalable open source NoSQL database. Cassandra is perfect for managing large amounts of structured, semi-structured, and unstructured data across multiple data centers and the cloud. Cassandra delivers continuous availability, linear scalability, and operational simplicity across many commodity servers with no single point of failure, along with a powerful dynamic data model designed for maximum flexibility and fast response times.

How does Cassandra work?

Cassandra's built-for-scale architecture means that it is capable of handling petabytes of information and thousands of concurrent users/operations per second.

Cassandra is a partitioned row store database	Cassandra's architecture allows any authorized user to connect to any node in any data center and access data using the CQL language. For ease of use, CQL uses a similar syntax to SQL. The most basic way to interact with Cassandra is using the CQL shell, cqlsh. Using cqlsh, you can create keyspaces and tables, insert and query tables, plus much more. If you prefer a graphical tool, you can use DataStax DevCenter. For production, DataStax supplies a number drivers so that CQL statements can be passed from client to cluster and back. Other administrative tasks can be accomplished using OpsCenter.
Automatic data distribution	Cassandra provides automatic data distribution across all nodes that participate in a *ring* or database cluster. There is nothing programmatic that a developer or administrator needs to do or code to distribute data across a cluster because data is transparently partitioned across all nodes in a cluster.
Built-in and customizable replication	Cassandra also provides built-in and customizable replication, which stores redundant copies of data across nodes that participate in a Cassandra ring. This means that if any node in a cluster goes down, one or more copies of that node's data is available on other machines in the cluster. Replication can be configured to work across one data center, many data centers, and multiple cloud availability zones.
Cassandra supplies linear scalability	Cassandra supplies linear scalability, meaning that capacity may be easily added simply by adding new nodes online. For example, if 2 nodes can handle 100,000 transactions per second, 4 nodes will support 200,000 transactions/sec and 8 nodes will tackle 400,000 transactions/sec:

What's new in Cassandra

Cassandra 2.0 included major enhancements to CQL, security, and performance. CQL for Cassandra 2.0.6 adds several important features including batching of conditional updates, static columns, and increased control over slicing of clustering columns.

Key features of Cassandra 2.0 are:

- Support for lightweight transactions

    - Use of the IF keyword in CQL INSERT and UPDATE statements
    - New SERIAL consistency level
- Triggers

    The first phase of support for triggers for firing an event that executes a set of programmatic logic, which runs either inside or outside a database cluster
- CQL paging support

    Paging of result sets of SELECT statements executed over a CQL native protocol 2 connection, which eliminates the need to use the token function to page through results. For example, to page through data in this table, a simple SELECT statement after Cassandra 2.0 replaces the complex one using the token function before Cassandra 2.0.

```
To query this table

CREATE TABLE timeline (
   user_id uuid,
   tweet_id timeuuid,
   tweet_author uuid,
   tweet_body text,
   PRIMARY KEY (userid, tweet_id)
)
```

```
Before Cassandra 2.0

SELECT *
FROM timeline
WHERE (user_id = :last_key
                AND tweet_id > :last_tweet)
     OR token (user_id) > token (:last_key)
```

```
After Cassandra 2.0

SELECT *
FROM timeline
```

- Improved authentication

SASL support for easier and better authentication over prior versions of the CQL native protocol

- Drop column support

  Re-introduction of the ALTER TABLE DROP command
- SELECT column aliases

  Support for column aliases in a SELECT statement, similar to aliases in RDBMS SQL:

  ```
  SELECT hdate AS hired_date
    FROM emp WHERE empid = 500
  ```
- Conditional DDL

  Conditionally tests for the existence of a table, keyspace, or index before issuing a DROP or CREATE statement using IF EXISTS or IF NOT EXISTS
- Index enhancements

  Indexing of any part, partition key or clustering columns, portion of a compound primary key
- One-off prepare and execute statements

  Use of a prepared statement, even for the single execution of a query to pass binary values for a statement, for example to avoid a conversion of a blob to a string, over a native protocol version 2 connection
- Performance enhancements

  - Off-heap partition summary
  - Eager retries support

    Sending the user request to other replicas before the query times out when a replica is unusually slow in delivering needed data
  - Compaction improvements

    Hybrid (leveled and size-tiered) compaction improvements to the leveled compaction strategy to reduce the performance overhead on read operations when compaction cannot keep pace with write-heavy workloads

Other changes in Cassandra 2.0 are:

- New commands to disable background compactions

  nodetool disableautocompaction and nodetool enableautocompaction
- A change to random token selection during cluster setup

  Auto_bootstrapping of a single-token node with no initial_token
- Removal of super column support

  Continued support for apps that query super columns, translation of super columns on the fly into CQL constructs and results
- Removal of the cqlsh ASSUME command

  Use the blobAsType and typeAsBlob conversion functions instead of ASSUME
- Cqlsh COPY command support for collections
- Inclusion of the native protocol version in the system.local table
- Inclusion of default_time_to_live, speculative_retry, and memtable_flush_period_in_ms in cqlsh DESCRIBE TABLE output
- Support for an empty list of values in the IN clause of SELECT, UPDATE, and DELETE commands, useful in Java Driver applications when passing empty arrays as arguments for the IN clause

CQL

Cassandra Query Language (CQL) is the default and primary interface into the Cassandra DBMS. Using CQL is similar to using SQL (Structured Query Language). CQL and SQL share the same abstract idea of a table constructed of tables and rows. The main difference from SQL is that Cassandra does not support joins or subqueries, except for batch analysis through Hive. Instead, Cassandra emphasizes denormalization through CQL features like collections and clustering specified at the schema level.

CQL is the recommended way to interact with Cassandra. Performance and the simplicity of reading and using CQL is an advantage of modern Cassandra over older Cassandra APIs.

The CQL documentation contains a data modeling section, examples, and command reference. The cqlsh utility for using CQL interactively on the command line is also covered.

Understanding the architecture

Architecture in brief

Cassandra is designed to handle big data workloads across multiple nodes with no single point of failure. Its architecture is based on the understanding that system and hardware failures can and do occur. Cassandra addresses the problem of failures by employing a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster. Each node exchanges information across the cluster every second. A sequentially written commit log on each node captures write activity to ensure data durability. Data is then indexed and written to an in-memory structure, called a memtable, which resembles a write-back cache. Once the memory structure is full, the data is written to disk in an SSTable data file. All writes are automatically partitioned and replicated throughout the cluster. Using a process called compaction Cassandra periodically consolidates SSTables, discarding obsolete data and tombstones (an indicator that data was deleted).

Cassandra is a row-oriented database. Cassandra's architecture allows any authorized user to connect to any node in any data center and access data using the CQL language. For ease of use, CQL uses a similar syntax to SQL. From the CQL perspective the database consists of tables. Typically, a cluster has one keyspace per application. Developers can access CQL through cqlsh as well as via drivers for application languages.

Client read or write requests can be sent to any node in the cluster. When a client connects to a node with a request, that node serves as the coordinator for that particular client operation. The coordinator acts as a proxy between the client application and the nodes that own the data being requested. The coordinator determines which nodes in the ring should get the request based on how the cluster is configured. For more information, see Client requests.

Key structures

- Node

  Where you store your data. It is the basic infrastructure component of Cassandra.
- Data center

  A collection of related nodes. A data center can be a physical data center or virtual data center. Different workloads should use separate data centers, either physical or virtual. Replication is set by data center. Using separate data centers prevents Cassandra transactions from being impacted by other workloads and keeps requests close to each other for lower latency. Depending on the replication factor, data can be written to multiple data centers. However, data centers should never span physical locations.
- Cluster

  A cluster contains one or more data centers. It can span physical locations.
- Commit log

  All data is written first to the commit log for durability. After all its data has been flushed to SSTables, it can be archived, deleted, or recycled.
- Table

  A collection of ordered columns fetched by row. A row consists of columns and have a primary key. The first part of the key is a column name.
- SSTable

13

A sorted string table (SSTable) is an immutable data file to which Cassandra writes memtables periodically. SSTables are append only and stored on disk sequentially and maintained for each Cassandra table.

Key components for configuring Cassandra

- Gossip

  A peer-to-peer communication protocol to discover and share location and state information about the other nodes in a Cassandra cluster. Gossip information is also persisted locally by each node to use immediately when a node restarts.

- Partitioner

  A partitioner determines how to distribute the data across the nodes in the cluster and which node to place the first copy of data on. Basically, a partitioner is a hash function for computing the token of a partition key. Each row of data is uniquely identified by a partition key and distributed across the cluster by the value of the token. The Murmur3Partitioner is the default partitioning strategy for new Cassandra clusters and the right choice for new clusters in almost all cases.

  You must set the partitioner and assign the node a num_tokens value for each node. The number of tokens you assign depends on the hardware capabilities of the system. If not using virtual nodes (vnodes), use the initial_token setting instead.

- Replication factor

  The total number of replicas across the cluster. A replication factor of 1 means that there is only one copy of each row on one node. A replication factor of 2 means two copies of each row, where each copy is on a different node. All replicas are equally important; there is no primary or master replica. You define the replication factor for each data center. Generally you should set the replication strategy greater than one, but no more than the number of nodes in the cluster.

- Replica placement strategy

  Cassandra stores copies (replicas) of data on multiple nodes to ensure reliability and fault tolerance. A replication strategy determines which nodes to place replicas on. The first replica of data is simply the first copy; it is not unique in any sense. The NetworkTopologyStrategy is highly recommended for most deployments because it is much easier to expand to multiple data centers when required by future expansion.

  When creating a keyspace, you must define the replica placement strategy and the number of replicas you want.

- Snitch

  A snitch defines groups of machines into data centers and racks (the topology) that the replication strategy uses to place replicas.

  You must configure a snitch when you create a cluster. All snitches use a dynamic snitch layer, which monitors performance and chooses the best replica for reading. It is enabled by default and recommended for use in most deployments. Configure dynamic snitch thresholds for each node in the `cassandra.yaml` configuration file.

  The default SimpleSnitch does not recognize data center or rack information. Use it for single-data center deployments or single-zone in public clouds. The GossipingPropertyFileSnitch is recommended for production. It defines a node's data center and rack and uses gossip for propagating this information to other nodes.

- The cassandra.yaml configuration file

  The main configuration file for setting the initialization properties for a cluster, caching parameters for tables, properties for tuning and resource utilization, timeout settings, client connections, backups, and security.

  By default, a node is configured to store the data it manages in a directory set in the `cassandra.yaml` file.

- Packaged installs: `/var/lib/cassandra`
- Tarball installs: `install_location/data/data`

  In a production cluster deployment, you can change the commitlog-directory to a different disk drive from the data_file_directories.

- System keyspace table properties

  You set storage configuration attributes on a per-keyspace or per-table basis programmatically or using a client application, such as CQL.

Internode communications (gossip)

Gossip is a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about. The gossip process runs every second and exchanges state messages with up to three other nodes in the cluster. The nodes exchange information about themselves and about the other nodes that they have gossiped about, so all nodes quickly learn about all other nodes in the cluster. A gossip message has a version associated with it, so that during a gossip exchange, older information is overwritten with the most current state for a particular node.

To prevent problems in gossip communications, use the same list of seed nodes for all nodes in a cluster. This is most critical the first time a node starts up. By default, a node remembers other nodes it has gossiped with between subsequent restarts. The seed node designation has no purpose other than bootstrapping the gossip process for new nodes joining the cluster. Seed nodes are *not* a single point of failure, nor do they have any other special purpose in cluster operations beyond the bootstrapping of nodes.

Attention: In multiple data-center clusters, the seed list should include at least one node from each data center (replication group). More than a single seed node per data center is recommended for fault tolerance. Otherwise, gossip has to communicate with another data center when bootstrapping a node. Making every node a seed node is **not** recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per data center).

Failure detection and recovery

Failure detection is a method for locally determining from gossip state and history if another node in the system is up or down. Cassandra uses this information to avoid routing client requests to unreachable nodes whenever possible. (Cassandra can also avoid routing requests to nodes that are alive, but performing poorly, through the dynamic snitch.)

The gossip process tracks state from other nodes both directly (nodes gossiping directly to it) and indirectly (nodes communicated about secondhand, thirdhand, and so on). Rather than have a fixed threshold for marking failing nodes, Cassandra uses an accrual detection mechanism to calculate a per-node threshold that takes into account network performance, workload, and historical conditions. During gossip exchanges, every node maintains a sliding window of inter-arrival times of gossip messages from other nodes in the cluster. Configuring the phi_convict_threshold property adjusts the sensitivity of the failure detector. Lower values increase the likelihood that an unresponsive node will be marked as down, while higher values decrease the likelihood that transient failures will cause a node failure. Use the default value for most situations, but increase it to 12 for Amazon EC2 (due to frequently enountered network congestion).

Node failures can result from various causes such as hardware failures and network outages. Node outages are often transient but can last for extended periods. Because a node outage rarely signifies a permanent departure from the cluster it does not automatically result in permanent removal of the node from the ring. Other nodes will periodically try to re-establish contact with failed nodes to see if they are

back up. To permanently change a node's membership in a cluster, administrators must explicitly add or remove nodes from a Cassandra cluster using the nodetool utility or OpsCenter.

When a node comes back online after an outage, it may have missed writes for the replica data it maintains. Once the failure detector marks a node as down, missed writes are stored by other replicas for a period of time providing hinted handoff is enabled. If a node is down for longer than max_hint_window_in_ms (3 hours by default), hints are no longer saved. Nodes that die may have stored undelivered hints. Run a repair after recovering a node that has been down for an extended period. Moreover, you should routinely run nodetool repair on all nodes to ensure they have consistent data.

For more explanation about hint storage, see Modern hinted handoff.

Data distribution and replication

In Cassandra, data distribution and replication go together. Data is organized by table and identified by a primary key, which determines which node the data is stored on. Replicas are copies of rows. When data is first written, it is also referred to as a replica.

Factors influencing replication include:

- Virtual nodes: assigns data ownership to physical machines.
- Partitioner: partitions the data across the cluster.
- Replication strategy: determines the replicas for each row of data.
- Snitch: defines the topology information that the replication strategy uses to place replicas.

Consistent hashing

Consistent hashing allows distributing data across a cluster which minimizes reorganization when nodes are added or removed. Consistent hashing partitions data based on the partition key. (For an explanation of partition keys and primary keys, see the Data modeling example in *CQL for Cassandra 2.0*.)

For example, if you have the following data:

name	age	car	gender
jim	36	camaro	M
carol	37	bmw	F
johnny	12		M
suzy	10		F

Cassandra assigns a hash value to each partition key:

Partition key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

Each node in the cluster is responsible for a range of data based on the hash value:

Hash values in a 4 node cluster
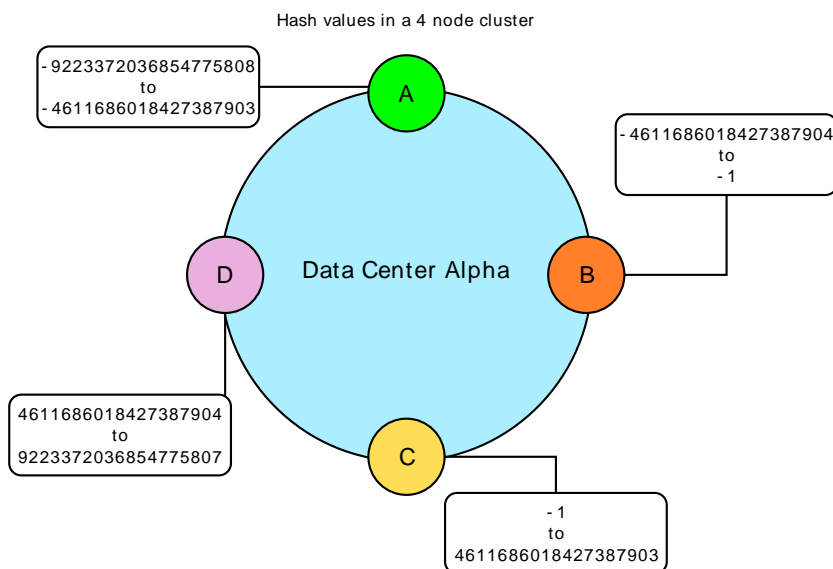
Cassandra places the data on each node according to the value of the partition key and the range that the node is responsible for. For example, in a four node cluster, the data in this example is distributed as follows:

Node	Start range	End range	Partition key	Hash value
A	-9223372036854775808	-4611686018427387903	johnny	-6723372854036780875
B	-4611686018427387904	-1	jim	-2245462676723223822
C	0	4611686018427387903	suzy	1168604627387940318
D	4611686018427387904	9223372036854775807	carol	7723358927203680754

Virtual nodes

Vnodes simplify many tasks in Cassandra:

• You no longer have to calculate and assign tokens to each node.
• Rebalancing a cluster is no longer necessary when adding or removing nodes. When a node joins the cluster, it assumes responsibility for an even portion of data from the other nodes in the cluster. If a node fails, the load is spread evenly across other nodes in the cluster.
• Rebuilding a dead node is faster because it involves every other node in the cluster.
• Improves the use of heterogeneous machines in a cluster. You can assign a proportional number of vnodes to smaller and larger machines.

For more information, see the article Virtual nodes in Cassandra 1.2 and Enabling virtual nodes on an existing production cluster.

How data is distributed across a cluster (using virtual nodes)

Prior to version 1.2, you had to calculate and assign a single token to each node in a cluster. Each token determined the node's position in the ring and its portion of data according to its hash value. Starting in version 1.2, Cassandra allows many tokens per node. The new paradigm is called virtual nodes (vnodes). Vnodes allow each node to own a large number of small partition ranges distributed throughout the cluster. Vnodes also use consistent hashing to distribute data but using them doesn't require token generation and assignment.
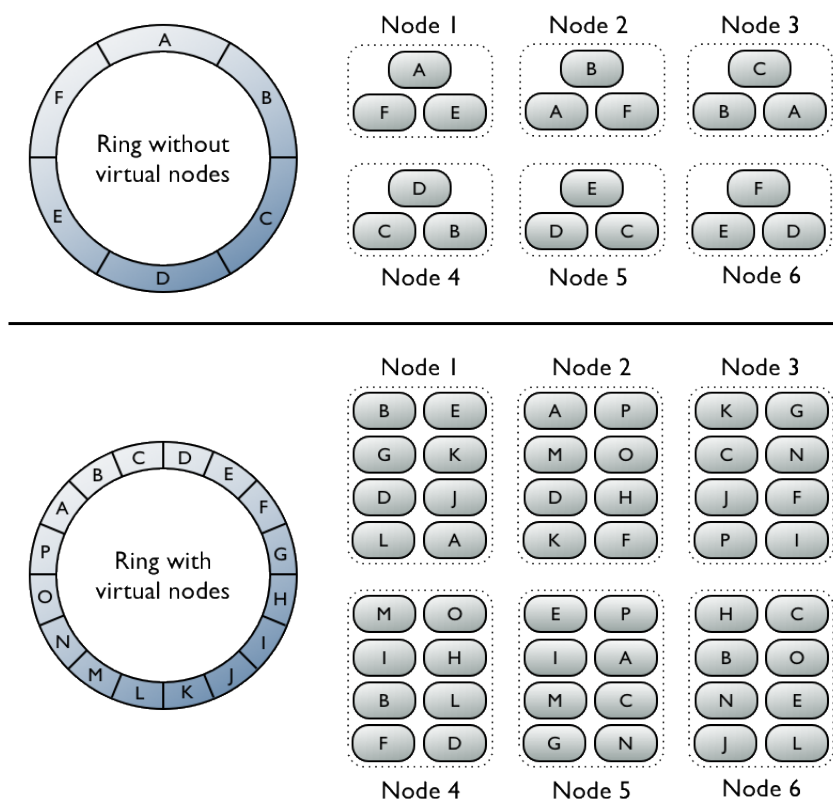
Figure 1: Virtual vs single-token architecture

The top portion of the graphic shows a cluster without vnodes. In this paradigm, each node is assigned a single token that represents a location in the ring. Each node stores data determined by mapping the partition key to a token value within a range from the previous node to its assigned value. Each node also contains copies of each row from other nodes in the cluster. For example, range E replicates to nodes 5, 6, and 1. Notice that a node owns exactly one contiguous partition range in the ring space.

The bottom portion of the graphic shows a ring with vnodes. Within a cluster, virtual nodes are randomly selected and non-contiguous. The placement of a row is determined by the hash of the partition key within many smaller partition ranges belonging to each node.

Data replication

Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance. A replication strategy determines the nodes where replicas are placed. The total number of replicas across the cluster is referred to as the replication factor. A replication factor of 1 means that there is only one copy of each row on one node. A replication factor of 2 means two copies of each row, where each copy is on a different node. All replicas are equally important; there is no primary or master replica. As a general rule, the replication factor should not exceed the number of nodes in the cluster. However, you can increase the replication factor and then add the desired number of nodes later.

Two replication strategies are available:

- `SimpleStrategy`: Use for a single data center only. If you ever intend more than one data center, use the `NetworkTopologyStrategy`.
- `NetworkTopologyStrategy`: Highly recommended for most deployments because it is much easier to expand to multiple data centers when required by future expansion.

SimpleStrategy

Use only for a single data center. `SimpleStrategy` places the first replica on a node determined by the partitioner. Additional replicas are placed on the next nodes clockwise in the ring without considering topology (rack or data center location).

NetworkTopologyStrategy

Use `NetworkTopologyStrategy` when you have (or plan to have) your cluster deployed across multiple data centers. This strategy specify how many replicas you want in each data center.

`NetworkTopologyStrategy` places replicas in the same data center by walking the ring clockwise until reaching the first node in another rack. `NetworkTopologyStrategy` attempts to place replicas on distinct racks because nodes in the same rack (or similar physical grouping) often fail at the same time due to power, cooling, or network issues.

When deciding how many replicas to configure in each data center, the two primary considerations are (1) being able to satisfy reads locally, without incurring cross data-center latency, and (2) failure scenarios. The two most common ways to configure multiple data center clusters are:

- Two replicas in each data center: This configuration tolerates the failure of a single node per replication group and still allows local reads at a consistency level of `ONE`.
- Three replicas in each data center: This configuration tolerates either the failure of a one node per replication group at a strong consistency level of `LOCAL_QUORUM` or multiple node failures per data center using consistency level `ONE`.

Asymmetrical replication groupings are also possible. For example, you can have three replicas in one data center to serve real-time application requests and use a single replica elsewhere for running analytics.

Choosing keyspace replication options

To set the replication strategy for a keyspace, see CREATE KEYSPACE.

When you use `NetworkTopologyStrategy`, during creation of the keyspace, you use the data center names defined for the snitch used by the cluster. To place replicas in the correct location, Cassandra requires a keyspace definition that uses the snitch-configured data center names. For example, if the cluster uses the PropertyFileSnitch, create the keyspace using the user-defined data center and rack names in the `cassandra-topologies.properties` file. If the cluster uses the Ec2Snitch, create the keyspace using EC2 data center and rack names.

Partitioners

A partitioner determines how data is distributed across the nodes in the cluster (including replicas). Basically, a partitioner is a function for deriving a token representing a row from its partion key, typically by hashing. Each row of data is then distributed across the cluster by the value of the token.

Both the `Murmur3Partitioner` and `RandomPartitioner` use tokens to help assign equal portions of data to each node and evenly distribute data from all the tables throughout the ring or other grouping, such as a keyspace. This is true even if the tables use different partition keys, such as usernames or timestamps. Moreover, the read and write requests to the cluster are also evenly distributed and load balancing is simplified because each part of the hash range receives an equal number of rows on average. For more detailed information, see Consistent hashing.

Cassandra offers the following partitioners:

- `Murmur3Partitioner` (default): uniformly distributes data across the cluster based on `MurmurHash` hash values.
- `RandomPartitioner`: uniformly distributes data across the cluster based on `MD5` hash values.
- `ByteOrderedPartitioner`: keeps an ordered distribution of data lexically by key bytes

The `Murmur3Partitioner` is the default partitioning strategy for new Cassandra clusters and the right choice for new clusters in almost all cases.

19

Set the partitioner in the cassandra.yaml file:

- `Murmur3Partitioner`: org.apache.cassandra.dht.Murmur3Partitioner
- `RandomPartitioner`: org.apache.cassandra.dht.RandomPartitioner
- `ByteOrderedPartitioner`: org.apache.cassandra.dht.ByteOrderedPartitioner

Note: If using virtual nodes (vnodes), you do **not** need to calculate the tokens. If not using vnodes, you **must** calculate the tokens to assign to the initial_token parameter in the `cassandra.yaml` file. See Generating tokens and use the method for the type of partitioner you are using.

Murmur3Partitioner

The `Murmur3Partitioner` provides faster hashing and improved performance than the previous default partitioner (`RandomPartitioner`). You can only use `Murmur3Partitioner` for new clusters; you cannot change the partitioner in existing clusters. The `Murmur3Partitioner` uses the `MurmurHash` function. This hashing function creates a 64-bit hash value of the partition key. The possible range of hash values is from -2^{63} to $+2^{63}-1$.

When using the `Murmur3Partitioner`, you can page through all rows using the token function in a CQL query.

RandomPartitioner

The `RandomPartitioner` was the default partitioner prior to Cassandra 1.2. It is included for backwards compatibility. You can use it in later versions of Cassandra, even when using virtual nodes (vnodes). However, if you don't use vnodes, you must calculate the tokens, as described in Generating tokens.

The `RandomPartition` distributes data evenly across the nodes using an MD5 hash value of the row key. The possible range of hash values is from 0 to $2^{127}-1$.

When using the `RandomPartitioner`, you can page through all rows using the token function in a CQL query.

ByteOrderedPartitioner

The `ByteOrderedPartitioner` is included for backwards compatibility. Cassandra provides this partitioner for ordered partitioning. This partitioner orders rows lexically by key bytes. You calculate tokens by looking at the actual values of your partition key data and using a hexadecimal representation of the leading character(s) in a key. For example, if you wanted to partition rows alphabetically, you could assign an A token using its hexadecimal representation of 41.

Using the ordered partitioner allows ordered scans by primary key. This means you can scan rows as though you were moving a cursor through a traditional index. For example, if your application has user names as the partition key, you can scan rows for users whose names fall between Jake and Joe. This type of query is not possible using randomly partitioned partition keys because the keys are stored in the order of their `MD5` hash (not sequentially).

Although having the capability to do range scans on rows sounds like a desirable feature of ordered partitioners, there are ways to achieve the same functionality using table indexes.

Using an ordered partitioner is not recommended for the following reasons:

Difficult load balancing

More administrative overhead is required to load balance the cluster. An ordered partitioner requires administrators to manually calculate partition ranges based on their estimates of the partition key distribution. In practice, this requires actively moving node tokens around to accommodate the actual distribution of data once it is loaded.

Sequential writes can cause hot spots

If your application tends to write or update a sequential block of rows at a time, then the writes are not be distributed across the cluster; they all go to one node. This is frequently a problem for applications dealing with timestamped data.

Uneven load balancing for multiple tables

If your application has multiple tables, chances are that those tables have different row keys and different distributions of data. An ordered partitioner that is balanced for one table may cause hot spots and uneven distribution for another table in the same cluster.

Snitches

A snitch determines which data centers and racks nodes belong to. Snitches inform Cassandra about the network topology so that requests are routed efficiently and allows Cassandra to distribute replicas by grouping machines into data centers and racks. Specifically, the replication strategy places the replicas based on the information provided by the new snitch. All nodes must return to the same rack and data center. Cassandra does its best not to have more than one replica on the same rack (which is not necessarily a physical location).

Note: If you change snitches, you may need to perform additional steps because the snitch affects where replicas are placed. See Switching snitches.

Dynamic snitching

By default, all snitches also use a dynamic snitch layer that monitors read latency and, when possible, routes requests away from poorly-performing nodes. The dynamic snitch is enabled by default and is recommended for use in most deployments. For information on how this works, see Dynamic snitching in Cassandra: past, present, and future. Configure dynamic snitch thresholds for each node in the `cassandra.yaml` configuration file.

For more information, see the properties listed under Failure detection and recovery.

SimpleSnitch

The SimpleSnitch is used only for single-data center deployments. The SimpleSnitch (the default) does not recognize data center or rack information. Use it for single-data center deployments or single-zone in public clouds.

Using a SimpleSnitch, you define the keyspace to use SimpleStrategy and specify a replication factor.

RackInferringSnitch

The RackInferringSnitch determines the location of nodes by rack and data center, which are assumed to correspond to the 3rd and 2nd octet of the node's IP address, respectively.

110.100.200.105

21
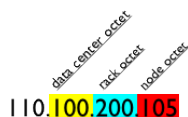
PropertyFileSnitch

About this task

This snitch determines proximity is determined by rack and data center. It uses a user-defined description of the network details located in the `cassandra-topology.properties` file. Use this snitch when your node IPs are not uniform or if you have complex replication grouping requirements. When using this snitch, you can define your data center names to be whatever you want. Make sure that the data center names you define correlate to the name of your data centers in your keyspace definition. Every node in the cluster should be described in the `cassandra-topology.properties` file, and this file should be exactly the same on every node in the cluster.

The location of the `cassandra-topology.properties` file depends on the type of installation:

* Packaged installs: `/etc/cassandra/cassandra-topology.properties`
* Tarball installs: `install_location/conf/cassandra-topology.properties`

Procedure

If you had non-uniform IPs and two physical data centers with two racks in each, and a third logical data center for replicating analytics data, the `cassandra-topology.properties` file might look like this:

Note: Data center and rack names are case-sensitive.

```
# Data Center One

175.56.12.105  =DC1:RAC1
175.50.13.200  =DC1:RAC1
175.54.35.197  =DC1:RAC1

120.53.24.101  =DC1:RAC2
120.55.16.200  =DC1:RAC2
120.57.102.103 =DC1:RAC2

# Data Center Two

110.56.12.120  =DC2:RAC1
110.50.13.201  =DC2:RAC1
110.54.35.184  =DC2:RAC1

50.33.23.120   =DC2:RAC2
50.45.14.220   =DC2:RAC2
50.17.10.203   =DC2:RAC2

# Analytics Replication Group

172.106.12.120 =DC3:RAC1
172.106.12.121 =DC3:RAC1
172.106.12.122 =DC3:RAC1

# default for unknown nodes default =DC3:RAC1
```

GossipingPropertyFileSnitch

This snitch is recommended for production. It uses rack and data center information for the local node defined in the `cassandra-rackdc.properties` file and propagates this information to other nodes via gossip.

The `cassandra-rackdc.properties` file defines the default data center and rack used by this snitch:

Note: Data center and rack names are case-sensitive.

```
dc=DC1
rack=RAC1
```

To save bandwidth, add the `prefer_local=true` option. This option tells Cassandra to use the local IP address when communication is not across different data centers.

The location of the `conf` directory depends on the type of installation:

- Packaged installs: `/etc/cassandra/cassandra-rackdc.properties`
- Tarball installs: *install_location*`/conf/cassandra-rackdc.properties`

To migrate from the PropertyFileSnitch to the GossipingPropertyFileSnitch, update one node at a time to allow gossip time to propagate. The PropertyFileSnitch is used as a fallback when `cassandra-topologies.properties` is present.

Ec2Snitch

Use the Ec2Snitch for simple cluster deployments on Amazon EC2 where all nodes in the cluster are within a single region.

The region name is treated as the data center name and availability zones are treated as racks within a data center. For example, if a node is in the us-east-1 region, us-east is the data center name and 1 is the rack location. (Racks are important for distributing replicas, but not for data center naming.) Because private IPs are used, this snitch does not work across multiple regions.

If you are using only a single data center, you do not need to specify any properties.

If you need multiple data centers, set the dc_suffix options in the `cassandra-rackdc.properties` file. Any other lines are ignored. The location of this file depends on the type of installation:

- Packaged installs: `/etc/cassandra/cassandra-rackdc.properties`
- Tarball installs: *install_location*`/conf/cassandra-rackdc.properties`

For example, for each node within the us-east region, specify the data center in its `cassandra-rackdc.properties` file:

Note: Data center names are case-sensitive.

- **node0**

  ```
  dc_suffix=_1_cassandra
  ```
- **node1**

  ```
  dc_suffix=_1_cassandra
  ```
- **node2**

  ```
  dc_suffix=_1_cassandra
  ```
- **node3**

  ```
  dc_suffix=_1_cassandra
  ```
- **node4**

  ```
  dc_suffix=_1_analytics
  ```
- **node5**

  ```
  dc_suffix=_1_search
  ```

This results in three data centers for the region:

```
us-east_1_cassandra
us-east_1_analytics
us-east_1_search
```

Note: The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200.

Keyspace strategy options

When defining your keyspace strategy options, use the EC2 region name, such as ``us-east``, as your data center name.

Ec2MultiRegionSnitch

Use the Ec2MultiRegionSnitch for deployments on Amazon EC2 where the cluster spans multiple regions.

Configuring cassandra.yaml for cross-region communication

The Ec2MultiRegionSnitch uses public IP designated in the broadcast_address to allow cross-region connectivity. Configure each node as follows:

1. In the `cassandra.yaml`, set the listen_address to the *private* IP address of the node, and the broadcast_address to the *public* IP address of the node.

   This allows Cassandra nodes in one EC2 region to bind to nodes in another region, thus enabling multiple data center support. For intra-region traffic, Cassandra switches to the private IP after establishing a connection.

   The location of `cassandra.yaml` file depends on the type of installation:

   - Packaged installs: `/etc/cassandra/cassandra-rackdc.properties`
   - Tarball installs: `install_location/conf/cassandra-rackdc.properties`

2. Set the addresses of the seed nodes in the `cassandra.yaml` file to that of the *public* IP. Private IP are not routable between networks. For example:

   ```
   seeds: 50.34.16.33, 60.247.70.52
   ```

   To find the public IP address, from each of the seed nodes in EC2:

   ```
   $ curl http://instance-data/latest/meta-data/public-ipv4
   ```

   Note: Do not make all nodes seeds, see Internode communications (gossip).

3. Be sure that the storage_port or ssl_storage_port is open on the public IP firewall.

Configuring the snitch for cross-region communication

In EC2 deployments, the The region name is treated as the data center name and availability zones are treated as racks within a data center. For example, if a node is in the us-east-1 region, us-east is the data center name and 1 is the rack location. (Racks are important for distributing replicas, but not for data center naming.)

For each node, specify its data center in the `cassandra-rackdc.properties`. The dc_suffix option defines the data centers used by the snitch. Any other lines are ignored.

In the example below, there are two cassandra data centers and each data center is named for its workload. The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (Data center names are case-sensitive.)

Region: us-east	Region: us-west
Node and data center:	Node and data center:

- **node0**
  `dc_suffix=_1_cassandra`
- **node1**
  `dc_suffix=_1_cassandra`
- **node2**
	- **node0**
	  `dc_suffix=_1_cassandra`
	- **node1**
	  `dc_suffix=_1_cassandra`
	- **node2**

Region: us-east	Region: us-west
`dc_suffix=_2_cassandra` • **node3**	`dc_suffix=_2_cassandra` • **node3**
`dc_suffix=_2_cassandra` • **node4**	`dc_suffix=_2_cassandra` • **node4**
`dc_suffix=_1_analytics` • **node5**	`dc_suffix=_1_analytics` • **node5**
`dc_suffix=_1_search`	`dc_suffix=_1_search`

This results in four us-east data centers:
 	
`us-east_1_cassandra`
`us-east_2_cassandra`
`us-east_1_analytics`
`us-east_1_search`
	This results in four us-west data centers:
	
	`us-west_1_cassandra`
	`us-west_2_cassandra`
	`us-west_1_analytics`
	`us-west_1_search`

Keyspace strategy options

When defining your keyspace strategy options, use the EC2 region name, such as ``us-east``, as your data center name.

GoogleCloudSnitch

Use the GoogleCloudSnitch for Cassandra deployments on Google Cloud Platform across one or more regions. The region is treated as a data center and the availability zones are treated as racks within the data center. All communication occurs over private IP addresses within the same logical network.

Note: Data center and rack names are case-sensitive.

CloudstackSnitch

Use the CloudstackSnitch for Apache Cloudstack environments. Because zone naming is free-form in Apache Cloudstack, this snitch uses the widely-used <country> <location> <az> notation.

Client requests

Client read or write requests can be sent to any node in the cluster because all nodes in Cassandra are peers. When a client connects to a node and issues a read or write request, that node serves as the coordinator for that particular client operation.

The job of the coordinator is to act as a proxy between the client application and the nodes (or replicas) that own the data being requested. The coordinator determines which nodes in the ring should get the request based on the cluster configured partitioner and replica placement strategy.

Planning a cluster deployment

When planning a Cassandra cluster deployment, you should have a good idea of the initial volume of data you plan to store and a good estimate of your typical application workload.

Selecting hardware for enterprise implementations

Choosing appropriate hardware depends on selecting the right balance of the following resources: memory, CPU, disk type, number of nodes, and network. Anti-patterns in Cassandra also contains important information about hardware, particularly SAN storage, NAS devices, and NFS.

Caution: Do not use a machine suited for development for load testing or production. Failure may result.

Memory

The more memory a Cassandra node has, the better read performance. More RAM allows for larger cache sizes and reduces disk I/O for reads. More RAM also allows memory tables (memtables) to hold more recently written data. Larger memtables lead to a fewer number of SSTables being flushed to disk and fewer files to scan during a read. The ideal amount of RAM depends on the anticipated size of your hot data.

For both dedicated hardware and virtual environments:

- Production: 16GB to 64GB; the minimum is 8GB.
- Development in non-loading testing environments: no less than 4GB.
- For setting Java heap space, see Tuning Java resources.

CPU

Insert-heavy workloads are CPU-bound in Cassandra before becoming memory-bound. (All writes go to the commit log, but Cassandra is so efficient in writing that the CPU is the limiting factor.) Cassandra is highly concurrent and uses as many CPU cores as available:

- Production environments:
  - For dedicated hardware, 8-core CPU processors are the current price-performance sweet spot.
  - For virtual environments, 4 to 8-core CPU processors.
- Development in non-loading testing environments:
  - For dedicated hardware, 2-core CPU processors.
  - For virtual environments, 2-core CPU processors.
- For virtual environments, consider using a provider that allows CPU bursting, such as Rackspace Cloud Servers.

Spinning disks versus Solid State Drives

SSDs are recommended for Cassandra. The NAND Flash chips that power SSDs provide extremely low-latency response times for random reads while supplying ample sequential write performance for compaction operations. In recent years, drive manufacturers have improved overall endurance, usually in conjunction with spare (unexposed) capacity. Additionally, because PBW/DWPD ratings are probabilistic estimates based on worst case scenarios, such as random write workloads, and Cassandra does only large sequential writes, drives significantly exceed their endurance ratings. However, it is important to plan for drive failures and have spares available. A large variety of SSDs are available on the market from server vendors and third-party drive manufacturers.

For purchasing SSDs, the best recommendation is to make SSD endurance decisions not based on workload, but on how difficult it is to change drives when they fail. Remember, your data is protected because Cassandra replicates data across the cluster. Buying strategies include:

- If drives are quickly available, buy the cheapest drives that provide the performance you want.
- If it is more challenging to swap the drives, consider higher endurance models, possibly starting in the mid range, and then choose replacements of higher or lower endurance based on the failure rates of the initial model chosen.

- Always buy cheap SSDs and keep several spares online and unused in the servers until the initial drives fail. This way you can replace the drives without touching the server.

DataStax customers that need help in determining the most cost-effective option for a given deployment and workload, should contact their Solutions Engineer or Architect.

Disks space

Disk space depends on usage, so it's important to understand the mechanism. Cassandra writes data to disk when appending data to the commit log for durability and when flushing memtable to SSTable data files for persistent storage. The commit log has a different access pattern (read/writes ratio) than the pattern for accessing data from SSTables. This is more important for spinning disks than for SSDs (solid state drives). See the recommendations below.

SSTables are periodically compacted. Compaction improves performance by merging and rewriting data and discarding old data. However, depending on the type of compaction strategy and size of the compactions, during compaction disk utilization and data directory volume temporarily increases. For large compactions, leave an adequate amount of free disk space available on a node: 50% (worst case) for SizeTieredCompactionStrategy and DateTieredCompactionStrategy, and 10% for LeveledCompactionStrategy. For more information about compaction, see:

- Compaction
- The Apache Cassandra storage engine
- Leveled Compaction in Apache Cassandra
- When to Use Leveled Compaction
- DateTieredCompactionStrategy: Compaction for Time Series Data

For information on calculating disk size, see Calculating usable disk capacity.

Recommendations:

Capacity per node

Most workloads work best with a capacity under 500GB to 1TB per node depending on I/O. Maximum recommended capacity for Cassandra 1.2 and later is 3 to 5TB per node for uncompressed data. For Cassandra 1.1, it is 500 to 800GB per node. Be sure to account for replication.

Capacity and I/O

When choosing disks, consider both capacity (how much data you plan to store) and I/O (the write/read throughput rate). Some workloads are best served by using less expensive SATA disks and scaling disk capacity and I/O by adding more nodes (with more RAM).

Number of disks - SATA

Ideally Cassandra needs at least two disks, one for the commit log and the other for the data directories. At a minimum the commit log should be on its own partition.

Commit log disk - SATA

The disk not need to be large, but it should be fast enough to receive all of your writes as appends (sequential I/O).

Commit log disk - SSD

Unlike spinning disks, it's all right to store both commit logs and SSTables are on the same mount point.

Data disks

Use one or more disks and make sure they are large enough for the data volume and fast enough to both satisfy reads that are not cached in memory and to keep up with compaction.

RAID on data disks

It is generally not necessary to use RAID for the following reasons:

- Data is replicated across the cluster based on the replication factor you've chosen.
- Starting in version 1.2, Cassandra includes a JBOD (Just a bunch of disks) feature to take care of disk management. Because Cassandra properly reacts to a disk failure either by stopping the affected node

or by blacklisting the failed drive, you can deploy Cassandra nodes with large disk arrays without the overhead of RAID 10. You can configure Cassandra to stop the affected node or blacklist the drive according to your availability/consistency requirements.

Also see Recovering from a single disk failure using JBOD.

RAID on the commit log disk

Generally RAID is not needed for the commit log disk. Replication adequately prevents data loss. If you need the extra redundancy, use RAID 1.

Extended file systems

DataStax recommends deploying Cassandra on either XFS or ext4. On ext2 or ext3, the maximum file size is 2TB even using a 64-bit kernel. On ext4 it is 16TB.

Because Cassandra can use almost half your disk space for a single file, use XFS when using large disks, particularly if using a 32-bit kernel. XFS file size limits are 16TB max on a 32-bit kernel, and essentially unlimited on 64-bit.

Number of nodes

Prior to version 1.2, the recommended size of disk space per node was 300 to 500GB. Improvement to Cassandra 1.2, such as JBOD support, virtual nodes (vnodes), off-heap Bloom filters, and parallel leveled compaction (SSD nodes only), allow you to use few machines with multiple terabytes of disk space.

Network

Since Cassandra is a distributed data store, it puts load on the network to handle read/write requests and replication of data across nodes. Be sure that your network can handle traffic between nodes without bottlenecks. You should bind your interfaces to separate Network Interface Cards (NIC). You can use public or private depending on your requirements.

- Recommended bandwidth is 1000 Mbit/s (gigabit) or greater.
- Thrift/native protocols use the rpc_address.
- Cassandra's internal storage protocol uses the listen_address.

Cassandra efficiently routes requests to replicas that are geographically closest to the coordinator node and chooses a replica in the same rack if possible; it always chooses replicas located in the same data center over replicas in a remote data center.

Firewall

If using a firewall, make sure that nodes within a cluster can reach each other. See Configuring firewall port access.

Generally, when you have firewalls between machines, it is difficult to run JMX across a network and maintain security. This is because JMX connects on port 7199, handshakes, and then uses any port within the 1024+ range. Instead use SSH to execute commands remotely connect to JMX locally or use the DataStax OpsCenter.

Planning an Amazon EC2 cluster

For planning an Amazon cluster, see the latest AMI documentation.

Calculating usable disk capacity

About this task

To calculate how much data your Cassandra nodes can hold, calculate the usable disk capacity per node and then multiply that by the number of nodes in your cluster. Remember that in a production cluster, you will typically have your commit log and data directories on different disks.

Procedure

1. Start with the raw capacity of the physical disks:

   ```
   raw_capacity = disk_size * number_of_data_disks
   ```

2. Calculate the formatted disk space as follows:

   ```
   formatted_disk_space = (raw_capacity * 0.9)
   ```

   During normal operations, Cassandra routinely requires disk capacity for compaction and repair operations. For optimal performance and cluster health, DataStax recommends not filling your disks to capacity, but running at 50% to 80% capacity depending on the compaction strategy and size of the compactions.

3. Calculate the usable disk space accounting for file system formatting overhead (roughly 10 percent):

   ```
   usable_disk_space = formatted_disk_space * (0.5 to 0.8)
   ```

Calculating user data size

About this task

As with all data storage systems, the size of your raw data will be larger once it is loaded into Cassandra due to storage overhead. On average, raw data is about two times larger on disk after it is loaded into the database, but could be much smaller or larger depending on the characteristics of your data and tables. The following calculations account for data persisted to disk, not for data stored in memory.

Procedure

- Determine column overhead:

  ```
  regular_total_column_size = column_name_size + column_value_size + 15

  counter - expiring_total_column_size = column_name_size + column_value_size
   + 23
  ```

  Every column in Cassandra incurs 15 bytes of overhead. Since each row in a table can have different column names as well as differing numbers of columns, metadata is stored for each column. For counter columns and expiring columns, you should add an additional 8 bytes (23 bytes total).

- Account for row overhead.

  Every row in Cassandra incurs 23 bytes of overhead.

- Estimate primary key index size:

  ```
  primary_key_index = number_of_rows * ( 32 + average_key_size )
  ```

  Every table also maintains a partition index. This estimation is in bytes.

- Determine replication overhead:

  ```
  replication_overhead = total_data_size * ( replication_factor - 1 )
  ```

  The replication factor plays a role in how much disk capacity is used. For a replication factor of 1, there is no overhead for replicas (as only one copy of data is stored in the cluster). If replication factor is greater than 1, then your total data storage requirement will include replication overhead.

Anti-patterns in Cassandra

Implementation or design patterns that are ineffective and/or counterproductive in Cassandra production installations. Correct patterns are suggested in most cases.

Storage area network

SAN storage is **not** recommended for on-premises deployments.

Note: Storage in clouds works very differently. Customers should contact DataStax for questions.

Although used frequently in Enterprise IT environments, SAN storage has proven to be a difficult and expensive architecture to use with distributed databases for a variety of reasons, including:

- SAN ROI does not scale along with that of Cassandra, in terms of capital expenses and engineering resources.
- In distributed architectures, SAN generally introduces a bottleneck and single point of failure because Cassandra's IO frequently surpasses the array controller's ability to keep up.
- External storage, even when used with a high-speed network and SSD, adds latency for all operations.
- Heap pressure is increased because pending I/O operations take longer.
- When the SAN transport shares operations with internal and external Cassandra traffic, it can saturate the network and lead to network availability problems.

Taken together these factors can create problems that are difficult to resolve in production. In particular, new users deploying Cassandra with SAN must first develop adequate methods and allocate sufficient engineering resources to identify these issues before they become a problem in production. For example, methods are needed for all key scaling factors, such as operational rates and SAN fiber saturation.

Impact of Shared Storage on Cassandra details metrics on how severely performance can be affected.

Network attached storage

Storing SSTables on a network attached storage (NAS) device is not recommended. Using a NAS device often results in network related bottlenecks resulting from high levels of I/O wait time on both reads and writes. The causes of these bottlenecks include:

- Router latency.
- The Network Interface Card (NIC) in the node.
- The NIC in the NAS device.

If you are required to use NAS for your environment, please contact a technical resource from DataStax for assistance.

Shared network file systems

Shared network file systems (NFS) have the same limitations as NAS. The temptation with NFS implementations is to place all SSTables in a node into one NFS. Doing this deprecates one of Cassandra's strongest features: No Single Point of Failure (SPOF). When all SSTables from all nodes are stored onto a single NFS, the NFS becomes a SPOF. To best use Cassandra, avoid using NFS.

Excessive heap space size

DataStax recommends using the default heap space size for most use cases. Exceeding this size can impair the Java virtual machine's (JVM) ability to perform fluid garbage collections (GC). The following table shows a comparison of heap space performances reported by a Cassandra user:

Heap	CPU utilization	Queries per second	Latency
40 GB	50%	750	1 second

Heap	CPU utilization	Queries per second	Latency
8 GB	5%	8500 (not maxed out)	10 ms

For information on heap sizing, see Tuning Java resources.

Cassandra's rack feature

This information applies only to single-token architecture, not to virtual nodes.

Defining one rack for the entire cluster is the simplest and most common implementation. Multiple racks should be avoided for the following reasons:

- Most users tend to ignore or forget rack requirements that racks should be organized in an alternating order. This order allows the data to get distributed safely and appropriately.
- Many users are not using the rack information effectively. For example, setting up with as many racks as nodes (or similar non-beneficial scenarios).
- Expanding a cluster when using racks can be tedious. The procedure typically involves several node moves and must ensure that racks are distributing data correctly and evenly. When clusters need immediate expansion, racks should be the last concern.

To use racks correctly:

- Use the same number of nodes in each rack.
- Use one rack and place the nodes in different racks in an alternating pattern. This allows you to still get the benefits of Cassandra's rack feature, and allows for quick and fully functional cluster expansions. Once the cluster is stable, you can swap nodes and make the appropriate moves to ensure that nodes are placed in the ring in an alternating fashion with respect to the racks.

Also see About Replication in Cassandra in the Cassandra 1.1 documentation.

SELECT ... IN or index lookups

SELECT ... IN and index lookups (formerly secondary indexes) should be avoided except for specific scenarios. See *When not to use IN* in SELECT and *When not to use an index* in Indexing in *CQL for Cassandra 2.0*.

Using the Byte Ordered Partitioner

The Byte Ordered Partitioner (BOP) is not recommended.

Use virtual nodes (vnodes) and use either the Murmur3Partitioner (default) or the RandomPartitioner. Vnodes allow each node to own a large number of small ranges distributed throughout the cluster. Using vnodes saves you the effort of generating tokens and assigning tokens to your nodes. If not using vnodes, these partitioners are recommended because all writes occur on the hash of the key and are therefore spread out throughout the ring amongst tokens range. These partitioners ensure that your cluster evenly distributes data by placing the key at the correct token using the key's hash value. Even if data becomes stale and needs to be deleted, this ensures that data removal also takes place while evenly distributing data around the cluster.

Reading before writing

Reads take time for every request, as they typically have multiple disk hits for uncached reads. In work flows requiring reads before writes, this small amount of latency can affect overall throughput. All write I/O in Cassandra is sequential so there is very little performance difference regardless of data size or key distribution.

Load balancers

Cassandra was designed to avoid the need for load balancers. Putting load balancers between Cassandra and Cassandra clients is harmful to performance, cost, availability, debugging, testing, and scaling. All high-level clients, such as Astyanax and pycassa, implement load balancing directly.

Insufficient testing

Be sure to test at scale and production loads. This the best way to ensure your system will function properly when your application goes live. The information you gather from testing is the best indicator of what throughput per node is needed for future expansion calculations.

To properly test, set up a small cluster with production loads. There will be a maximum throughput associated with each node count before the cluster can no longer increase performance. Take the maximum throughput at this cluster size and apply it linearly to a cluster size of a different size. Next extrapolate (graph) your results to predict the correct cluster sizes for required throughputs for your production cluster. This allows you to predict the correct cluster sizes for required throughputs in the future. The Netflix case study shows an excellent example for testing.

Lack of familiarity with Linux

Linux has a great collection of tools. Become familiar with the Linux built-in tools. It will help you greatly and ease operation and management costs in normal, routine functions. The essential list of tools and techniques to learn are:

- Parallel SSH and Cluster SSH: The pssh and cssh tools allow SSH access to multiple nodes. This is useful for inspections and cluster wide changes.
- Passwordless SSH: SSH authentication is carried out by using public and private keys. This allows SSH connections to easily hop from node to node without password access. In cases where more security is required, you can implement a password Jump Box and/or VPN.
- Useful common command-line tools include:

  - top: Provides an ongoing look at CPU processor activity in real time.
  - System performance tools: Tools such as iostat, mpstat, iftop, sar, lsof, netstat, htop, vmstat, and similar can collect and report a variety of metrics about the operation of the system.
  - vmstat: Reports information about processes, memory, paging, block I/O, traps, and CPU activity.
  - iftop: Shows a list of network connections. Connections are ordered by bandwidth usage, with the pair of hosts responsible for the most traffic at the top of list. This tool makes it easier to identify the hosts causing network congestion.

Running without the recommended settings

Be sure to use the recommended settings in the Cassandra documentation.

Also be sure to consult the Planning a Cassandra cluster deployment documentation, which discusses hardware and other recommendations before making your final hardware purchases.

Repeated DROP TABLE/CREATE TABLE with same table name

Customers who have previously dropped a table and recreated it with the same name reported various issues including:

- old data not being deleted, i.e. data from previous version of the table reappeared in queries
- read timeouts
- table queries returning

  ```
  TombstoneOverwhelmingException
  ```

- failed table compactions

The issue is that Cassandra keeps track of the table names internally even after the table is dropped. When a new table is created with the same name, Cassandra associates the table with the old reference ID and so continues to access the old SSTables on disk, and returns deleted rows (tombstones) during reads.

The preferred solution to solve this anti-pattern is to use the TRUNCATE command to remove data from a table if you wish to rewrite all the data at a later time.

More anti-patterns

For more about anti-patterns, visit Matt Dennis` slideshare.

Installing DataStax Community

Installing DataStax Community on RHEL-based systems

About this task

Use these steps to install Cassandra using Yum repositories on RHEL, CentOS, and Oracle Linux.

Note: To install on SUSE, use the Cassandra binary tarball distribution.

For a complete list of supported platforms, see DataStax Community – Supported Platforms.

Before you begin

- Yum Package Management application installed.
- Root or sudo access to the install machine.
- Latest version of Oracle Java SE Runtime Environment (JRE) 8 (recommended) or OpenJDK 7.
- Python 2.6+ (needed if installing OpsCenter).
- Java Native Access (JNA) is required for production installations (latest version recommended). See Installing the JNA on RHEL-based systems.

About this task

The packaged releases create a `cassandra` user. When starting Cassandra as a service, the service runs as this user.

Procedure

In a terminal window:

1. Check which version of Java is installed by running the following command:

```
$ java -version
```

It is recommended to use the latest version of Oracle Java 8 on all nodes. (Oracle Java 7 is also supported.)

See Installing the JRE on RHEL-based systems.

2. Add the DataStax Community repository to the `/etc/yum.repos.d/datastax.repo`:

```
[datastax]
name = DataStax Repo for Apache Cassandra
baseurl = http://rpm.datastax.com/community
enabled = 1
gpgcheck = 0
```

3. Install the packages:

```
$ sudo yum install dsc20-2.0.x-1 cassandra2.0.x-1
```

For example, to install DataStax Community 2.0.11:

```
$ sudo yum install dsc20-2.0.11-1 cassandra20-2.0.11-1
```

Check Download DataStax Community Edition on Planet Cassandra for the latest version.

The DataStax Community distribution of Cassandra is ready for configuration.

What to do next

- Initializing a multiple node cluster (single data center)
- Initializing a multiple node cluster (multiple data centers)
- Recommended production settings
- Installing OpsCenter
- Key components for configuring Cassandra
- Starting Cassandra as a service
- Package installation directories

Installing DataStax Community on Debian-based systems

About this task

Use these steps to install Cassandra using APT repositories on Debian and Ubuntu Linux.

For a complete list of supported platforms, see DataStax Community – Supported Platforms.

Before you begin

- Advanced Package Tool is installed.
- Root or sudo access to the install machine.
- Python 2.6+ (needed if installing OpsCenter).
- Latest version of Oracle Java SE Runtime Environment (JRE) 8 (recommended) or OpenJDK 7.
- Java Native Access (JNA) is required for production installations (latest version recommended). Installing the JNA on Debian-based systems.

Note: If you are using Ubuntu 10.04 LTS, you must update to JNA 3.4, as described in Installing the JNA from the JAR file.

About this task

The packaged releases create a `cassandra` user. When starting Cassandra as a service, the service runs as this user.

Procedure

In a terminal window:

1. Check which version of Java is installed by running the following command:

```
$ java -version
```

It is recommended to use the latest version of Oracle Java 8 on all nodes. (Oracle Java 7 is also supported.)

See Installing the JRE on RHEL-based systems.

2. Add the DataStax Community repository to the `/etc/apt/sources.list.d/cassandra.sources.list`

```
$ echo "deb http://debian.datastax.com/community stable main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
```

3. Debian systems only:

   a) In `/etc/apt/sources.list`, find the line that describes your source repository for Debian and add `contrib non-free` to the end of the line. For example:

      ```
      deb http://some.debian.mirror/debian/ $distro main contrib non-free
      ```

      This allows installation of the Oracle JVM instead of the OpenJDK JVM.

   b) Save and close the file when you are done adding/editing your sources.

4. Add the DataStax repository key to your aptitude trusted keys.

   ```
   $ curl -L http://debian.datastax.com/debian/repo_key | sudo apt-key add -
   ```

5. Install the package. For example:

   ```
   $ sudo apt-get update
   $ sudo apt-get install dsc20=2.0.11-1 cassandra=2.0.11
   ```

   Check Download DataStax Community Edition on Planet Cassandra for the latest version.

   This installs the DataStax Community distribution of Cassandra. .

6. Because the Debian packages start the Cassandra service automatically, you must stop the server and clear the data:

   Doing this removes the default cluster_name (Test Cluster) from the system table. All nodes must use the same cluster name.

   ```
   $ sudo service cassandra stop
   $ sudo rm -rf /var/lib/cassandra/data/system/*
   ```

   The DataStax Community distribution of Cassandra is ready for configuration.

What to do next

- Initializing a multiple node cluster (single data center)
- Initializing a multiple node cluster (multiple data centers)
- Recommended production settings
- Installing OpsCenter
- Key components for configuring Cassandra
- Starting Cassandra as a service
- Package installation directories

Installing DataStax Community on any Linux-based platform

About this task

Use these steps to install Cassandra on all Linux-based platforms using a binary tarball.

Use this install for Mac OS X and platforms without package support, or if you do not have or want a root installation.

For a complete list of supported platforms, see DataStax Community – Supported Platforms.

Before you begin

- Latest version of Oracle Java SE Runtime Environment (JRE) 8 (recommended) or OpenJDK 7.
- Java Native Access (JNA) is required for production installations (latest version recommended). See Installing the JNA.
- Python 2.6+ (needed if installing OpsCenter).

- If you are using Ubuntu 10.04 LTS, you must update to JNA 3.4, as described in Installing the JNA from the JAR file.
- If you are using an older RHEL-based Linux distribution, such as CentOS-5, you may see the following error: `GLIBCXX_3.4.9 not found`. You must replace the Snappy compression/decompression library (`snappy-java-1.0.5.jar`) with the snappy-java-1.0.4.1.jar.

About this task

The binary tarball runs as a stand-alone process.

Procedure

In a terminal window:

1. Check which version of Java is installed by running the following command:

   ```
   $ java -version
   ```

   It is recommended to use the latest version of Oracle Java 8 on all nodes. (Oracle Java 7 is also supported.)

   See Installing the JRE on RHEL-based systems.

2. Download the DataStax Community:

   ```
   $ curl -L http://downloads.datastax.com/community/dsc-2.0.tar.gz | tar xz
   ```

   Check Download DataStax Community Edition on Planet Cassandra for the latest version.

   You can also download from Planet Cassandra.

3. Go to the install directory:

   ```
   $ cd dsc-cassandra-2.0.x
   ```

   The DataStax Community distribution of Cassandra is ready for configuration.

What to do next

- Initializing a multiple node cluster (single data center)
- Initializing a multiple node cluster (multiple data centers)
- Recommended production settings
- Installing OpsCenter
- Key components for configuring Cassandra
- Tarball installation directories
- Starting Cassandra as a stand-alone process

Installing DataStax Community on Windows systems

About this task

To install or uninstall DataStax Community on Windows systems, see Cassandra & DataStax Enterprise Essentials.

Installing prior releases of DataStax Community

About this task

How to install the same version as other nodes in your cluster.

Follow the install instructions in the relevant documentation and specify the specific version in the install command.

Installing the packages on RHEL-based platforms

Examples:

```
$ sudo yum install dsc20-2.0.2.1 cassandra20-2.0.2-1
$ sudo yum install dsc20-2.0.2.1 cassandra20-2.0.2-1 cassandra20-tools-2.0.2-1
 ## Optional
```

Installing the packages on Debian-based platforms

Examples:

```
$ sudo apt-get install dsc20=2.0.2-1 cassandra=2.0.2
$ sudo apt-get install dsc20=2.0.2-1 cassandra=2.0.2 cassandra-tools=2.0.2 ##
 Optional
```

Installing from the binary tarball

1. Download the tarball using the URL for the prior version.

   Cassandra 2.0.0 example:

   ```
   http://downloads.datastax.com/community/dsc-cassandra-2.0.0-bin.tar.gz
   ```

   Cassandra 2.0.3 example:

   ```
   http://downloads.datastax.com/community/dsc-cassandra-2.0.3-bin.tar.gz
   ```

2. Unpack the distribution. For example:

   ```
   $ tar -xzvf dsc-cassandra-2.0.3-bin.tar.gz
   ```

   The files are extracted into the dsc-cassandra-2.0.3 directory.

Uninstalling DataStax Community

About this task

Select the uninstall method for your type of installation.

Uninstalling Debian- and RHEL-based packages

Use this method when you have installed DataStax Community using APT or Yum.

1. Stop the Cassandra and DataStax Agent services:

   ```
   $ sudo service cassandra stop
   ```

2. Make sure all services are stopped:

```
$ ps auwx | grep cassandra
$ ps auwx | grep datastax-agent ## If the DataStax Agent was installed.
```

3. If services are still running, use the PID to kill the service:

```
$ sudo kill cassandra_pid
$ sudo kill datastax_agent_pid  ## If the DataStax Agent was installed.
```

4. Remove the library and log directories:

```
$ sudo rm -r /var/lib/cassandra
$ sudo rm -r /var/log/cassandra
```

5. Remove the installation directories:

   RHEL-based packages:

```
$ sudo yum remove "cassandra-*" "datastax-*" ## datastax-x required if the
 DataStax Agent was installed.
```

   Debian-based packages:

```
$ sudo apt-get purge "cassandra-*" "datastax-*"  ## datastax-x required if
 the DataStax Agent was installed..
```

Uninstalling the binary tarball

Use this method when you have installed DataStax Community using the binary tarball.

1. Stop the node:

```
$ ps auwx | grep cassandra
$ sudo  kill <pid>
```

2. Stop the DataStax Agent if installed:

```
$ sudo kill datastax_agent_pid
```

3. Remove the installation directory.

Installing on cloud providers

Installing a Cassandra cluster on Amazon EC2

About this task

For instructions on installing the DataStax AMI (Amazon Machine Image), see the latest AMI documentation.

Clearing the data for an AMI restart

About this task

Use these steps to clear the data for an Amazon Machine Image (AMI) restart. The logs for AMIs are stored in the same location as the data. To preserve the log files, you need to delete everything except the log directory.

Procedure

To clear the data from the **default** directories:

After stopping the service, run the following command:

```
$ sudo rm -rf /var/lib/cassandra/commitlog
$ sudo rm -rf /var/lib/cassandra/data
$ sudo rm -rf /var/lib/cassandra/saved_caches
```

Installing and deploying a Cassandra cluster using GoGrid

About this task

You can use GoGrid's 1-Button Deploy for installing and deploying a developer (3-node) or production (5-node) Cassandra cluster. Additional introductory documentation is available from GoGrid at:

- GoGrid Cassandra Wiki
- Getting Started

The 1-Button Deploy of Cassandra does the following:

- Installs the latest version of Cassandra on X-Large SSD Cloud Servers running Debian 7.2.
- Installs OpsCenter.
- Installs Oracle JDK 7.
- Installs Python Driver.
- Enables the Firewall Service - All services are blocked except SSH (22) and ping for public traffic.
- Deploys Cassandra using virtual nodes (vnodes).

Procedure

1. Register with GoGrid.
2. Fill out the registration form and complete the account verification.
3. Access the management console with the login credentials you received in your email.

   Your cluster automatically starts deploying. A green status indicator shows that a server is up and running.

   Hover over any item to view its details or right-click to display a context menu.

COMMENT
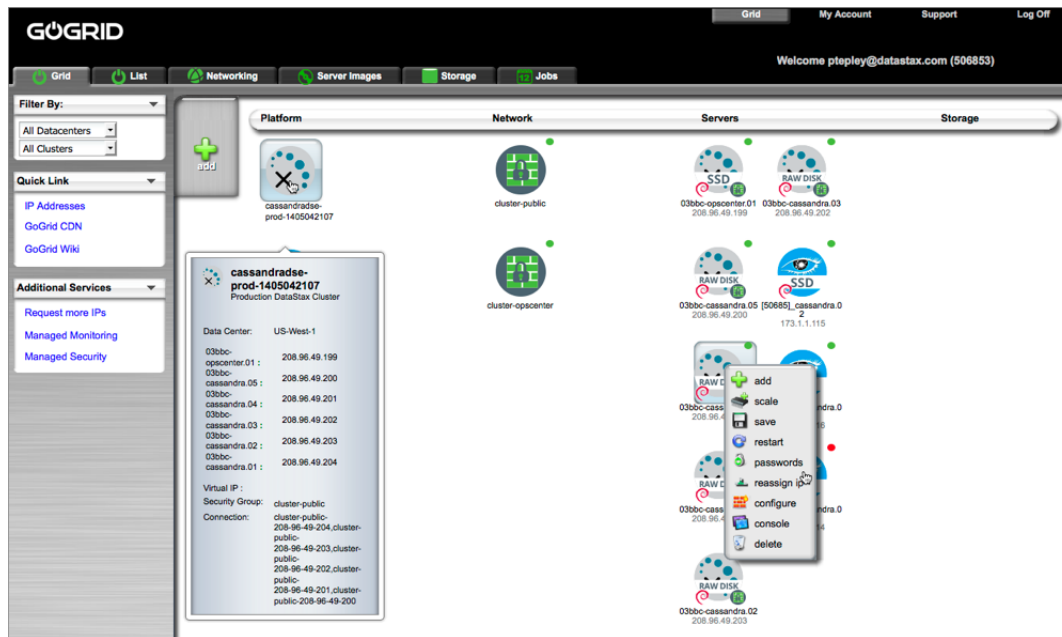
4. Login to one of the servers and validate that the servers are configured and communicating:

   Note: You can login to any member of the cluster either with SSH, a third-party client (like PuTTY), or through the GoGrid Console service.

   a) To find your server credentials, right-click the server and select **Passwords**.

   b) From your secure connection client, login to the server with the proper credentials. For example from SSH:

   ```
   $ ssh root@ip_address
   ```

   c) Validate that the cluster is running:

   ```
   $ nodestool status
   ```

   Each node should be listed and it's status and state should be UN (Up Normal):

   ```
   Datacenter: datacenter1
   =======================
   Status=Up/Down |/ State=Normal/Leaving/Joining/Moving
   -- Address        Load        Tokens Owns (effective)   Host ID
                     Rack
   UN 10.110.94.2   71.46 KB    256     65.9%              3ed072d6-49cb-4713-
   bd55-ea4de25576a9    rack1
   UN 10.110.94.5   40.91 KB    256     66.7%
    d5d982bc-6e30-40a0-8fe7-e46d6622c1d5    rack1
   UN 10.110.94.4   73.33 KB    256     67.4%              f6c3bf08-
   d9e5-43c8-85fa-5420db785052    rack1
   ```

What to do next

The following provides information about using and configuring Cassandra, OpsCenter, GoGrid, and the Cassandra Query Language (CQL):

- About Apache Cassandra
- OpsCenter documentation
- GoGrid Documentation
- CQL for Cassandra 2.0

Installing the Oracle JRE and the JNA

Installing Oracle JRE on RHEL-based Systems

About this task

Configure your operating system to use the latest version of Oracle Java SE Runtime Environment 8 or OpenJDK 7.

Note: After installing the JRE, you may need to set *JAVA_HOME* to your profile:

For shell or bash: `export JAVA_HOME=path_to_java_home`

For csh (C shell): `setenv JAVA_HOME=path_to_java_home`

Procedure

1. Check which version of the JRE your system is using:

   ```
   $ java -version
   ```

   If Oracle Java is used, the results should look like:

   ```
   java version "1.8.0_45"
   Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
   Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
   ```

2. If necessary, go to Oracle Java SE Downloads, accept the license agreement, and download the installer for your distribution.

   Note: If installing the Oracle JRE in a cloud environment, accept the license agreement, download the installer to your local client, and then use `scp` (secure copy) to transfer the file to your cloud machines.

3. From the directory where you downloaded the package, run the install:

   ```
   $ sudo rpm -ivh jre-8uversion-linux-x64.rpm
   ```

   The RPM installs the JRE into the `/usr/java/` directory.

4. Use the `alternatives` command to add a symbolic link to the Oracle JRE installation so that your system uses the Oracle JRE instead of the OpenJDK JRE:

   ```
   $ sudo alternatives --install /usr/bin/java java /usr/java/jre1.8.0_version/
   bin/java 200000
   ```

   If you have any problems, set the PATH and JAVA_HOME variables:

   ```
   export PATH="$PATH:/usr/java/latest/bin"
   set JAVA_HOME=/usr/java/latest
   ```

5. Make sure your system is now using the correct JRE. For example:

   ```
   $ java -version
   ```

   ```
   java version "1.8.0_45"
   Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
   Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
   ```

6. If the OpenJDK JRE is still being used, use the `alternatives` command to switch it. For example:

```
$ sudo alternatives --config java

There are 2 programs which provide java.

  Selection      Command
-----------------------------------------------------------
   1               /usr/lib/jvm/jre-1.7.0-openjdk.x86_64/bin/java
*+ 2               /usr/java/jre1.8.0_45/bin/java

Enter to keep the current selection [+ ], or type selection number:
```

Installing Oracle JRE on Debian or Ubuntu Systems

About this task

Configure your operating system to use the latest version of Oracle Java SE Runtime Environment 8 or OpenJDK 7.

Note: After installing the JRE, you may need to set *JAVA_HOME* to your profile:

For shell or bash: `export JAVA_HOME=path_to_java_home`

For csh (C shell): `setenv JAVA_HOME=path_to_java_home`

About this task

The Oracle Java Runtime Environment (JRE) has been removed from the official software repositories of Ubuntu and only provides a binary (`.bin`) version. You can get the JRE from the Java SE Downloads.

Procedure

1. Check which version of the JRE your system is using:

   ```
   $ java –version
   ```

   If Oracle Java is used, the results should look like:

   ```
    java version "1.8.0_45"
    Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
    Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
   ```

2. If necessary, go to Oracle Java SE Downloads, accept the license agreement, and download the installer for your distribution.

   Note: If installing the Oracle JRE in a cloud environment, accept the license agreement, download the installer to your local client, and then use `scp` (secure copy) to transfer the file to your cloud machines.

3. Make a directory for the JRE:

   ```
   $ sudo mkdir -p /usr/lib/jvm
   ```

4. Unpack the tarball and install the JRE:

   ```
   $ sudo tar zxvf jre-8u45-linux-x64.tar.gz -C /usr/lib/jvm
   ```

   The JRE files are installed into a directory called `/usr/lib/jvm/jre-8u_version`.

5. Tell the system that there's a new Java version available:

   ```
   $ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/
   jre1.8.0_version/bin/java" 1
   ```

If updating from a previous version that was removed manually, execute the above command twice, because you'll get an error message the first time.

6. Set the new JRE as the default:

```
$ sudo update-alternatives --set java /usr/lib/jvm/jre1.8.0_version/bin/java
```

7. Make sure your system is now using the correct JRE. For example:

```
$ java -version

 java version "1.8.0_45"
 Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
 Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
```

Installing the JNA on RHEL or CentOS Systems

About this task

Installing JNA can improve Cassandra memory usage. When installed and configured, Linux does not swap out the JVM, and thus avoids related performance issues. The latest version is recommended.

Before you begin

- Cassandra requires JNA 3.2.7 or later. Some Yum repositories may provide earlier versions.
- EPEL (Extra Packages for Enterprise Linux). See Installing EPEL.

Procedure

1. Install with the following command:

```
$ sudo yum install jna
```

2. If you can't install using Yum or it provides a version of the JNA earlier than 3.2.7, install as described in Installing the JNA from the JAR file.

Installing the JNA on Debian or Ubuntu Systems

About this task

Installing JNA can improve Cassandra memory usage. When installed and configured, Linux does not swap out the JVM, and thus avoids related performance issues. The latest version is recommended.

Procedure

Install the JNA with the following command:

```
$ sudo apt-get install libjna-java
```

For Ubuntu 10.04 LTS, update to JNA 3.4 as follows:

1. Download the jna.jar from https://github.com/twall/jna.
2. Remove older versions of the JNA from the /usr/share/java/ directory.
3. Place the new jna.jar file in /usr/share/java/ directory.
4. Create a symbolic link to the file:

```
ln -s /usr/share/java/jna.jar install_location/lib
```

Installing the JNA from the JAR file

About this task

Installing JNA can improve Cassandra memory usage. When installed and configured, Linux does not swap out the JVM, and thus avoids related performance issues. The latest version is recommended.

Procedure

1. Download the `jna.jar` or `jna-platform.jar` from https://github.com/twall/jna.
2. Add the JAR file to `install_location`/`lib` (or place it in the CLASSPATH).
3. Add the following lines in the `/etc/security/limits.conf` file for the user/group that runs Cassandra:

```
$USER soft memlock unlimited
$USER hard memlock unlimited
```

Recommended production settings

Recommendations for production environments; adjust them accordingly for your implementation.

Optimizing SSDs

For the majority of Linux distributions, SSDs are not configured optimally by default. The following steps ensures best practice settings for SSDs:

1. Ensure that the SysFS rotational flag is set to false (zero).

   This overrides any detection by the operating system to ensure the drive is considered an SSD.
2. Repeat for any block devices created from SSD storage, such as mdarrays.
3. Set the IO scheduler to either deadline or noop:
   - The noop scheduler is the right choice when the target block device is an array of SSDs behind a high-end IO controller that performs IO optimization.
   - The deadline scheduler optimizes requests to minimize IO latency. If in doubt, use the deadline scheduler.
4. Set the read-ahead value for the block device to 8KB.

   This setting tells the operating system not to read extra bytes, which can increase IO time and pollute the cache with bytes that weren't requested by the user.

For example, if the SSD is `/dev/sda`, in `/etc/rc.local`:

```
echo deadline > /sys/block/sda/queue/scheduler
#OR...
#echo noop > /sys/block/sda/queue/scheduler
echo 0 > /sys/class/block/sda/queue/rotational
echo 8 > /sys/class/block/sda/queue/read_ahead_kb
```

Disable zone_reclaim_mode on NUMA systems

The Linux kernel can be inconsistent in enabling/disabling zone_reclaim_mode. This can result in odd performance problems.

To ensure that zone_reclaim_mode is disabled:

```
echo 0 > /proc/sys/vm/zone_reclaim_mode
```

For more information, see Peculiar Linux kernel performance problem on NUMA systems.

User resource limits

You can view the current limits using the `ulimit -a` command. Although limits can also be temporarily set using this command, DataStax recommends making the changes permanent:

Packaged installs: Ensure that the following settings are included in the `/etc/security/limits.d/cassandra.conf` file:

```
cassandra - memlock unlimited
cassandra - nofile 100000
cassandra - nproc 32768
cassandra - as unlimited
```

Tarball installs: Ensure that the following settings are included in the `/etc/security/limits.conf` file:

```
* - memlock unlimited
* - nofile 100000
* - nproc 32768
* - as unlimited
```

If you run Cassandra as root, some Linux distributions such as Ubuntu, require setting the limits for root explicitly instead of using *:

```
root - memlock unlimited
root - nofile 100000
root - nproc 32768
root - as unlimited
```

For CentOS, RHEL, OEL systems, also set the nproc limits in `/etc/security/limits.d/90-nproc.conf`:

```
* - nproc 32768
```

For all installations, add the following line to `/etc/sysctl.conf`:

```
vm.max_map_count = 131072
```

To make the changes take effect, reboot the server or run the following command:

```
$ sudo sysctl -p
```

To confirm the limits are applied to the Cassandra process, run the following command where *pid* is the process ID of the currently running Cassandra process:

```
$ cat /proc/<pid>/limits
```

For more information, see Insufficient user resource limits errors.

Disable swap

You must disable swap entirely. Failure to do so can severely lower performance. Because Cassandra has multiple replicas and transparent failover, it is preferable for a replica to be killed immediately when memory is low rather than go into swap. This allows traffic to be immediately redirected to a functioning replica instead of continuing to hit the replica that has high latency due to swapping. If your system has a lot of DRAM, swapping still lowers performance significantly because the OS swaps out executable code so that more DRAM is available for caching disks.

If you insist on using swap, you can set `vm.swappiness=1`. This allows the kernel swap out the absolute least used parts.

```
$ sudo swapoff --all
```

To make this change permanent, remove all swap file entries from `/etc/fstab`.

For more information, see Nodes seem to freeze after some period of time.

Synchronize clocks

The clocks on all nodes should be synchronized. You can use NTP (Network Time Protocol) or other methods.

This is required because columns are only overwritten if the timestamp in the new version of the column is more recent than the existing column.

Optimum blockdev --setra settings for RAID

Typically, a readahead of 128 is recommended, especially on Amazon EC2 RAID0 devices.

Check to ensure setra is not set to 65536:

```
sudo blockdev --report /dev/<device>
```

To set setra:

```
sudo blockdev --setra 128 /dev/<device>
```

Java Virtual Machine

The latest 64-bit version of Oracle Java SE Runtime Environment 8 or OpenJDK 7.

Java Native Access

Java Native Access (JNA) is required for production installations.

Initializing a cluster

Initializing a multiple node cluster (single data center)

About this task

This topic contains information for deploying a Cassandra cluster with a single data center. If you're new to Cassandra, and haven't set up a cluster, see the Cassandra & DataStax Enterprise Essentials or 10 Minute Cassandra Walkthrough.

Before you begin

Each node must be correctly configured before starting the cluster. You must determine or perform the following before starting the cluster:

* A good understanding of how Cassandra works. Be sure to read at least Understanding the architecture, Data replication, and Cassandra's rack feature.
* Install Cassandra on each node.
* Choose a name for the cluster.
* Get the IP address of each node.
* Determine which nodes will be seed nodes. **Do not make all nodes seed nodes.** Please read Internode communications (gossip).
* Determine the snitch and replication strategy. The GossipingPropertyFileSnitch and NetworkTopologyStrategy are recommended for production environments.
* If using multiple data centers, determine a naming convention for each data center and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102. Choose the name carefully; renaming a data center is not possible.
* Other possible configuration settings are described in The cassandra.yaml configuration file and property files such as `cassandra-rackdc.properties`.

About this task

This example describes installing a 6 node cluster spanning 2 racks in a single data center. Each node is configured to use the GossipingPropertyFileSnitch and 256 virtual nodes (vnodes).

In Cassandra, the term data center is a grouping of nodes. Data center is synonymous with replication group, that is, a grouping of nodes configured together for replication purposes.

Procedure

1. Suppose you install Cassandra on these nodes:

```
node0 110.82.155.0 (seed1)
node1 110.82.155.1
node2 110.82.155.2
node3 110.82.156.3 (seed2)
node4 110.82.156.4
node5 110.82.156.5
```

   Note: It is a best practice to have more than one seed node per data center.

2. If you have a firewall running in your cluster, you must open certain ports for communication between the nodes. See Configuring firewall port access.

3. If Cassandra is running, you must stop the server and clear the data:

Doing this removes the default cluster_name (Test Cluster) from the system table. All nodes must use the same cluster name.

Package installations:

a) Stop Cassandra:

```
$ sudo service cassandra stop
```

b) Clear the data:

```
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

Tarball installations:

a) Stop Cassandra:

```
$ ps auwx | grep cassandra
$ sudo  kill pid
```

b) Clear the data:

```
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

4. Set the properties in the `cassandra.yaml` file for each node:

- Package installations: `/etc/cassandra/cassandra.yaml`
- Tarball installations: `install_location/conf/cassandra.yaml`

Note: After making any changes in the `cassandra.yaml` file, you must restart the node for the changes to take effect.

Properties to set:

- num_tokens: *recommended value: 256*
- -seeds: *internal IP address of each seed node*

  Seed nodes do not bootstrap, which is the process of a new node joining an existing cluster. For new clusters, the bootstrap process on seed nodes is skipped.
- listen_address:

  If not set, Cassandra asks the system for the local address, the one associated with its hostname. In some cases Cassandra doesn't produce the correct address and you must specify the listen_address.
- endpoint_snitch: *name of snitch* (See endpoint_snitch.) If you are changing snitches, see Switching snitches.
- auto_bootstrap: false (Add this setting **only** when initializing a fresh cluster with no data.)

Note: If the nodes in the cluster are identical in terms of disk layout, shared libraries, and so on, you can use the same copy of the `cassandra.yaml` file on all of them.

Example:

```
cluster_name: 'MyCassandraCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
         - seeds: "110.82.155.0,110.82.155.3"
listen_address:
rpc_address: 0.0.0.0
endpoint_snitch: GossipingPropertyFileSnitch
```

5. In the `cassandra-rackdc.properties` file, assign the data center and rack names you determined in the Prerequisites. For example:

```
# indicate the rack and dc for this node
dc=DC1
rack=RAC1
```

6. After you have installed and configured Cassandra on all nodes, start the seed nodes one at a time, and then start the rest of the nodes.

   Note: If the node has restarted because of automatic restart, you must first stop the node and clear the data directories, as described above.

   Package installations:

   ```
   $ sudo service cassandra start
   ```

   Tarball installations:

   ```
   $ cd install_location
   $ bin/cassandra
   ```

7. To check that the ring is up and running, run:

   Package installations:

   ```
   $ nodetool status
   ```

   Tarball installations:

   ```
   $ cd install_location
   $ bin/nodetool status
   ```

   Each node should be listed and it's status and state should be UN (Up Normal).

```
paul@ubuntu:~/cassandra-2.0.0$ bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens  Owns   Host ID                               Rack
UN  10.194.171.160  53.98 KB   256     0.8%   a9fa31c7-f3c0-44d1-b8e7-a2628867840c  rack1
UN  10.196.14.48    93.62 KB   256     9.9%   f5bb146c-db51-475c-a44f-9facf2f1ad6e  rack1
UN  10.196.14.239   83.98 KB   256     8.2%   b8e6748f-ec11-410d-c94f-b8e7d88a28e7  rack1
...
```

Initializing a multiple node cluster (multiple data centers)

About this task

This topic contains information for deploying a Cassandra cluster with multiple data centers. If you're new to Cassandra, and haven't set up a cluster, see the Cassandra & DataStax Enterprise Essentials or 10 Minute Cassandra Walkthrough.

This example describes installing a six node cluster spanning two data centers. Each node is configured to use the GossipingPropertyFileSnitch (multiple rack aware) and 256 virtual nodes (vnodes).

In Cassandra, the term data center is a grouping of nodes. Data center is synonymous with replication group, that is, a grouping of nodes configured together for replication purposes.

Before you begin

Each node must be correctly configured before starting the cluster. You must determine or perform the following before starting the cluster:

- A good understanding of how Cassandra works. Be sure to read at least Understanding the architecture, Data replication, and Cassandra's rack feature.
- Install Cassandra on each node.
- Choose a name for the cluster.
- Get the IP address of each node.
- Determine which nodes will be seed nodes. **Do not make all nodes seed nodes.** Please read Internode communications (gossip).
- Determine the snitch and replication strategy. The GossipingPropertyFileSnitch and NetworkTopologyStrategy are recommended for production environments.
- If using multiple data centers, determine a naming convention for each data center and rack, for example: DC1, DC2 or 100, 200 and RAC1, RAC2 or R101, R102. Choose the name carefully; renaming a data center is not possible.
- Other possible configuration settings are described in The cassandra.yaml configuration file and property files such as `cassandra-rackdc.properties`.

Procedure

1. Suppose you install Cassandra on these nodes:

```
node0 10.168.66.41 (seed1)
node1 10.176.43.66
node2 10.168.247.41
node3 10.176.170.59 (seed2)
node4 10.169.61.170
node5 10.169.30.138
```

   Note: It is a best practice to have more than one seed node per data center.

2. If you have a firewall running in your cluster, you must open certain ports for communication between the nodes. See Configuring firewall port access.

3. If Cassandra is running, you must stop the server and clear the data:

   Doing this removes the default cluster_name (Test Cluster) from the system table. All nodes must use the same cluster name.

   Package installations:

   a) Stop Cassandra:

   ```
   $ sudo service cassandra stop
   ```

   b) Clear the data:

   ```
   $ sudo rm -rf /var/lib/cassandra/data/system/*
   ```

   Tarball installations:

   a) Stop Cassandra:

   ```
   $ ps auwx | grep cassandra
   $ sudo  kill pid
   ```

   b) Clear the data:

   ```
   $ sudo rm -rf /var/lib/cassandra/data/system/*
   ```

4. Set the properties in the `cassandra.yaml` file for each node:

   - Package installations: `/etc/cassandra/cassandra.yaml`
   - Tarball installations: `install_location/conf/cassandra.yaml`

   Note: After making any changes in the `cassandra.yaml` file, you must restart the node for the changes to take effect.

   Properties to set:

- num_tokens: *recommended value: 256*
- -seeds: *internal IP address of each seed node*

  Seed nodes do not bootstrap, which is the process of a new node joining an existing cluster. For new clusters, the bootstrap process on seed nodes is skipped.

- listen_address:

  If not set, Cassandra asks the system for the local address, the one associated with its hostname. In some cases Cassandra doesn't produce the correct address and you must specify the listen_address.

- endpoint_snitch: *name of snitch* (See endpoint_snitch.) If you are changing snitches, see Switching snitches.
- auto_bootstrap: false (Add this setting **only** when initializing a fresh cluster with no data.)

Note: If the nodes in the cluster are identical in terms of disk layout, shared libraries, and so on, you can use the same copy of the `cassandra.yaml` file on all of them.

Example:

```
cluster_name: 'MyCassandraCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
        - seeds:  "10.168.66.41,10.176.170.59"
listen_address:
endpoint_snitch: GossipingPropertyFileSnitch
```

Note: Include at least one node from *each* data center.

5. In the `cassandra-rackdc.properties` file, assign the data center and rack names you determined in the Prerequisites. For example:

   Nodes 0 to 2

   ```
   # indicate the rack and dc for this node
   dc=DC1
   rack=RAC1
   ```

   Nodes 3 to 5

   ```
   # indicate the rack and dc for this node
   dc=DC2
   rack=RAC1
   ```

6. After you have installed and configured Cassandra on all nodes, start the seed nodes one at a time, and then start the rest of the nodes.

   Note: If the node has restarted because of automatic restart, you must first stop the node and clear the data directories, as described above.

   Package installations:

   ```
   $ sudo service cassandra start
   ```

   Tarball installations:

   ```
   $ cd install_location
   $ bin/cassandra
   ```

7. To check that the ring is up and running, run:

   Package installations:

   ```
   $ nodetool status
   ```

Tarball installations:

```
$ cd install_location
$ bin/nodetool status
```

Each node should be listed and it's status and state should be UN (Up Normal).

```
paul@ubuntu:~/cassandra-2.0.0$ bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address          Load       Tokens  Owns   Host ID                               Rack
UN  10.194.171.160   53.98 KB   256     0.8%   a9fa31c7-f3c0-44d1-b8e7-a2628867840c  rack1
UN  10.196.14.48     93.62 KB   256     9.9%   f5bb146c-db51-475c-a44f-9facf2f1ad6e  rack1
UN  10.196.14.239    83.98 KB   256     8.2%   b8e6748f-ec11-410d-c94f-b8e7d88a28e7  rack1
...
```

Security

Securing Cassandra

Cassandra provides these security features to the open source community.

- Client-to-node encryption

  Cassandra includes an optional, secure form of communication from a client machine to a database cluster. Client to server SSL ensures data in flight is not compromised and is securely transferred back/ forth from client machines.

- Authentication based on internally controlled login accounts/passwords

  Administrators can create users who can be authenticated to Cassandra database clusters using the CREATE USER command. Internally, Cassandra manages user accounts and access to the database cluster using passwords. User accounts may be altered and dropped using the Cassandra Query Language (CQL).

- Object permission management

  Once authenticated into a database cluster using either internal authentication, the next security issue to be tackled is permission management. What can the user do inside the database? Authorization capabilities for Cassandra use the familiar GRANT/REVOKE security paradigm to manage object permissions.

- Enabling JMX authentication

  The default settings for Cassandra make JMX accessible only from localhost. If you want to enable remote JMX connections, change the LOCAL_JMX setting in `cassandra-env.sh` and enable authentication and/or ssl. After you enable JMX authentication, ensure that tools that use JMX, such as nodetool and DataStax OpsCenter, are configured to use authentication.

SSL encryption

Client-to-node encryption

About this task

Client-to-node encryption protects data in flight from client machines to a database cluster using SSL (Secure Sockets Layer). It establishes a secure channel between the client and the coordinator node.

Before you begin

All nodes must have all the relevant SSL certificates on all nodes. See Preparing server certificates.

About this task

To enable client-to-node SSL, you must set the client_encryption_options in the `cassandra.yaml` file.

Procedure

On each node under client_encryption_options:

- Enable encryption.
- Set the appropriate paths to your `.keystore` and `.truststore` files.
- Provide the required passwords. The passwords must match the passwords used when generating the keystore and truststore.
- To enable client certificate authentication, set require_client_auth to true. (Available starting with Cassandra 1.2.3.)

Example

```
client_encryption_options:
enabled: true
keystore: conf/.keystore ## The path to your .keystore file
keystore_password: <keystore password> ## The password you used when
 generating the keystore.
truststore: conf/.truststore
truststore_password: <truststore password>
require_client_auth: <true or false>
```

Node-to-node encryption

About this task

Node-to-node encryption protects data transferred between nodes in a cluster, including gossip communications, using SSL (Secure Sockets Layer).

Before you begin

All nodes must have all the relevant SSL certificates on all nodes. See Preparing server certificates.

About this task

To enable node-to-node SSL, you must set the server_encryption_options in the `cassandra.yaml` file.

Procedure

On each node under sever_encryption_options:

- Enable internode_encryption.

  The available options are:

  - all
  - none
  - dc: Cassandra encrypts the traffic between the data centers.
  - rack: Cassandra encrypts the traffic between the racks.
- Set the appropriate paths to your `.keystore` and `.truststore` files.
- Provide the required passwords. The passwords must match the passwords used when generating the keystore and truststore.
- To enable client certificate authentication, set require_client_auth to true. (Available starting with Cassandra 1.2.3.)

Example

```
server_encryption_options:
    internode_encryption: <internode_option>
    keystore: resources/dse/conf/.keystore
    keystore_password: <keystore password>
    truststore: resources/dse/conf/.truststore
    truststore_password: <truststore password>
```

```
require_client_auth: <true or false>
```

Using cqlsh with SSL encryption

About this task

Using a `.cqlshrc` file, or `cqlshrc` file means you don't have to override the SSL_CERTFILE environmental variables every time.

Note: You cannot use cqlsh when client certificate authentication is enabled (require_client_auth=true).

Procedure

To run cqlsh with SSL encryption, create a `.cassandra/cqlshrc` file in your home or client program directory.

Sample files are available in the following directories:

- Cassandra package installations: `/etc/cassandra`
- Cassandra tarball installations: `install_location/conf`

Example

```
[authentication]
username = fred
password = !!bang!!$

[connection]
hostname = 127.0.0.1
port = 9042

[ssl]
certfile = ~/keys/cassandra.cert
validate = true ## Optional, true by default

[certfiles]  ## Optional section, overrides the default certfile in the [ssl]
 section
192.168.1.3 = ~/keys/cassandra01.cert
192.168.1.4 = ~/keys/cassandra02.cert
```

Note:

When validate is enabled, the host in the certificate is compared to the host of the machine that it is connected to. The SSL certificate must be provided either in the configuration file or as an environment variable. The environment variables (*SSL_CERTFILE* and *SSL_VALIDATE*) override any options set in this file.

Preparing server certificates

About this task

How to generate SSL certificates for client-to-node encryption or node-to-node encryption.

If you generate the certificates for one type of encryption, you do not need to generate them again for the other: the same certificates are used for both. All nodes must have all the relevant SSL certificates on all nodes. A keystore contains private keys. The truststore contains SSL certificates for each node and doesn't require signing by a trusted and recognized public certification authority.

Procedure

1. Generate the private and public key pair for the nodes of the cluster.

   A prompt for the new keystore and key password appears.
2. Leave key password the same as the keystore password.
3. Repeat steps 1 and 2 on each node using a different alias for each one.

   ```
   keytool -genkey -keyalg RSA -alias <cassandra_node0> -keystore .keystore
   ```
4. Export the public part of the certificate to a separate file and copy these certificates to all other nodes.

   ```
   keytool -export -alias cassandra -file cassandranode0.cer -
   keystore .keystore
   ```
5. Add the certificate of each node to the truststore of each node, so nodes can verify the identity of other nodes.

   ```
   keytool -import -v -trustcacerts -alias <cassandra_node0> -file
    <cassandra_node0>.cer -keystore .truststore
   keytool -import -v -trustcacerts -alias <cassandra_node1> -file
    <cassandra_node1>.cer -keystore .truststore
   . . .
   ```
6. Distribute the `.keystore` and `.truststore` files to all Cassandra nodes.
7. Make sure `.keystore` is readable only to the Cassandra daemon and not by any user of the system.

Adding new trusted users

About this task

How to add new users when client certificate authentication is enabled.

Before you begin

The client certificate authentication must be enabled (require_client_auth=true).

Procedure

1. Generate the certificate as described in Client-to-node encryption.
2. Import the user's certificate into every node's truststore using keytool:

   ```
   keytool -import -v -trustcacerts -alias <username> -file <certificate file>
    -keystore .truststore
   ```

Internal authentication

Internal authentication

Like object permission management using internal authorization, internal authentication is based on Cassandra-controlled login accounts and passwords. Internal authentication works for the following clients when you provide a user name and password to start up the client:

- Astyanax
- cassandra-cli
- cqlsh
- DataStax drivers - produced and certified by DataStax to work with Cassandra.
- Hector

- pycassa

Internal authentication stores usernames and bcrypt-hashed passwords in the system_auth.credentials table.

PasswordAuthenticator is an IAuthenticator implementation that you can use to configure Cassandra for internal authentication out-of-the-box.

Configuring authentication

About this task

To configure Cassandra to use internal authentication, first make a change to the cassandra.yaml file and increase the replication factor of the system_auth keyspace, as described in this procedure. Next, start up Cassandra using the default user name and password (cassandra/cassandra), and start cqlsh using the same credentials. Finally, use these CQL statements to set up user accounts to authorize users to access the database objects:

- ALTER USER
- CREATE USER
- DROP USER
- LIST USERS

Note: To configure authorization, see Configuring internal authorization.

Procedure

1. Change the authenticator option in the `cassandra.yaml` to PasswordAuthenticator.

   By default, the authenticator option is set to AllowAllAuthenticator.

   `authenticator: PasswordAuthenticator`

2. Increase the replication factor for the system_auth keyspace to N (number of nodes).

   If you use the default, 1, and the node with the lone replica goes down, you will not be able to log into the cluster because the system_auth keyspace was not replicated.

3. Restart the Cassandra client.

   The default superuser name and password that you use to start the client is stored in Cassandra.

   `<client startup string> -u cassandra -p cassandra`

4. Start cqlsh using the superuser name and password.

   `./cqlsh -u cassandra -p cassandra`

5. Create another superuser, not named cassandra. This step is optional but highly recommended.

6. Log in as that new superuser.

7. Change the cassandra user password to something long and incomprehensible, and then forget about it. It won't be used again.

8. Take away the cassandra user's superuser status.

9. Use the CQL statements listed previously to set up user accounts and then grant permissions to access the database objects.

Logging in using cqlsh

About this task

Typically, after configuring authentication, you log into cqlsh using the -u and -p options to the `cqlsh` command. To avoid having enter credentials every time you launch cqlsh, you can create a `cqlshrc` file

in the `.cassandra` directory, which is in your home directory. When present, this file passes default login information to cqlsh.

Procedure

1. Open a text editor and create a file that specifies a user name and password.

```
[authentication]
username = fred
password = !!bang!!$
```

2. Save the file in your `home/.cassandra` directory and name it cqlshrc.

3. Set permissions on the file.

To protect database login information, ensure that the file is secure from unauthorized access.

Note: Sample cqlshrc files are available in:

• Packaged installations

/usr/share/doc/dse-libcassandra

• Binary installations

install_location/conf

Internal authorization

Object permissions

You use familiar relational database GRANT/REVOKE paradigm to grant or revoke permissions to access Cassandra data. A superuser grants initial permissions, and subsequently a user may or may not be given the permission to grant/revoke permissions. Object permission management is based on internal authorization.

Read access to these system tables is implicitly given to every authenticated user because the tables are used by most Cassandra tools:

• system.schema_keyspace
• system.schema_columns
• system.schema_columnfamilies
• system.local
• system.peers

Configuring internal authorization

About this task

CassandraAuthorizer is one of many possible IAuthorizer implementations, and the one that stores permissions in the system_auth.permissions table to support all authorization-related CQL statements. Configuration consists mainly of changing the authorizer option in the cassandra.yaml to use the CassandraAuthorizer.

Note: To configure authentication, see Configuring authentication.

Procedure

1. In the `cassandra.yaml` file, comment out the default AllowAllAuthorizer and add the CassandraAuthorizer.

   ```
   authorizer: CassandraAuthorizer
   ```

   You can use any authenticator except AllowAll.

2. Configure the replication factor for the system_auth keyspace to increase the replication factor to a number greater than 1.

3. Adjust the validity period for permissions caching by setting the permissions_validity_in_ms option in the `cassandra.yaml` file.

   Alternatively, disable permission caching by setting this option to 0.

Results

CQL supports these authorization statements:

- GRANT
- LIST PERMISSIONS
- REVOKE

Configuring firewall port access

If you have a firewall running on the nodes in your Cassandra cluster, you must open up the following ports to allow communication between the nodes, including certain Cassandra ports. If this isn't done, when you start Cassandra on a node, the node acts as a standalone database server rather than joining the database cluster.

Table 1: Public ports

Port number	Description
22	SSH port
8888	OpsCenter website. The opscenterd daemon listens on this port for HTTP requests coming directly from the browser.

Table 2: Cassandra inter-node ports

Port number	Description
7000	Cassandra inter-node cluster communication.
7001	Cassandra SSL inter-node cluster communication.
7199	Cassandra JMX monitoring port.

Table 3: Cassandra client ports

Port number	Description
9042	Cassandra client port.

Port number	Description
9160	Cassandra client port (Thrift).

Table 4: Cassandra OpsCenter ports

Port number	Description
61620	OpsCenter monitoring port. The opscenterd daemon listens on this port for TCP traffic coming from the agent.
61621	OpsCenter agent port. The agents listen on this port for SSL traffic initiated by OpsCenter.

Enabling JMX authentication

About this task

The default settings for Cassandra make JMX accessible only from localhost. If you want to enable remote JMX connections, change the LOCAL_JMX setting in `cassandra-env.sh` and enable authentication and/or ssl. After you enable JMX authentication, ensure that tools that use JMX, such as nodetool and DataStax OpsCenter, are configured to use authentication.

To use JMX authentication for OpsCenter, follow the steps in Modifying OpsCenter cluster connections.

Procedure

1. Open the `cassandra-env.sh` file for editing and update or add these lines:

   ```
   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"
   JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.password.file=/etc/
   cassandra/jmxremote.password"
   ```

   If the LOCAL_JMX setting is in your file:

   ```
   LOCAL_JMX=no
   ```

2. Copy the `jmxremote.password.template` from `/jre_install_location/lib/management/` to `/etc/cassandra/` and rename it to `jmxremote.password`:

   ```
   cp /<jre_install_dir/lib/management/jmxremote.password.template /etc/
   cassandra/jmxremote.password
   ```

3. Change the ownership of `jmxremote.password` to the user you run cassandra with and change permission to read only:

   ```
   chown cassandra:cassandra /etc/cassandra/jmxremote.password
   chmod 400 /etc/cassandra/jmxremote.password
   ```

4. Edit `jmxremote.password` and add the user and password for JMX-compliant utilities:

   ```
   monitorRole QED
   controlRole R&D
   cassandra cassandrapassword
   ```

   Note: This cassandra user and cassandra password is just an example. Specify the user and password for your environment.

5. Add the cassandra user with read and write permission to `/jre_install_location/lib/management/jmxremote.access`:

```
    monitorRole readonly
    cassandra readwrite
    controlRole readwrite \
    create javax.management.monitor.,javax.management.timer. \
    unregister
```

6. Restart Cassandra.

7. Run nodetool with the cassandra user and password.

```
    $ nodetool status -u cassandra -pw cassandra
```

Results

If you run nodetool without user and password, you see an error similar to:

```
root@VM1 cassandra]# nodetool status
Exception in thread "main" java.lang.SecurityException: Authentication failed!
 Credentials required
at
 com.sun.jmx.remote.security.JMXPluggableAuthenticator.authenticationFailure(Unknown
 Source)
at com.sun.jmx.remote.security.JMXPluggableAuthenticator.authenticate(Unknown
 Source)
at sun.management.jmxremote.ConnectorBootstrap
$AccessFileCheckerAuthenticator.authenticate(Unknown Source)
at javax.management.remote.rmi.RMIServerImpl.doNewClient(Unknown Source)
at javax.management.remote.rmi.RMIServerImpl.newClient(Unknown Source)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
at java.lang.reflect.Method.invoke(Unknown Source)
at sun.rmi.server.UnicastServerRef.dispatch(Unknown Source)
at sun.rmi.transport.Transport$1.run(Unknown Source)
at sun.rmi.transport.Transport$1.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at sun.rmi.transport.Transport.serviceCall(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport.handleMessages(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run0(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
at java.lang.Thread.run(Unknown Source)
at sun.rmi.transport.StreamRemoteCall.exceptionReceivedFromServer(Unknown
 Source)
at sun.rmi.transport.StreamRemoteCall.executeCall(Unknown Source)
at sun.rmi.server.UnicastRef.invoke(Unknown Source)
at javax.management.remote.rmi.RMIServerImpl_Stub.newClient(Unknown Source)
at javax.management.remote.rmi.RMIConnector.getConnection(Unknown Source)
at javax.management.remote.rmi.RMIConnector.connect(Unknown Source)
at javax.management.remote.JMXConnectorFactory.connect(Unknown Source)
at org.apache.cassandra.tools.NodeProbe.connect(NodeProbe.java:146)
at org.apache.cassandra.tools.NodeProbe.<init>(NodeProbe.java:116)
at org.apache.cassandra.tools.NodeCmd.main(NodeCmd.java:1099)
```

Database internals

Managing data

Cassandra uses a storage structure similar to a Log-Structured Merge Tree, unlike a typical relational database that uses a B-Tree. Cassandra avoids reading before writing. Read-before-write, especially in a large distributed system, can produce stalls in read performance and other problems. For example, two clients read at the same time, one overwrites the row to make update A, and then the other overwrites the row to make update B, removing update A. Reading before writing also corrupts caches and increases IO requirements. To avoid a read-before-write condition, the storage engine groups inserts/updates to be made, and sequentially writes only the updated parts of a row in append mode. Cassandra never re-writes or re-reads existing data, and never overwrites the rows in place.

A log-structured engine that avoids overwrites and uses sequential IO to update data is essential for writing to hard disks (HDD) and solid-state disks (SSD). On HDD, writing randomly involves a higher number of seek operations than sequential writing. The seek penalty incurred can be substantial. Using sequential IO, and thereby avoiding write amplification and disk failure, Cassandra accommodates inexpensive, consumer SSDs extremely well.

Separate table directories

Cassandra provides fine-grained control of table storage on disk, writing tables to disk using separate table directories within each keyspace directory. Data files are stored using this directory and file naming format:

/var/lib/cassandra/data/ks1/cf1/ks1-cf1-hc-1-Data.db

The new file name format includes the keyspace name to distinguish which keyspace and table the file contains when streaming or bulk loading data. Cassandra creates a subdirectory for each table, which allows you to symlink a table to a chosen physical drive or data volume. This provides the capability to move very active tables to faster media, such as SSD's for better performance, and also divvy up tables across all attached storage devices for better I/O balance at the storage layer.

Cassandra storage basics

To manage and access data in Cassandra, it is important to understand how Casssandra stores data. The hinted handoff feature and Cassandra conformance and non-conformance to the ACID (atomic, consistent, isolated, durable) database properties are key concepts in this discussion. In Cassandra, consistency refers to how up-to-date and synchronized a row of data is on all of its replicas.

Client utilities and application programming interfaces (APIs) for developing applications for data storage and retrieval are available.

The write path to compaction

Cassandra processes data at several stages on the write path, starting with the immediate logging of a write and ending in compaction:

- Logging data in the commit log
- Writing data to the memtable
- Flushing data from the memtable

- Storing data on disk in SSTables
- Compaction

Logging writes and memtable storage

When a write occurs, Cassandra stores the data in a structure in memory, the memtable, and also appends writes to the commit log on disk, providing configurable durability. The commit log receives every write made to a Cassandra node, and these durable writes survive permanently even after power failure. The memtable is a write-back cache of data partitions that Cassandra looks up by key. The memtable stores writes until reaching a limit, and then is flushed.

Flushing data from the memtable

When memtable contents exceed a configurable threshold, the memtable data, which includes indexes, is put in a queue to be flushed to disk. You can configure the length of the queue by changing memtable_flush_queue_size in the cassandra.yaml. If the data to be flushed exceeds the queue size, Cassandra blocks writes until the next flush succeeds. You can manually flush a table using the `nodetool flush` command. Typically, before restarting nodes, flushing the memtable is recommended to reduce commit log replay time. To flush the data, Cassandra sorts memtables by token and then writes the data to disk sequentially.

Storing data on disk in SSTables

Data in the commit log is purged after its corresponding data in the memtable is flushed to an SSTable.
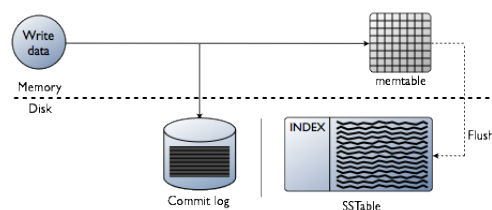
Memtables and SSTables are maintained per table. SSTables are immutable, not written to again after the memtable is flushed. Consequently, a partition is typically stored across multiple SSTable files.

For each SSTable, Cassandra creates these structures:

- Partition index

  A list of partition keys and the start position of rows in the data file (on disk)
- Partition summary (in memory)

  A sample of the partition index.
- Bloom filter

Compaction

Periodic compaction is essential to a healthy Cassandra database because Cassandra does not insert/update in place. As inserts/updates occur, instead of overwriting the rows, Cassandra writes a new timestamped version of the inserted or updated data in another SSTable. Cassandra manages the accumulation of SSTables on disk using compaction.

Cassandra also does not delete in place because the SSTable is immutable. Instead, Cassandra marks data to be deleted using a tombstone. Tombstones exist for a configured time period defined by the gc_grace_seconds value set on the table. During compaction, there is a temporary spike in disk space usage and disk I/O because the old and new SSTables co-exist. This diagram depicts the compaction process:
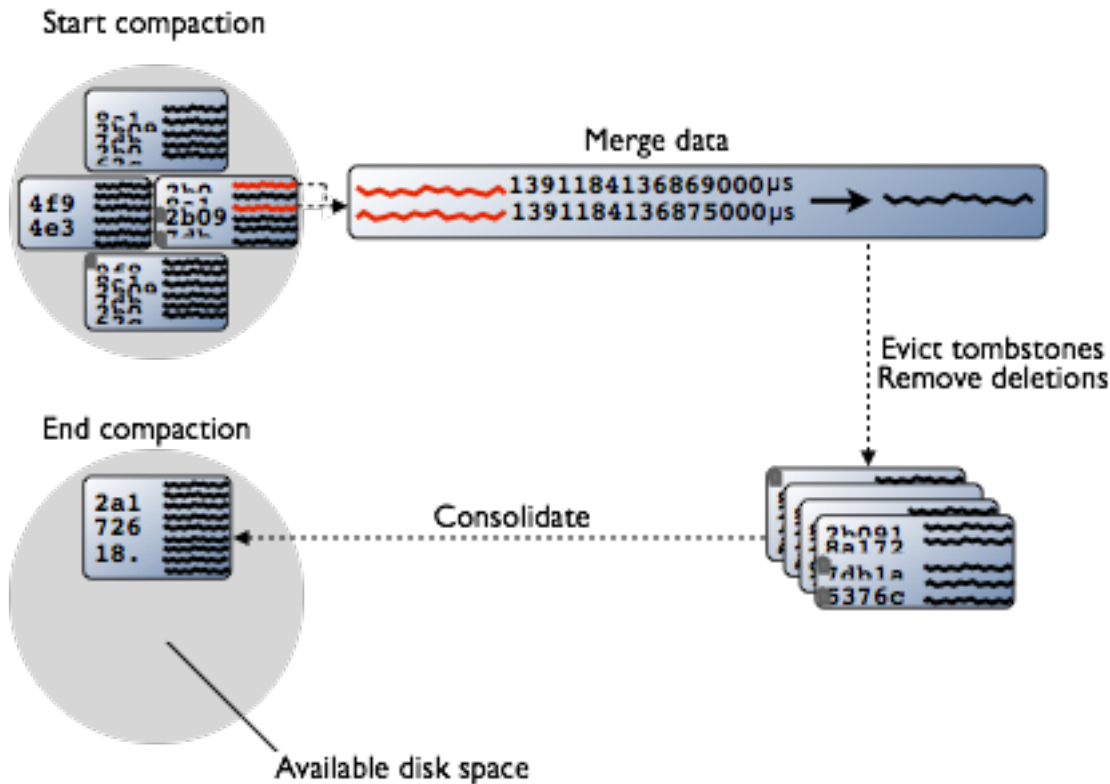
Compaction merges the data in each SSTable data by partition key, selecting the latest data for storage based on its timestamp. Cassandra can merge the data performantly, without random IO, because rows are sorted by partition key within each SSTable. After evicting tombstones and removing deleted data, columns, and rows, the compaction process consolidates SSTables into a single file. The old SSTable files are deleted as soon as any pending reads finish using the files. Disk space occupied by old SSTables becomes available for reuse.

Data input to SSTables is sorted to prevent random I/O during SSTable consolidation. After compaction, Cassandra uses the new consolidated SSTable instead of multiple old SSTables, fulfilling read requests more efficiently than before compaction. The old SSTable files are deleted as soon as any pending reads finish using the files. Disk space occupied by old SSTables becomes available for reuse.

Although no random I/O occurs, compaction can still be a fairly heavyweight operation. During compaction, there is a temporary spike in disk space usage and disk I/O because the old and new SSTables co-exist. To minimize deteriorating read speed, compaction runs in the background.

To lessen the impact of compaction on application requests, Cassandra performs these operations:

- Throttles compaction I/O to compaction_throughput_mb_per_sec (default 16MB/s).
- Requests that the operating system pull newly compacted partitions into the page cache when the key cache indicates that the compacted partition is hot for recent reads.

You can configure these types of compaction to run periodically: SizeTieredCompactionStrategy, DateTieredCompactionStrategy (Cassandra 2.0.11), and LeveledCompactionStrategy.

SizeTieredCompactionStrategy is designed for write-intensive workloads, DateTieredCompactionStrategy for time-series and expiring data, and LeveledCompactionStrategy for read-intensive workloads. You can manually start compaction using the `nodetool compact` command.

For more information about compaction strategies, see When to Use Leveled Compaction, Leveled Compaction in Apache Cassandra, and DateTieredCompactionStrategy.

How Cassandra stores indexes

Internally, a Cassandra index is a data partition.In the example of a music service, the playlists table includes an artist column and uses a compound primary key: id is the partition key and song_order is the clustering column.

```
CREATE TABLE playlists (
   id uuid,
   song_order int,
   . . .
   artist text,
PRIMARY KEY  (id, song_order ) );
```

As shown in the music service example, to filter the data based on the artist, create an index on artist. Cassandra uses the index to pull out the records in question. An attempt to filter the data before creating the index will fail because the operation would be very inefficient. A sequential scan across the entire playlists dataset would be required. After creating the artist index, Cassandra can filter the data in the playlists table by artist, such as Fu Manchu.

The partition is the unit of replication in Cassandra. In the music service example, partitions are distributed by hashing the playlist id and using the ring to locate the nodes that store the distributed data. Cassandra would generally store playlist information on different nodes, and to find all the songs by Fu Manchu, Cassandra would have to visit different nodes. To avoid these problems, each node indexes its own data.

This technique, however, does not guarantee trouble-free indexing, so know when and when not to use an index.

About index updates

As with relational databases, keeping indexes up to date is not free, so unnecessary indexes should be avoided. When a column is updated, the index is updated as well. If the old column value was still in the memtable, which typically occurs when updating a small set of rows repeatedly, Cassandra removes the corresponding obsolete index entry; otherwise, the old entry remains to be purged by compaction. If a read sees a stale index entry before compaction purges it, the reader thread invalidates it.

The write path of an update

Inserting a duplicate primary key is treated as an upsert. Eventually, the updates are streamed to disk using sequential I/O and stored in a new SSTable. During an update, Cassandra time-stamps and writes columns to disk using the write path. During the update, if multiple versions of the column exist in the memtable, Cassandra flushes only the newer version of the column to disk, as described in the Compaction section.

About deletes

The way Cassandra deletes data differs from the way a relational database deletes data. A relational database might spend time scanning through data looking for expired data and throwing it away or an administrator might have to partition expired data by month, for example, to clear it out faster. Data in a Cassandra column can have an optional expiration date called TTL (time to live). Use CQL to set the TTL in seconds for data. Cassandra marks TTL data with a tombstone after the requested amount of time has expired. Tombstones exist for a period of time defined by gc_grace_seconds. After data is marked with a tombstone, the data is automatically removed during the normal compaction process.

Facts about deleted data to keep in mind are:

- Cassandra does not immediately remove data marked for deletion from disk. The deletion occurs during compaction.
- If you use the sized-tiered or date-tiered compaction strategy, you can drop data immediately by manually starting the compaction process. Before doing so, understand the documented disadvantages of the process.
- A deleted column can reappear if you do not run node repair routinely.

Why deleted data can reappear

Marking data with a tombstone signals Cassandra to retry sending a delete request to a replica that was down at the time of delete. If the replica comes back up within the grace period of time, it eventually receives the delete request. However, if a node is down longer than the grace period, the node can miss the delete because the tombstone disappears after gc_grace_seconds. Cassandra always attempts to replay missed updates when the node comes back up again. After a failure, it is a best practice to run node repair to repair inconsistencies across all of the replicas when bringing a node back into the cluster. If the node doesn't come back within gc_grace,_seconds, remove the node, wipe it, and bootstrap it again.

About hinted handoff writes

Hinted handoff is a Cassandra feature that optimizes the cluster consistency process and anti-entropy when a replica-owning node is not available, due to network issues or other problems, to accept a replica from a successful write operation. Hinted handoff is not a process that guarantees successful write operations, except when a client application uses a consistency level of ANY. You enable or disable hinted handoff in the cassandra.yaml file.

How hinted handoff works

During a write operation, when hinted handoff is enabled and consistency can be met, the coordinator stores a hint about dead replicas in the local system.hints table under either of these conditions:

- A replica node for the row is known to be down ahead of time.
- A replica node does not respond to the write request.

When the cluster cannot meet the consistency level specified by the client, Cassandra does not store a hint.

A hint indicates that a write needs to be replayed to one or more unavailable nodes. The hint consists of:

- The location of the replica that is down
- Version metadata
- The actual data being written

By default, hints are saved for three hours after a replica fails because if the replica is down longer than that, it is likely permanently dead. You can configure this interval of time using the max_hint_window_in_ms property in the cassandra.yaml file. If the node recovers after the save time has elapsed, run a repair to re-replicate the data written during the down time.

After a node discovers from gossip that a node for which it holds hints has recovered, the node sends the data row corresponding to each hint to the target. Additionally, the node checks every ten minutes for any hints for writes that timed out during an outage too brief for the failure detector to notice through gossip.

For example, in a cluster of two nodes, A and B, having a replication factor (RF) of 1, each row is stored on one node. Suppose node A is down while we write row K to it with consistency level of one. The write fails because reads always reflect the most recent write when:

W + R > replication factor

where W is the number of nodes to block for writes and R is the number of nodes to block for reads. Cassandra does not write a hint to B and call the write good because Cassandra cannot read the data at any consistency level until A comes back up and B forwards the data to A.

In a cluster of three nodes, A (the coordinator), B, and C, each row is stored on two nodes in a keyspace having a replication factor of 2. Suppose node C goes down. The client writes row K to node A. The coordinator, replicates row K to node B, and writes the hint for downed node C to node A.
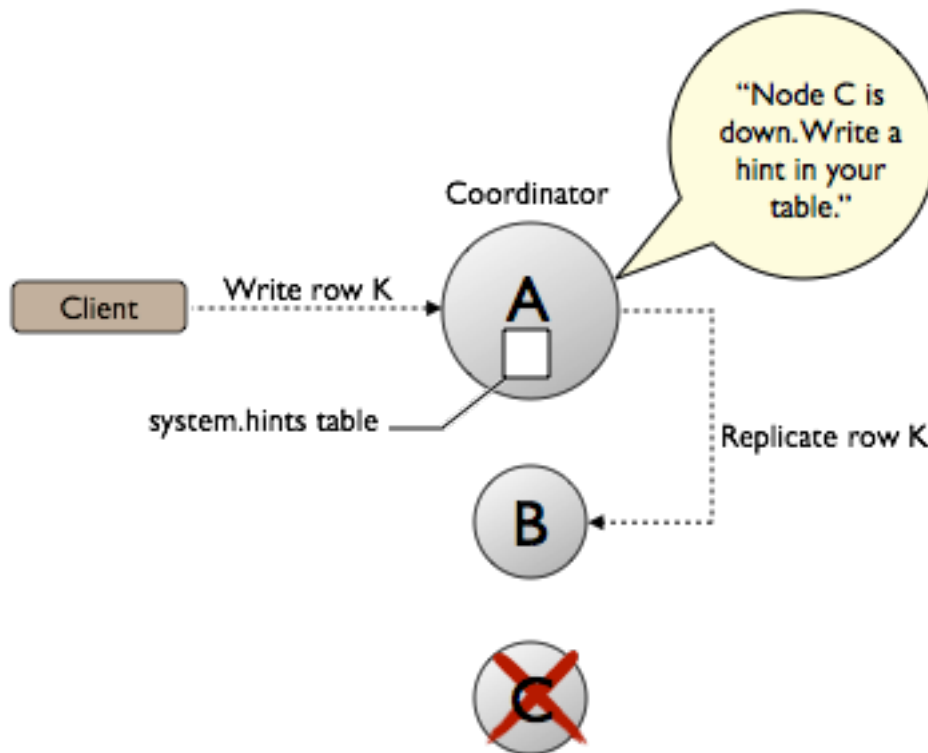
Cassandra, configured with a consistency level of ONE, calls the write good because Cassandra can read the data on node B. When node C comes back up, node A reacts to the hint by forwarding the data to node C. For more information about how hinted handoff works, see "Modern hinted handoff" by Jonathan Ellis.

Extreme write availability

For applications that want Cassandra to accept writes even when all the normal replicas are down, when not even consistency level ONE can be satisfied, Cassandra provides consistency level ANY. ANY guarantees that the write is durable and will be readable after an appropriate replica target becomes available and receives the hint replay.

Performance

By design, hinted handoff inherently forces Cassandra to continue performing the same number of writes even when the cluster is operating at reduced capacity. Pushing your cluster to maximum capacity with no allowance for failures is a bad idea.

Hinted handoff is designed to minimize the extra load on the cluster.

All hints for a given replica are stored under a single partition key, so replaying hints is a simple sequential read with minimal performance impact.

If a replica node is overloaded or unavailable, and the failure detector has not yet marked it down, then expect most or all writes to that node to fail after the timeout triggered by write_request_timeout_in_ms, which defaults to 10 seconds. During that time, Cassandra writes the hint when the timeout is reached.

If this happens on many nodes at once this could become substantial memory pressure on the coordinator. So the coordinator tracks how many hints it is currently writing, and if this number gets too high it will temporarily refuse writes (withOverloadedException) whose replicas include the misbehaving nodes.

Removal of hints

When removing a node from the cluster by decommissioning the node or by using the nodetool removenode command, Cassandra automatically removes hints targeting the node that no longer exists. Cassandra also removes hints for dropped tables.

Scheduling repair weekly

At first glance, it may appear that hinted handoff lets you safely get away without needing repair. This is only true if you never have hardware failure. Hardware failure has the following ramifications:

- Loss of the historical data necessary to tell the rest of the cluster exactly what data is missing.
- Loss of hints-not-yet-replayed from requests that the failed node coordinated.

About reads

Cassandra must combine results from the active memtable and potentially mutliple SSTables to satisfy a read. First, Cassandra checks the Bloom filter. Each SSTable has a Bloom filter associated with it that checks the probability of having any data for the requested partition in the SSTable before doing any disk I/O.

If the Bloom filter does not rule out the SSTable, Cassandra checks the partition key cache and takes one of these courses of action:

- If an index entry is found in the cache:

  - Cassandra goes to the compression offset map to find the compressed block having the data.
  - Fetches the compressed data on disk and returns the result set.
- If an index entry is not found in the cache:

  - Cassandra searches the partition summary to determine the approximate location on disk of the index entry.
  - Next, to fetch the index entry, Cassandra hits the disk for the first time, performing a single seek and a sequential read of columns (a range read) in the SSTable if the columns are contiguous.
  - Cassandra goes to the compression offset map to find the compressed block having the data.
  - Fetches the compressed data on disk and returns the result set.
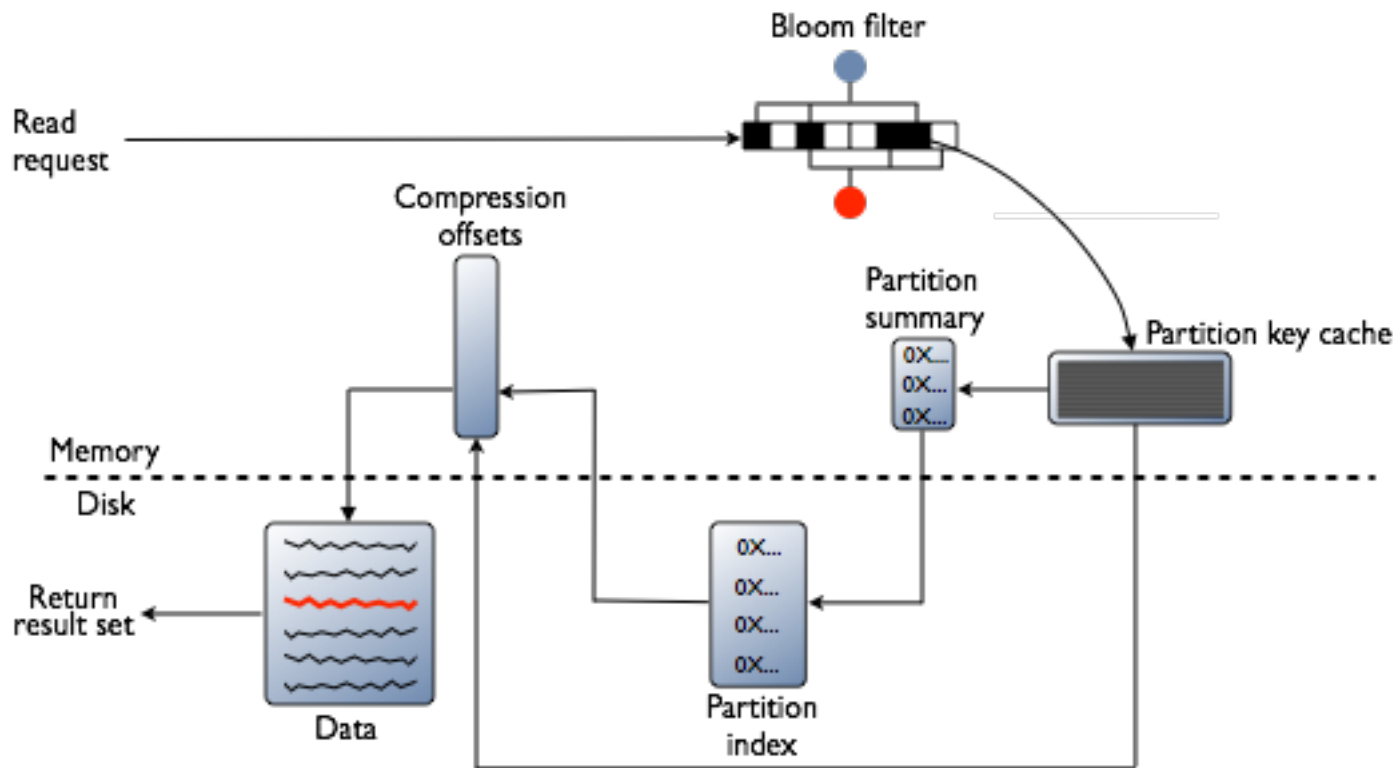
How off-heap components affect reads

To increase the data handling capacity per node, Cassandra keeps these components off-heap:

- Bloom filter
- Compression offsets map
- Partition summary

Of the components in memory, only the partition key cache is a fixed size. Other components grow as the data set grows.

Bloom filter

The Bloom filter grows to approximately 1-2 GB per billion partitions. In the extreme case, you can have one partition per row, so you can easily have billions of these entries on a single machine. The Bloom filter is tunable if you want to trade memory for performance.

Partition summary

By default, the partition summary is a sample of the partition index. You configure sample frequency by changing the index_interval property in the table definition, also if you want to trade memory for performance.

Compression offsets

The compression offset map grows to 1-3 GB per terabyte compressed. The more you compress data, the greater number of compressed blocks you have and the larger the compression offset table. Compression is enabled by default even though going through the compression offset map consumes CPU resources. Having compression enabled makes the page cache more effective, and typically, almost always pays off.

Reading from a partition

Within a partition, all rows are not equally expensive to query. The very beginning of the partition—the first rows, clustered by your key definition—is slightly less expensive to query because there is no need to consult the partition-level index.

How write patterns affect reads

The type of compaction strategy Cassandra performs on your data is configurable and can significantly affect read performance. Using the SizeTieredCompactionStrategy or DateTieredCompactionStrategy tends to cause data fragmentation when rows are frequently updated. The LeveledCompactionStrategy (LCS) was designed to prevent fragmentation under this condition. For more information about LCS, see the article, *Leveled Compaction in Apache Cassandra*.

How the row cache affects reads

Typical of any database, reads are fastest when the most in-demand data (or *hot* working set) fits into memory. Although all modern storage systems rely on some form of caching to allow for fast access to hot data, not all of them degrade gracefully when the cache capacity is exceeded and disk I/O is required. Cassandra's read performance benefits from built-in caching, shown in the following diagram.
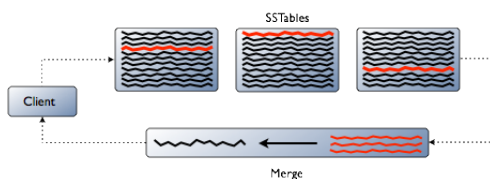
The red lines in the SSTables in this diagram are fragments of a row that Cassandra needs to combine to give the user the requested results. Cassandra caches the merged value, not the raw row fragments. That saves some CPU and disk I/O.

The row cache is not write-through. If a write comes in for the row, the cache for it is invalidated and is not be cached again until it is read again.

About transactions and concurrency control

Cassandra does not use RDBMS ACID transactions with rollback or locking mechanisms, but instead offers atomic, isolated, and durable transactions with eventual/tunable consistency that lets the user decide how strong or eventual they want each transaction's consistency to be.

- Atomic

  Everything in a transaction succeeds or the entire transaction is rolled back.
- Consistent

  A transaction cannot leave the database in an inconsistent state.
- Isolated

  Transactions cannot interfere with each other.
- Durable

  Completed transactions persist in the event of crashes or server failure.

As a non-relational database, Cassandra does not support joins or foreign keys, and consequently does not offer consistency in the ACID sense. For example, when moving money from account A to B the total in the accounts does not change. Cassandra supports atomicity and isolation at the row-level, but trades

transactional isolation and atomicity for high availability and fast write performance. Cassandra writes are durable.

Lightweight transactions

While durable transactions with eventual/tunable consistency is quite satisfactory for many use cases, situations do arise where more is needed. Lightweight transactions, also known as compare and set, that use linearizable consistency can probably fulfill those needs.

For example, two users attempting to create a unique user account in the same cluster could overwrite each other's work with neither user knowing about it. To avoid this situation, Cassandra introduces lightweight transactions (or 'compare and set').

Using and extending the Paxos consensus protocol (which allows a distributed system to agree on proposed data additions/modifications with a quorum-based algorithm, and without the need for any one 'master' database or two-phase commit), Cassandra now offers a way to ensure a transaction isolation level similar to the serializable level offered by RDBMS's. Extensions to CQL enable an easy way to carry out such operations.

A new IF clause has been introduced for both the INSERT and UPDATE commands that lets the user invoke lightweight transactions. For example, if a user wants to ensure an insert they are about to make into a new accounts table is unique for a new customer, they would use the IF NOT EXISTS clause:

```
INSERT INTO customer_account (customerID, customer_email)
VALUES ('LauraS', 'lauras@gmail.com')
IF NOT EXISTS;
```

DML modifications via UPDATE can also make use of the new IF clause by comparing one or more columns to various values:

```
UPDATE customer_account
SET    customer_email='laurass@gmail.com'
IF     customer_email='lauras@gmail.com';
```

Behind the scenes, Cassandra is making four round trips between a node proposing a lightweight transaction and any needed replicas in the cluster to ensure proper execution so performance is affected. Consequently, reserve lightweight transactions for those situations where they are absolutely necessary; Cassandra's normal eventual consistency can be used for everything else.

A SERIAL consistency level allows reading the current (and possibly uncommitted) state of data without proposing a new addition or update. If a SERIAL read finds an uncommitted transaction in progress, Cassandra commits the transaction as part of the read.

Atomicity

In Cassandra, a write is atomic at the partition-level, meaning inserting or updating columns in a row is treated as one write operation. Cassandra does not support transactions in the sense of bundling multiple row updates into one all-or-nothing operation. Nor does it roll back when a write succeeds on one replica, but fails on other replicas. It is possible in Cassandra to have a write operation report a failure to the client, but still actually persist the write to a replica.

For example, if using a write consistency level of QUORUM with a replication factor of 3, Cassandra will replicate the write to all nodes in the cluster and wait for acknowledgement from two nodes. If the write fails on one of the nodes but succeeds on the other, Cassandra reports a failure to replicate the write on that node. However, the replicated write that succeeds on the other node is not automatically rolled back.

Cassandra uses timestamps to determine the most recent update to a column. The timestamp is provided by the client application. The latest timestamp always wins when requesting data, so if multiple client sessions update the same columns in a row concurrently, the most recent update is the one that readers see.

Consistency

Cassandra 2.0 offers two types of consistency:

- Tunable consistency

  Availability and consistency can be tuned, and can be strong in the CAP sense--data is made consistent across all the nodes in a distributed database cluster.
- Linearizable consistency

  In ACID terms, linearizable consistency is a serial (immediate) isolation level for lightweight (compare-and-set, CAS) transactions.

In Cassandra, there are no locking or transactional dependencies when concurrently updating multiple rows or tables. Tuning availability and consistency always gives you partition tolerance. A user can pick and choose on a per operation basis how many nodes must receive a DML command or respond to a SELECT query.

Linearizable consistency is used in rare cases when a strong version of tunable consistency in a distributed, masterless Cassandra with quorum reads and writes is not enough. Such cases might be encountered when performing uninterrupted sequential operations or when producing the same results when running an operation concurrently or not. For example, an application that registers new accounts needs to ensure that only one user can claim a given account. The challenge is handling a race condition analogous to two threads attempting to make an insertion into a non-concurrent Map. Checking for the existence of the account before performing the insert in thread A does not guarantee that thread X will not insert the account between the check time and A's insert. Linearizable consistency meets these challenges.

Cassandra 2.0 uses the Paxos consensus protocol, which resembles 2-phase commit, to support linearizable consistency. All operations are quorum-based and updates will incur a performance hit, effectively a degradation to one-third of normal. For in-depth information about this new consistency level, see the article, *Lightweight transactions in Cassandra*.

To support linearizable consistency, a consistency level of SERIAL has been added to Cassandra. Additions to CQL have been made to support lightweight transactions.

Isolation

In early versions of Cassandra, it was possible to see partial updates in a row when one user was updating the row while another user was reading that same row. For example, if one user was writing a row with two thousand columns, another user could potentially read that same row and see some of the columns, but not all of them if the write was still in progress.

Full row-level isolation is in place, which means that writes to a row are isolated to the client performing the write and are not visible to any other user until they are complete.

Durability

Writes in Cassandra are durable. All writes to a replica node are recorded both in memory and in a commit log on disk before they are acknowledged as a success. If a crash or server failure occurs before the memtables are flushed to disk, the commit log is replayed on restart to recover any lost writes. In addition to the local durability (data immediately written to disk), the replication of data on other nodes strengthens durability.

You can manage the local durability to suit your needs for consistency using the commitlog_sync option in the `cassandra.yaml` file. Set the option to either periodic or batch.

About data consistency

Consistency refers to how up-to-date and synchronized a row of Cassandra data is on all of its replicas. Cassandra extends the concept of eventual consistency by offering tunable consistency. For any given read or write operation, the client application decides how consistent the requested data must be.

A tutorial in the CQL documentation compares consistency levels using cqlsh tracing.

Even at low consistency levels, Cassandra writes to all replicas of the partition key, even replicas in other data centers. The consistency level determines only the number of replicas that need to acknowledge the write success to the client application. Typically, a client specifies a consistency level that is less than the replication factor specified by the keyspace. This practice ensures that the coordinating server node reports the write successful even if some replicas are down or otherwise not responsive to the write.

The read consistency level specifies how many replicas must respond to a read request before returning data to the client application. Cassandra checks the specified number of replicas for data to satisfy the read request.

About built-in consistency repair features

You can use these built-in repair utilities to ensure that data remains consistent across replicas.

- Read repair
- Hinted handoff
- Anti-entropy node repair

Configuring data consistency

Consistency refers to how up-to-date and synchronized a row of Cassandra data is on all of its replicas. Cassandra extends the concept of eventual consistency by offering tunable consistency. For any given read or write operation, the client application decides how consistent the requested data must be.

Tunable consistency for client requests

Consistency levels in Cassandra can be configured to manage availability versus data accuracy. A tutorial in the CQL documentation compares consistency levels using cqlsh tracing. You can configure consistency on a cluster, data center, or individual I/O operation basis. Consistency among participating nodes can be set globally and also controlled on a per-operation basis (for example insert or update) using Cassandra's drivers and client libraries.

About write consistency

The consistency level determines the number of replicas on which the write must succeed before returning an acknowledgment to the client application.

Table 5: Write Consistency Levels

Level	Description	Usage
ALL	A write must be written to the commit log and memtable on all replica nodes in the cluster for that partition.	Provides the highest consistency and the lowest availability of any other level.
EACH_QUORUM	Strong consistency. A write must be written to the commit log and	Used in multiple data center clusters to strictly maintain consistency at the same level in each data center. For example,

Level	Description	Usage
	memtable on a quorum of replica nodes in *all* data centers.	choose this level if you want a read to fail when a data center is down and the QUORUM cannot be reached on that data center.
QUORUM	A write must be written to the commit log and memtable on a quorum of replica nodes.	Provides strong consistency if you can tolerate some level of failure.
LOCAL_QUORUM	Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in the same data center as the coordinator node. Avoids latency of inter-data center communication.	Used in multiple data center clusters with a rack-aware replica placement strategy, such as NetworkTopologyStrategy, and a properly configured snitch. Use to maintain consistency locally (within the single data center). Can be used with SimpleStrategy.
ONE	A write must be written to the commit log and memtable of at least one replica node.	Satisfies the needs of most users because consistency requirements are not stringent.
TWO	A write must be written to the commit log and memtable of at least two replica nodes.	Similar to ONE.
THREE	A write must be written to the commit log and memtable of at least three replica nodes.	Similar to TWO.
LOCAL_ONE	A write must be sent to, and successfully acknowledged by, at least one replica node in the local data center.	In a multiple data center clusters, a consistency level of ONE is often desirable, but cross-DC traffic is not. LOCAL_ONE accomplishes this. For security and quality reasons, you can use this consistency level in an offline datacenter to prevent automatic connection to online nodes in other data centers if an offline node goes down.
ANY	A write must be written to at least one node. If all replica nodes for the given partition key are down, the write can still succeed after a hinted handoff has been written. If all replica nodes are down at write time, an ANY write is not readable until the replica nodes for that partition have recovered.	Provides low latency and a guarantee that a write never fails. Delivers the lowest consistency and highest availability.
SERIAL	Achieves linearizable consistency for lightweight transactions by preventing unconditional updates.	You cannot configure this level as a normal consistency level, configured at the driver level using the consistency level field. You configure this level using the

Level	Description	Usage
		serial consistency field as part of the native protocol operation. See failure scenarios.
LOCAL_SERIAL	Same as SERIAL but confined to the data center. A write must be written conditionally to the commit log and memtable on a quorum of replica nodes in the same data center.	Same as SERIAL. Used for disaster recovery. See failure scenarios.

Even at low consistency levels, the write is still sent to all replicas for the written key, even replicas in other data centers. The consistency level just determines how many replicas are required to respond that they received the write.

SERIAL and LOCAL_SERIAL write failure scenarios

If one of three nodes is down, the Paxos commit fails under the following conditions:

- CQL query-configured consistency level of ALL
- Driver-configured serial consistency level of SERIAL
- Replication factor of 3

A WriteTimeout with a WriteType of CAS occurs and further reads do not see the write. If the node goes down in the middle of the operation instead of before the operation started, the write is committed, the value is written to the live nodes, and a WriteTimeout with a WriteType of SIMPLE occurs.

Under the same conditions, if two of the nodes are down at the beginning of the operation, the Paxos commit fails and nothing is committed. If the two nodes go down after the Paxos proposal is accepted, the write is committed to the remaining live nodes and written there, but a WriteTimeout with WriteType SIMPLE is returned.

About read consistency

The consistency level specifies how many replicas must respond to a read request before returning data to the client application.

Cassandra checks the specified number of replicas for data to satisfy the read request.

Table 6: Read Consistency Levels

Level	Description	Usage
ALL	Returns the record after all replicas have responded. The read operation will fail if a replica does not respond.	Provides the highest consistency of all levels and the lowest availability of all levels.
EACH_QUORUM	Returns the record once a quorum of replicas in each data center of the cluster has responded.	Same as LOCAL_QUORUM
QUORUM	Returns the record after a quorum of replicas has responded from any data center.	Ensures strong consistency if you can tolerate some level of failure.
LOCAL_QUORUM	Returns the record after a quorum of replicas in the current data	Used in multiple data center clusters with a rack-aware

Level	Description	Usage
	center as the coordinator node has reported. Avoids latency of inter-data center communication.	replica placement strategy (`NetworkTopologyStrategy`) and a properly configured snitch. Fails when using `SimpleStrategy`.
ONE	Returns a response from the closest replica, as determined by the snitch. By default, a read repair runs in the background to make the other replicas consistent.	Provides the highest availability of all the levels if you can tolerate a comparatively high probability of stale data being read. The replicas contacted for reads may not always have the most recent write.
TWO	Returns the most recent data from two of the closest replicas.	Similar to ONE.
THREE	Returns the most recent data from three of the closest replicas.	Similar to TWO.
LOCAL_ONE	Returns a response from the closest replica in the local data center.	Same usage as described in the table about write consistency levels.
SERIAL	Allows reading the current (and possibly uncommitted) state of data without proposing a new addition or update. If a SERIAL read finds an uncommitted transaction in progress, it will commit the transaction as part of the read. Similar to QUORUM.	To read the latest value of a column after a user has invoked a lightweight transaction to write to the column, use SERIAL. Cassandra then checks the inflight lightweight transaction for updates and, if found, returns the latest data.
LOCAL_SERIAL	Same as SERIAL, but confined to the data center. Similar to LOCAL_QUORUM.	Used to achieve linearizable consistency for lightweight transactions.

About the QUORUM level

The QUORUM level writes to the number of nodes that make up a quorum. A quorum is calculated, and then rounded down to a whole number, as follows:

(sum_of_replication_factors / 2) + 1

The sum of all the replication_factor settings for each data center is the sum_of_replication_factors.

For example, in a single data center cluster using a replication factor of 3, a quorum is 2 nodes#the cluster can tolerate 1 replica nodes down. Using a replication factor of 6, a quorum is 4#the cluster can tolerate 2 replica nodes down. In a two data center cluster where each data center has a replication factor of 3, a quorum is 4 nodes#the cluster can tolerate 2 replica nodes down. In a five data center cluster where each data center has a replication factor of 3, a quorum is 8 nodes.

If consistency is top priority, you can ensure that a read always reflects the most recent write by using the following formula:

(nodes_written + nodes_read) > replication_factor

For example, if your application is using the QUORUM consistency level for both write and read operations and you are using a replication factor of 3, then this ensures that 2 nodes are always written and 2 nodes

are always read. The combination of nodes written and read (4) being greater than the replication factor (3) ensures strong read consistency.

Configuring client consistency levels

You can use a new cqlsh command, CONSISTENCY, to set the consistency level for queries from the current cqlsh session. The WITH CONSISTENCY clause has been removed from CQL commands. You set the consistency level programmatically (at the driver level). For example, call QueryBuilder.insertInto with a setConsistencyLevel argument. The consistency level defaults to ONE for all write and read operations.

About built-in consistency repair features

You can use these built-in repair utilities to ensure that data remains consistent across replicas.

* Read repair
* Hinted handoff
* Anti-entropy node repair

Read requests

There are three types of read requests that a coordinator can send to a replica:

* A direct read request
* A digest request
* A background read repair request

The coordinator node contacts one replica node with a direct read request. Then the coordinator sends a digest request to a number of replicas determined by the consistency level specified by the client. The digest request checks the data in the replica node to make sure it is up to date. Then the coordinator sends a digest request to all remaining replicas. If any replica nodes have out of date data, a background read repair request is sent. Read repair requests ensure that the requested row is made consistent on all replicas.

For a digest request the coordinator first contacts the replicas specified by the consistency level. The coordinator sends these requests to the replicas that are currently responding the fastest. The nodes contacted respond with a digest of the requested data; if multiple nodes are contacted, the rows from each replica are compared in memory to see if they are consistent. If they are not, then the replica that has the most recent data (based on the timestamp) is used by the coordinator to forward the result back to the client.

To ensure that all replicas have the most recent version of frequently-read data, the coordinator also contacts and compares the data from all the remaining replicas that own the row in the background. If the replicas are inconsistent, the coordinator issues writes to the out-of-date replicas to update the row to the most recent values. This process is known as read repair. Read repair can be configured per table for non-QUORUM consistency levels (using read_repair_chance), and is enabled by default.

For illustrated examples of read requests, see the examples of read consistency levels.

Rapid read protection using speculative_retry

Rapid read protection allows Cassandra to still deliver read requests when the originally selected replica nodes are either down or taking too long to respond. If the table has been configured with the speculative_retry property, the coordinator node for the read request will retry the request with another replica node if the original replica node exceeds a configurable timeout value to complete the read request.

Recovering from replica node failure with rapid read protection

- Coordinator node
- Chosen node
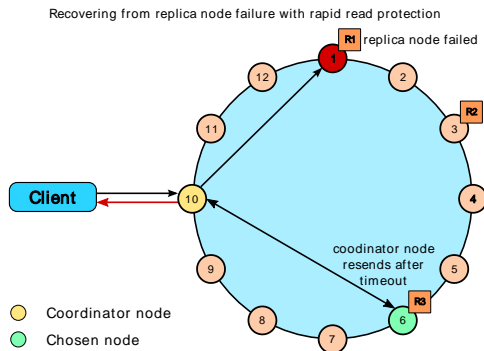
Examples of read consistency levels

The following diagrams show examples of read requests using these consistency levels:

- QUORUM in a single data center
- ONE in a single data center
- QUORUM in two data centers
- LOCAL_QUORUM in two data centers
- ONE in two data centers
- LOCAL_ONE in two data centers

Rapid read protection diagram shows how the speculative retry table property affects consistency.

A single data center cluster with a consistency level of QUORUM

In a single data center cluster with a replication factor of 3, and a read consistency level of QUORUM, 2 of the 3 replicas for the given row must respond to fulfill the read request. If the contacted replicas have different versions of the row, the replica with the most recent version will return the requested data. In the background, the third replica is checked for consistency with the first two, and if needed, a read repair is initiated for the out-of-date replicas.

Single data center cluster with 3 replica nodes and consistency set to QUORUM

- Coordinator node
- Chosen node
- —— Read response
- ----- Read repair
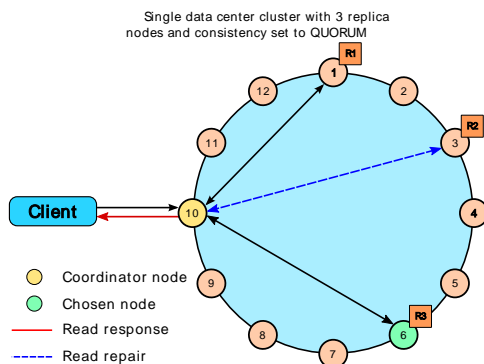
A single data center cluster with a consistency level of ONE

In a single data center cluster with a replication factor of 3, and a read consistency level of ONE, the closest replica for the given row is contacted to fulfill the read request. In the background a read repair is potentially initiated, based on the `read_repair_chance` setting of the table, for the other replicas.

Single data center cluster with 3 replica nodes and consistency set to ONE
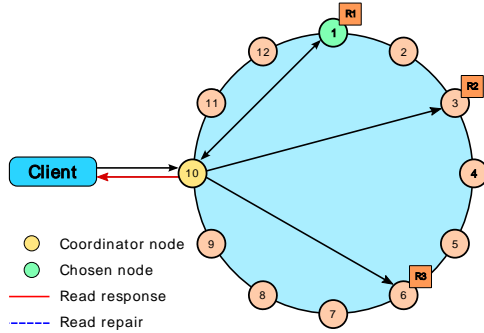
A two data center cluster with a consistency level of QUORUM

In a two data center cluster with a replication factor of 3, and a read consistency of QUORUM, 4 replicas for the given row must resond to fulfill the read request. The 4 replicas can be from any data center. In the background, the remaining replicas are checked for consistency with the first four, and if needed, a read repair is initiated for the out-of-date replicas.

Multiple data center cluster with 3 replica nodes and consistency set to QUORUM
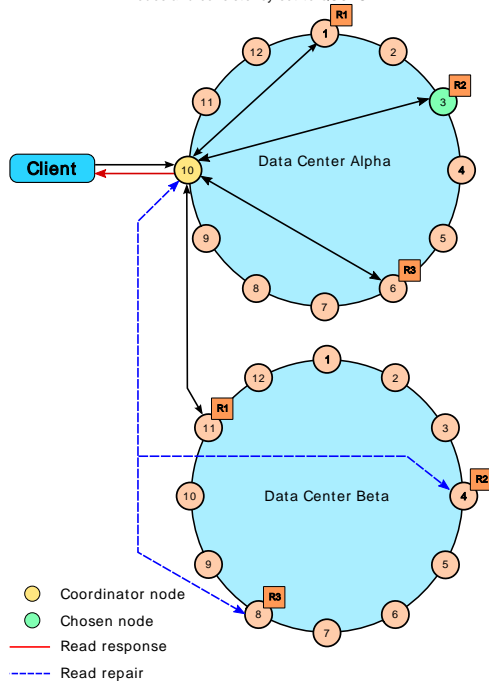
A two data center cluster with a consistency level of LOCAL_QUORUM

In a multiple data center cluster with a replication factor of 3, and a read consistency of LOCAL_QUORUM, 2 replicas in the same data center as the coordinator node for the given row must respond to fulfill the read request. In the background, the remaining replicas are checked for consistency with the first 2, and if needed, a read repair is initiated for the out-of-date replicas.

Multiple data center cluster with 3 replica nodes and consistency set to LOCAL_QUORUM
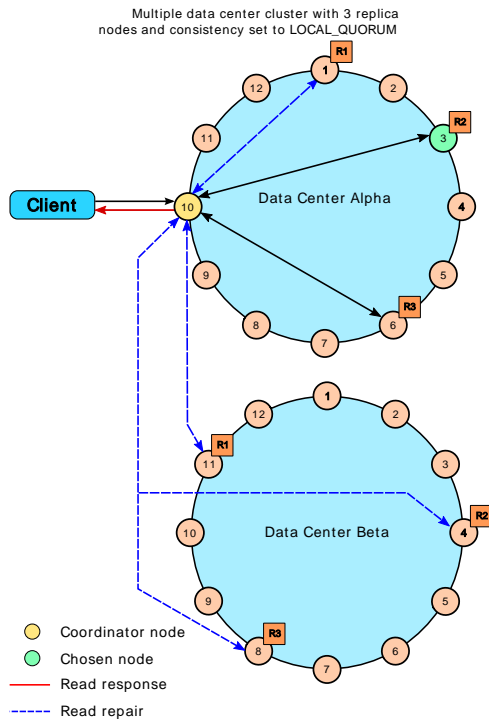
A two data center cluster with a consistency level of ONE

In a multiple data center cluster with a replication factor of 3, and a read consistency of ONE, the closest replica for the given row, regardless of data center, is contacted to fulfill the read request. In the background a read repair is potentially initiated, based on the read_repair_chance setting of the table, for the other replicas.

Multiple data center cluster with 3 replica nodes and consistency set to ONE
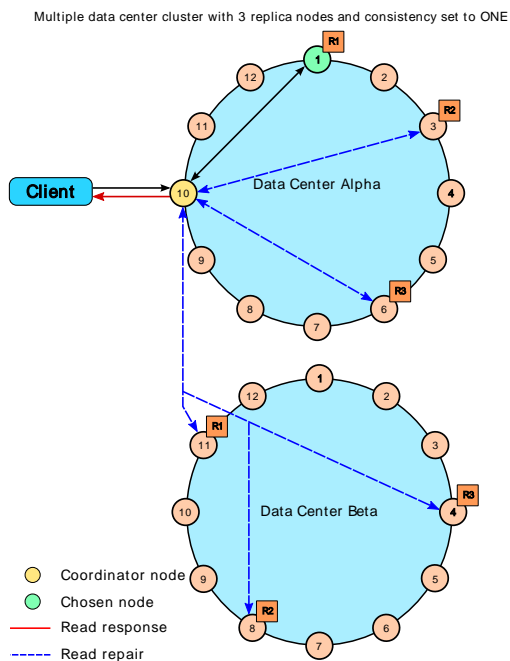
A two data center cluster with a consistency level of LOCAL_ONE

In a multiple data center cluster with a replication factor of 3, and a read consistency of LOCAL_ONE, the closest replica for the given row in the same data center as the coordinator node is contacted to fulfill the read request. In the background a read repair is potentially initiated, based on the read_repair_chance setting of the table, for the other replicas.

Multiple data center cluster with 3 replica nodes and consistency set to LOCAL_ONE
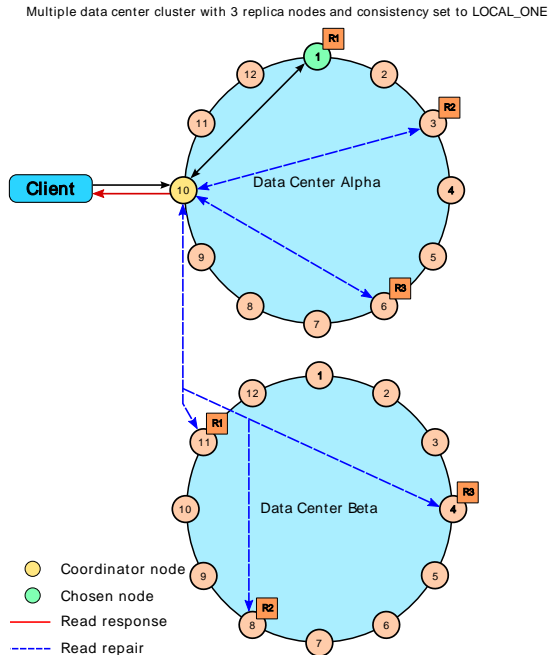
Write requests

The coordinator sends a write request to *all* replicas that own the row being written. As long as all replica nodes are up and available, they will get the write regardless of the consistency level specified by the client. The write consistency level determines how many replica nodes must respond with a success acknowledgment in order for the write to be considered successful. Success means that the data was written to the commit log and the memtable as described in About writes.

For example, in a single data center 10 node cluster with a replication factor of 3, an incoming write will go to all 3 nodes that own the requested row. If the write consistency level specified by the client is ONE, the first node to complete the write responds back to the coordinator, which then proxies the success message back to the client. A consistency level of ONE means that it is possible that 2 of the 3 replicas could miss the write if they happened to be down at the time the request was made. If a replica misses a write, Cassandra will make the row consistent later using one of its built-in repair mechanisms: hinted handoff, read repair, or anti-entropy node repair.

That node forwards the write to all replicas of that row. It responds back to the client once it receives a write acknowledgment from the number of nodes specified by the consistency level.

Single data center cluster with 3 replica nodes and consistency set to ONE
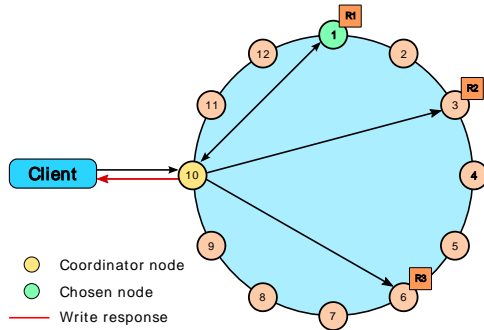
Multiple data center write requests

In multiple data center deployments, Cassandra optimizes write performance by choosing one coordinator node. The coordinator node contacted by the client application forwards the write request to each replica node in each all the data centers.

If using a consistency level of `LOCAL_ONE` or `LOCAL_QUORUM`, only the nodes in the same data center as the coordinator node must respond to the client request in order for the request to succeed. This way, geographical latency does not impact client request response times.

Multiple data center cluster with 3 replica nodes and consistency set to QUORUM
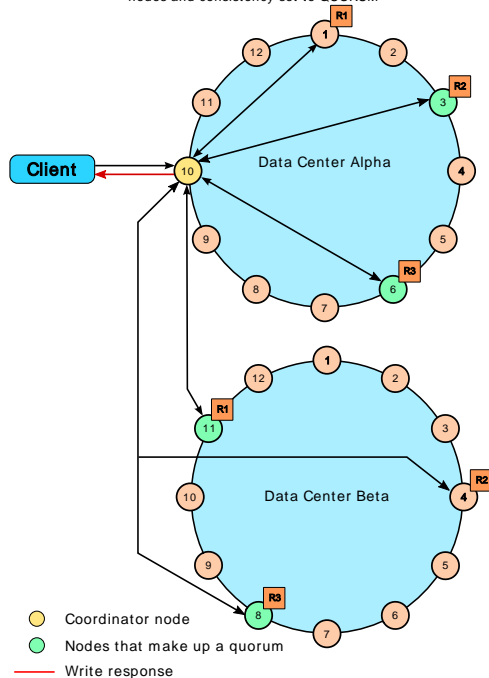
Configuration

The cassandra.yaml configuration file

The `cassandra.yaml` file is the main configuration file for Cassandra.

Important: After changing properties in the `cassandra.yaml` file, you must restart the node for the changes to take effect. It is located in the following directories:

* Cassandra package installations: `/etc/cassandra`
* Cassandra tarball installations: `install_location/conf`
* DataStax Enterprise package installations: `/etc/dse/cassandra`
* DataStax Enterprise tarball installations: `install_location/resources/cassandra/conf`

The configuration properties are grouped into the following sections:

* Quick start

    The minimal properties needed for configuring a cluster.
* Commonly used

    Properties most frequently used when configuring Cassandra.
* Performance tuning

    Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.
* Advanced

    Properties for advanced users or properties that are less commonly used.
* Security

    Server and client security settings.

Note: Values with [note] indicate default values that are defined internally, missing, commented out, or implementation depends on other properties in the `cassandra.yaml` file. Additionally, some commented out values may not match the actual default value; these values are recommended when changing from the default.

Quick start properties

The minimal properties needed for configuring a cluster.

Related information: Initializing a multiple node cluster (single data center) and Initializing a multiple node cluster (multiple data centers).

cluster_name

(Default: Test Cluster) The name of the cluster. This setting prevents nodes in one logical cluster from joining another. All nodes in a cluster must have the same value.

listen_address

(Default: localhost) The IP address or hostname that Cassandra binds to for listening to other Cassandra nodes. If you do not know if or have your networking properly configured, it's best to use your private IP addresses. If you want your nodes to be able to communicate, you must change this setting:

* Generally set to empty. If the node is properly configured (host name, name resolution, and so on), Cassandra uses `InetAddress.getLocalHost()` to get the local address from the system.

- For a single node cluster, you can use the default setting (localhost).
- If Cassandra can't find the correct address, you must specify the IP address or host name.
- Never specify 0.0.0.0; it is always wrong.

Default directories

If you have changed any of the default directories during installation, make sure you have root access and set these properties:

commitlog_directory

(Default: Packaged installs: `/var/lib/cassandra/commitlog`, Tarball installs: `install_location/data/commitlog`) The directory where the commit log is stored. For optimal write performance, it is recommended the commit log be on a separate disk partition (ideally, a separate physical device) from the data file directories. Using an HDD is acceptable because the commit log is append only.

data_file_directories

(Default:Packaged installs: `/var/lib/cassandra/data`, Tarball installs: `install_location/data/data`) The directory location where table data (SSTables) is stored. As a production best practice, use RAID 0 and SSDs.

saved_caches_directory

(Default: Packaged installs: `/var/lib/cassandra/saved_caches`, Tarball installs: `install_location/data/saved_caches`) The directory location where table key and row caches are stored.

Commonly used properties

Properties most frequently used when configuring Cassandra.

Before starting a node for the first time, you should carefully evaluate your requirements.

Common initialization properties

Note: Be sure to set the properties in the Quick start section as well.

commit_failure_policy

(Default: stop) Policy for commit disk failures:

- stop: Shut down gossip and Thrift, leaving the node effectively dead, but can be inspected using JMX.
- stop_commit: Shut down the commit log, letting writes collect but continuing to service reads (as in pre-2.0.5 Cassandra).
- ignore: Ignore fatal errors and let the batches fail.

disk_failure_policy

(Default: stop) Sets how Cassandra responds to disk failure.

- stop: Shuts down gossip and Thrift, leaving the node effectively dead, but it can still be inspected using JMX.
- stop_paranoid: Shut down gossip and Thrift even for single-SSTable errors.
- best_effort: Cassandra does its best in the event of disk errors. If it cannot write to a disk, the disk is blacklisted for writes and the node continues writing elsewhere. If Cassandra cannot read from the disk, those SSTables are marked unreadable, and the node continues serving data from readable SSTables. This means you will see obsolete data at consistency level of ONE.
- ignore: Use for upgrading. Cassandra acts as in versions prior to 1.2. Ignores fatal errors and lets the requests fail; all file system errors are logged but otherwise ignored. It is recommended using stop or best_effort.

Related information: Handling Disk Failures In Cassandra 1.2 blog and Recovering from a single disk failure using JBOD.

endpoint_snitch

(Default: `org.apache.cassandra.locator.SimpleSnitch`) Sets which snitch Cassandra uses for locating nodes and routing requests. It must be set to a class that implements `IEndpointSnitch`. For descriptions of the available snitches, see Snitches.

Note: The GossipingPropertyFileSnitch is recommended for production. It defines a node's data center and rack and uses gossip for propagating this information to other nodes.

rpc_address

(Default: localhost) The listen address for client connections. Valid values are:

- 0.0.0.0: Listens on all configured interfaces.
- IP address
- hostname
- unset: Resolves the address using the hostname configuration of the node. If left unset, the hostname must resolve to the IP address of this node using `/etc/hostname, /etc/hosts`, or DNS.

Related information: Network

seed_provider

The addresses of hosts deemed contact points. Cassandra nodes use the -seeds list to find each other and learn the topology of the ring.

- class_name (Default: `org.apache.cassandra.locator.SimpleSeedProvider`) The class within Cassandra that handles the seed logic. It can be customized, but this is typically not required.
- -seeds (Default: 127.0.0.1) A comma-delimited list of IP addresses. When running multiple nodes, you must change the list from the default value. In multiple data-center clusters, the seed list should include at least one node from each data center (replication group). More than a single seed node per data center is recommended for fault tolerance. Otherwise, gossip has to communicate with another data center when bootstrapping a node. Making every node a seed node is **not** recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per data center).

Related information: Initializing a multiple node cluster (single data center) and Initializing a multiple node cluster (multiple data centers).

Common compaction settings

compaction_throughput_mb_per_sec

(Default: 16) Throttles compaction to the specified total throughput across the entire system. The faster you insert data, the faster you need to compact in order to keep the SSTable count down. The recommended Value is 16 to 32 times the rate of write throughput (in MB/second). Setting the value to 0 disables compaction throttling.

Related information: Configuring compaction

Common memtable settings

memtable_total_space_in_mb

(Default: 1/4 of the heap)[note] Specifies the total memory used for all memtables on a node. This replaces the per-table storage settings memtable_operations_in_millions and memtable_throughput_in_mb.

Related information: Tuning the Java heap

Common disk settings

concurrent_reads

(Default: 32) For workloads with more data than can fit in memory, the bottleneck is reads fetching data from disk. Setting to (16 × number_of_drives) allows operations to queue low enough in the stack so that the OS and drives can reorder them.The default setting applies to both logical volume managed (LVM) and RAID drives.

concurrent_writes

(Default: 32) Writes in Cassandra are rarely I/O bound, so the ideal number of concurrent writes depends on the number of CPU cores in your system. The recommended value is 8 × number_of_cpu_cores.

Common automatic backup settings

incremental_backups

(Default: false) Backs up data updated since the last snapshot was taken. When enabled, Cassandra creates a hard link to each SSTable flushed or streamed locally in a backups/ subdirectory of the keyspace data. Removing these links is the operator's responsibility.

Related information: Enabling incremental backups

snapshot_before_compaction

(Default: false) Enable or disable taking a snapshot before each compaction. This option is useful to back up data when there is a data format change. Be careful using this option because Cassandra does not clean up older snapshots automatically.

Related information: Configuring compaction

Common fault detection setting

phi_convict_threshold

(Default: 8)[note] Adjusts the sensitivity of the failure detector on an exponential scale. Lower values increase the likelihood that an unresponsive node will be marked as down, while higher values decrease the likelihood that transient failures will cause a node failure. In unstable network environments (such as EC2 at times), raising the value to 10 or 12 helps prevent false failures. Values higher than 12 and lower than 5 are not recommended.

Related information: Failure detection and recovery

Performance tuning properties

Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.

Commit log settings

commitlog_sync

(Default: periodic) The method that Cassandra uses to acknowledge writes in milliseconds:

- periodic: Used with commitlog_sync_period_in_ms (Default: 10000 - 10 seconds) to control how often the commit log is synchronized to disk. Periodic syncs are acknowledged immediately.
- batch: Used with commitlog_sync_batch_window_in_ms (Default: disabled)[note] to control how long Cassandra waits for other writes before performing a sync. When using this method, writes are not acknowledged until fsynced to disk.

Related information: Durability

commitlog_periodic_queue_size

(Default: 1024 × number_of_cpu_cores) pending entries on the commitlog queue). When writing very large blobs, reduce this number. For example, 16 × number_of_cpu_cores works reasonably well for 1MB blobs. This setting should be at least as large as the concurrent_writes setting.

commitlog_segment_size_in_mb

(Default: 32) Sets the size of the individual commitlog file segments. A commitlog segment may be archived, deleted, or recycled after all its data has been flushed to SSTables. This amount of data can potentially include commitlog segments from every table in the system. The default size is usually suitable for most commitlog archiving, but if you want a finer granularity, 8 or 16 MB is reasonable. See Commit log archive configuration.

commitlog_total_space_in_mb

(Default: 32 for 32-bit JVMs, 1024 for 64-bit JVMs)[note] Total space used for commitlogs. If the used space goes above this value, Cassandra rounds up to the next nearest segment multiple and flushes memtables to disk for the oldest commitlog segments, removing those log segments. This reduces the amount of data to replay on start-up, and prevents infrequently-updated tables from indefinitely keeping commitlog segments. A small total commitlog space tends to cause more flush activity on less-active tables.

Related information: Configuring memtable throughput

Compaction settings

Related information: Configuring compaction

compaction_preheat_key_cache

(Default: true) When set to true, cached row keys are tracked during compaction, and re-cached to their new positions in the compacted SSTable. If you have extremely large key caches for tables, set the value to false; see Global row and key caches properties.

concurrent_compactors

(Default: 1 per CPU core)[note] Sets the number of concurrent compaction processes allowed to run simultaneously on a node, not including validation compactions for anti-entropy repair. Simultaneous compactions help preserve read performance in a mixed read-write workload by mitigating the tendency of small SSTables to accumulate during a single long-running compaction. If compactions run too slowly or too fast, change compaction_throughput_mb_per_sec first.

in_memory_compaction_limit_in_mb

(Default: 64) Size limit for rows being compacted in memory. Larger rows spill to disk and use a slower two-pass compaction process. When this occurs, a message is logged specifying the row key. The recommended value is 5 to 10 percent of the available Java heap size.

multithreaded_compaction

(Default: false) When set to true, each compaction operation uses one thread per core and one thread per SSTable being merged. This is typically useful only on nodes with SSD hardware. With HDD hardware, the goal is to limit the disk I/O for compaction (see compaction_throughput_mb_per_sec).

Note: It is strongly recommended to **not** enable multithreaded_compaction. In most cases it has severe performance impact. See CASSANDRA-6142.

preheat_kernel_page_cache

(Default: false) Enable or disable kernel page cache preheating from contents of the key cache after compaction. When enabled it preheats only first page (4KB) of each row to optimize for sequential access. It can be harmful for fat rows, see CASSANDRA-4937 for more details.

Memtable settings

file_cache_size_in_mb

(Default: smaller of 1/4 heap or 512) Total memory to use for SSTable-reading buffers.

memtable_flush_queue_size

(Default: 4) The number of full memtables to allow pending flush (memtables waiting for a write thread). At a minimum, set to the maximum number of indexes created on a single table.

Related information: Flushing data from the memtable

memtable_flush_writers

(Default: 1 per data directory)[note] Sets the number of memtable flush writer threads. These threads are blocked by disk I/O, and each one holds a memtable in memory while blocked. If you have a large Java heap size and many data directories, you can increase the value for better flush performance.

Related information: Flushing data from the memtable

Cache and index settings

column_index_size_in_kb

(Default: 64) Add column indexes to a row when the data reaches this size. This value defines how much row data must be de-serialized to read the column. Increase this setting if your column values are large or if you have a very large number of columns. If consistently reading only a few columns from each row or doing many partial-row reads, keep it small. All index data is read for each access, so take that into consideration when setting the index size.

populate_io_cache_on_flush

(Default: false)[note] Adds newly flushed or compacted SSTables to the operating system page cache, potentially evicting other cached data to make room. Enable when all data in the table is expected to fit in memory. See also the global option, compaction_preheat_key_cache.

Related information: CQLCompression Subproperties in *CQL for Cassandra 2.x.*

reduce_cache_capacity_to

(Default: 0.6) Sets the size percentage to which maximum cache capacity is reduced when Java heap usage reaches the threshold defined by reduce_cache_sizes_at.

reduce_cache_sizes_at

(Default: 0.85) When Java heap usage (after a full concurrent mark sweep (CMS) garbage collection) exceeds this percentage, Cassandra reduces the cache capacity to the fraction of the current size as specified by reduce_cache_capacity_to. To disable, set the value to 1.0.

Disks settings

stream_throughput_outbound_megabits_per_sec

(Default: 200)[note] Throttles all outbound streaming file transfers on a node to the specified throughput. Cassandra does mostly sequential I/O when streaming data during bootstrap or repair, which can lead to saturating the network connection and degrading client (RPC) performance.

trickle_fsync

(Default: false) When doing sequential writing, enabling this option tells fsync to force the operating system to flush the dirty buffers at a set interval trickle_fsync_interval_in_kb. Enable this parameter to avoid sudden dirty buffer flushing from impacting read latencies. Recommended to use on SSDs, but not on HDDs.

trickle_fsync_interval_in_kb

(Default: 10240]). Sets the size of the fsync in kilobytes.

Advanced properties

Properties for advanced users or properties that are less commonly used.

Advanced initialization properties

auto_bootstrap

(Default: true) This setting has been removed from default configuration. It makes new (non-seed) nodes automatically migrate the right data to themselves. When initializing a fresh cluster *without* data, add auto_bootstrap: false.

Related information: Initializing a multiple node cluster (single data center) and Initializing a multiple node cluster (multiple data centers).

batch_size_warn_threshold_in_kb

(Default: 64kb per batch) Log WARN on any batch size exceeding this value in kilobytes. Caution should be taken on increasing the size of this threshold as it can lead to node instability.

broadcast_address

(Default: listen_address)[note] The IP address a node tells other nodes in the cluster to contact it by. It allows public and private address to be different. For example, use the broadcast_address parameter in topologies where not all nodes have access to other nodes by their private IP addresses.

If your Cassandra cluster is deployed across multiple Amazon EC2 regions and you use the `Ec2MultiRegionSnitch`, set the broadcast_address to public IP address of the node and the listen_address to the private IP. See Ec2MultiRegionSnitch.

initial_token

(Default: disabled) Used in the single-node-per-token architecture, where a node owns exactly one contiguous range in the ring space. Setting this property overrides num_tokens.

If you not using vnodes or have num_tokens set it to 1 or unspecified (`#num_tokens`), you should always specify this parameter when setting up a production cluster for the first time and when adding capacity. For more information, see this parameter in the Cassandra 1.1 Node and Cluster Configuration documentation.

num_tokens

(Default: 256)[note] Defines the number of tokens randomly assigned to this node on the ring when using virtual nodes (vnodes). The more tokens, relative to other nodes, the larger the proportion of data that the node stores. Generally all nodes should have the same number of tokens assuming equal hardware capability. The recommended value is 256. If unspecified (`#num_tokens`), Cassandra uses 1 (equivalent to `#num_tokens : 1`) for legacy compatibility and uses the initial_token setting.

If you do not wish to use vnodes, comment `#num_tokens : 256` or set `num_tokens : 1` and use initial_token. If you already have an existing cluster with one token per node and wish to migrate to vnodes, see Enabling virtual nodes on an existing production cluster.

Note: If using DataStax Enterprise, the default setting of this property depends on the type of node and type of install.

partitioner

(Default: `org.apache.cassandra.dht.Murmur3Partitioner`) Distributes rows (by key) across nodes in the cluster. Any `IPartitioner` may be used, including your own as long as it is in the class path. Cassandra provides the following partitioners for backwards compatibility:

- `RandomPartitioner`
- `ByteOrderedPartitioner`
- `OrderPreservingPartitioner` (deprecated)

Related information: Partitioners

storage_port

(Default: 7000) The port for inter-node communication.

Advanced automatic backup setting

auto_snapshot

(Default: true) Enable or disable whether a snapshot is taken of the data before keyspace truncation or dropping of tables. To prevent data loss, using the default setting is strongly advised. If you set to false, you will lose data on truncation or drop.

Key caches and global row properties

When creating or modifying tables, you enable or disable the key cache (partition key cache) or row cache for that table by setting the caching parameter. Other row and key cache tuning and configuration options are set at the global (node) level. Cassandra uses these settings to automatically distribute memory for each table on the node based on the overall workload and specific table usage. You can also configure the save periods for these caches globally.

Related information: Configuring caches

key_cache_keys_to_save

(Default: disabled - all keys are saved)[note] Number of keys from the key cache to save.

key_cache_save_period

(Default: 14400 - 4 hours) Duration in seconds that keys are saved in cache. Caches are saved to saved_caches_directory. Saved caches greatly improve cold-start speeds and has relatively little effect on I/O.

key_cache_size_in_mb

(Default: empty) A global cache setting for tables. It is the maximum size of the key cache in memory. When no value is set, the cache is set to the smaller of 5% of the available heap, or 100MB. To disable set to 0.

row_cache_keys_to_save

(Default: disabled - all keys are saved)[note] Number of keys from the row cache to save.

row_cache_size_in_mb

(Default: 0 - disabled) A global cache setting for tables.

row_cache_save_period

(Default: 0 - disabled) Duration in seconds that rows are saved in cache. Caches are saved to saved_caches_directory.

memory_allocator

(Default: NativeAllocator) The off-heap memory allocator. In addition to caches, this property affects storage engine metadata. Supported values:

- NativeAllocator
- JEMallocAllocator - Experiments show that jemalloc saves some memory compared to the native allocator because it is more fragmentation resistant. To use, install jemalloc as a library and modify `cassandra-env.sh` (instructions in file).

Tombstone settings

When executing a scan, within or across a partition, tombstones must be kept in memory to allow returning them to the coordinator. The coordinator uses them to ensure other replicas know about the deleted rows. Workloads that generate numerous tombstones may cause performance problems and exhaust the server heap. Adjust these thresholds only if you understand the impact and want to scan more tombstones. Additionally, you can adjust these thresholds at runtime using the `StorageServiceMBean`.

Related information: Cassandra anti-patterns: Queues and queue-like datasets

tombstone_warn_threshold

(Default: 1000) The maximum number of tombstones a query can scan before warning.

tombstone_failure_threshold

(Default: 100000) The maximum number of tombstones a query can scan before aborting.

Network timeout settings

range_request_timeout_in_ms

(Default: 10000) The time in milliseconds that the coordinator waits for sequential or index scans to complete.

read_request_timeout_in_ms

(Default: 10000) The time in milliseconds that the coordinator waits for read operations to complete.

cas_contention_timeout_in_ms

(Default: 1000) The time in milliseconds that the coordinator continues to retry a CAS (compare and set) operation that contends with other proposals for the same row.

truncate_request_timeout_in_ms

(Default: 60000) The time in milliseconds that the coordinator waits for truncates to complete. The long default value allows for flushing of all tables, which ensures that anything in the commitlog is removed that could cause truncated data to reappear. If auto_snapshot is disabled, you can reduce this time.

write_request_timeout_in_ms

(Default: 10000) The time in milliseconds that the coordinator waits for write operations to complete.

request_timeout_in_ms

(Default: 10000) The default time out in milliseconds for other, miscellaneous operations.

Related information: About hinted handoff writes

Inter-node settings

cross_node_timeout

(Default: false) Enable or disable operation timeout information exchange between nodes (to accurately measure request timeouts). If disabled Cassandra assumes the request was forwarded to the replica instantly by the coordinator.

Caution: Before enabling this property make sure NTP (network time protocol) is installed and the times are synchronized between the nodes.

internode_send_buff_size_in_bytes

(Default: N/A)[note] Sets the sending socket buffer size in bytes for inter-node calls.

internode_recv_buff_size_in_bytes

(Default: N/A)[note] Sets the receiving socket buffer size in bytes for inter-node calls.

internode_compression

(Default: all) Controls whether traffic between nodes is compressed. The valid values are:

- all: All traffic is compressed.
- dc: Traffic between data centers is compressed.
- none: No compression.

inter_dc_tcp_nodelay

(Default: false) Enable or disable tcp_nodelay for inter-data center communication. When disabled larger, but fewer, network packets are sent. This reduces overhead from the TCP protocol itself. However, if cross data-center responses are blocked, it will increase latency.

streaming_socket_timeout_in_ms

(Default: 0 - never timeout streams)[note] Enable or disable socket timeout for streaming operations. When a timeout occurs during streaming, streaming is retried from the start of the current file. Avoid setting this value too low, as it can result in a significant amount of data re-streaming.

Native transport (CQL Binary Protocol)

start_native_transport

(Default: true) Enable or disable the native transport server. Note that the address on which the native transport is bound is the same as the rpc_address. However, the port is different from the rpc_port and specified in native_transport_port.

native_transport_port

(Default: 9042) Port on which the CQL native transport listens for clients.

native_transport_max_threads

(Default: 128)[note] The maximum number of thread handling requests. The meaning is the same as rpc_max_threads.

native_transport_max_frame_size_in_mb

(Default: 256MB) The maximum size of allowed frame. Frame (requests) larger than this are rejected as invalid.

RPC (remote procedure call) settings

Settings for configuring and tuning client connections.

rpc_port

(Default: 9160) The port for the Thrift RPC service, which is used for client connections.

start_rpc

(Default: true) Starts the Thrift RPC server.

rpc_keepalive

(Default: true) Enable or disable keepalive on client connections.

rpc_max_threads

(Default: unlimited)[note] Regardless of your choice of RPC server (rpc_server_type), the number of maximum requests in the RPC thread pool dictates how many concurrent requests are possible. However, if you are using the parameter sync in the rpc_server_type, it also dictates the number of clients that can be connected. For a large number of client connections, this could cause excessive memory usage for the thread stack. Connection pooling on the client side is highly recommended. Setting a maximum thread pool size acts as a safeguard against misbehaved clients. If the maximum is reached, Cassandra blocks additional connections until a client disconnects.

rpc_min_threads

(Default: 16)[note] Sets the minimum thread pool size for remote procedure calls.

rpc_recv_buff_size_in_bytes

(Default: N/A)[note] Sets the receiving socket buffer size for remote procedure calls.

rpc_send_buff_size_in_bytes

(Default: N/A)[note] Sets the sending socket buffer size in bytes for remote procedure calls.

rpc_server_type

(Default: sync) Cassandra provides three options for the RPC server. On Windows, sync is about 30% slower than hsha. On Linux, sync and hsha performance is about the same, but hsha uses less memory.

- sync: (default) One connection per thread in the RPC pool. For a very large number of clients, memory is the limiting factor. On a 64 bit JVM, 128KB is the minimum stack size per thread. Connection pooling is strongly recommended.
- hsha: Half synchronous, half asynchronous. The RPC thread pool is used to manage requests, but the threads are multiplexed across the different clients. All Thrift clients are handled asynchronously using a small number of threads that does not vary with the number of clients (and thus scales well to many clients). The RPC requests are synchronous (one thread per active request).
- Your own RPC server: You must provide a fully-qualified class name of an `o.a.c.t.TServerFactory` that can create a server instance.

Advanced fault detection settings

Settings to handle poorly performing or failing nodes.

dynamic_snitch_badness_threshold

(Default: 0.1) Sets the performance threshold for dynamically routing requests away from a poorly performing node. A value of 0.2 means Cassandra continues to prefer the static snitch values until the node response time is 20% worse than the best performing node. Until the threshold is reached, incoming client requests are statically routed to the closest replica (as determined by the snitch). Having requests consistently routed to a given replica can help keep a working set of data hot when read repair is less than 1.

dynamic_snitch_reset_interval_in_ms

(Default: 600000) Time interval in milliseconds to reset all node scores, which allows a bad node to recover.

dynamic_snitch_update_interval_in_ms

(Default: 100) The time interval in milliseconds for calculating read latency.

hinted_handoff_enabled

(Default: true) Enable or disable hinted handoff. It can also contain a list of data centers to enable per data-center. For example hinted_handoff_enabled: DC1,DC2. A hint indicates that the write needs to be replayed to an unavailable node. Where Cassandra writes the hint depends on the version:

- Prior to 1.0: Writes to a live replica node.
- 1.0 and later: Writes to the coordinator node.

Related information: About hinted handoff writes

hinted_handoff_throttle_in_kb

(Default: 1024) Maximum throttle per delivery thread in kilobytes per second. This rate reduces proportionally to the number of nodes in the cluster. For example, if there are two nodes in the cluster, each delivery thread will use the maximum rate; if there are three, each node will throttle to half of the maximum, since the two nodes are expected to deliver hints simultaneously.

max_hint_window_in_ms

(Default: 10800000 - 3 hours) Defines how long in milliseconds to generate and save hints for an unresponsive node. After this interval, new hints are no longer generated until the node is back up and responsive. If the node goes down again, a new interval begins. This setting can prevent a sudden demand for resources when a node is brought back online and the rest of the cluster attempts to replay a large volume of hinted writes.

Related information: Failure detection and recovery

max_hints_delivery_threads

(Default: 2) Number of threads with which to deliver hints. For multiple data center deployments, consider increasing this number because cross data-center handoff is generally slower.

batchlog_replay_throttle_in_kb

(Default: 1024) Maximum throttle in KBs per second, total. It is reduced proportionally to the number of nodes in the cluster.

Request scheduler properties

Settings to handle incoming client requests according to a defined policy. If you need to use these properties, your nodes are overloaded and dropping requests. It is recommended that you add more nodes and not try to prioritize requests.

request_scheduler

(Default: `org.apache.cassandra.scheduler.NoScheduler`) Defines a scheduler to handle incoming client requests according to a defined policy. This scheduler is useful for throttling client requests in single clusters containing multiple keyspaces. This is specifically for requests from the client and does not affect inter-node communication. Valid values are:

- `org.apache.cassandra.scheduler.NoScheduler`: No scheduling takes place and does not have any options.
- `org.apache.cassandra.scheduler.RoundRobinScheduler`: See request_scheduler_options properties.
- A Java class that implements the `RequestScheduler` interface.

request_scheduler_id

(Default: keyspace)[note] An identifier on which to perform request scheduling. Currently the only valid value is keyspace.

request_scheduler_options

(Default: disabled) Contains a list of properties that define configuration options for request_scheduler:

- throttle_limit: (Default: 80) The number of active requests per client. Requests beyond this limit are queued up until running requests complete. Recommended value is ((concurrent_reads + concurrent_writes) × 2).
- default_weight: (Default: 5)[note] How many requests are handled during each turn of the `RoundRobin`.
- weights: (Default: Keyspace1: 1, Keyspace2: 5) How many requests are handled during each turn of the `RoundRobin`, based on the request_scheduler_id. Takes a list of keyspaces: weights.

Thrift interface properties

Legacy API for older clients. CQL is a simpler and better API for Cassandra.

thrift_framed_transport_size_in_mb

(Default: 15) Frame size (maximum field length) for Thrift. The frame is the row or part of the row that the application is inserting.

thrift_max_message_length_in_mb

(Default: 16) The maximum length of a Thrift message in megabytes, including all fields and internal Thrift overhead (1 byte of overhead for each frame). Message length is usually used in conjunction with batches. A frame length greater than or equal to 24 accommodates a batch with four inserts, each of which is 24 bytes. The required message length is greater than or equal to 24+24+24+24+4 (number of frames).

Security properties

Server and client security settings.

authenticator

(Default: `AllowAllAuthenticator`) The authentication backend. It implements IAuthenticator, which is used to identify users. The available authenticators are:

- `AllowAllAuthenticator`: Disables authentication; no checks are performed.
- `PasswordAuthenticator`: Authenticates users with user names and hashed passwords stored in the system_auth.credentials table. If you use this authenticator, increase the system_auth keyspace replication factor.

Related information: Internal authentication

internode_authenticator

(Default: enabled)[note] Internode authentication backend, implementing `org.apache.cassandra.auth.AllowAllInternodeAuthenticator`. Allows or disallows connections from peer nodes.

authorizer

(Default: `AllowAllAuthorizer`) The authorization backend. It implements IAuthenticator, which limits access and provides permissions. The available authorizers are:

- `AllowAllAuthorizer`: Disables authorization; allows any action to any user.
- `CassandraAuthorizer`: Stores permissions in system_auth.permissions table. If you use this authenticator, increase the system_auth keyspace replication factor.

Related information: Object permissions

permissions_validity_in_ms

(Default: 2000) How long permissions in cache remain valid. Depending on the authorizer, fetching permissions can be resource intensive. This setting is automatically disabled when `AllowAllAuthorizer` is set.

Related information: Object permissions

server_encryption_options

Enable or disable inter-node encryption. You must also generate keys and provide the appropriate key and trust store locations and passwords. No custom encryption options are currently enabled. The available options are:

- internode_encryption: (Default: none) Enable or disable encryption of inter-node communication using the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite for authentication, key exchange, and encryption of data transfers. The available inter-node options are:
  - all: Encrypt all inter-node communications.
  - none: No encryption.
  - dc: Encrypt the traffic between the data centers (server only).
  - rack: Encrypt the traffic between the racks(server only).
- keystore: (Default: `conf/.keystore`) The location of a Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and

Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.

- keystore_password: (Default: *cassandra*) Password for the keystore.
- truststore: (Default: `conf/.truststore`) Location of the truststore containing the trusted certificate for authenticating remote servers.
- truststore_password: (Default: *cassandra*) Password for the truststore.

The passwords used in these options must match the passwords used when generating the keystore and truststore. For instructions on generating these files, see Creating a Keystore to Use with JSSE.

The advanced settings are:

- protocol: (Default: TLS)
- algorithm: (Default: SunX509)
- store_type: (Default: JKS)
- cipher_suites: (Default: TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA)
- require_client_auth: (Default: false) Enables or disables certificate authentication.

Related information: Node-to-node encryption

client_encryption_options

Enable or disable client-to-node encryption. You must also generate keys and provide the appropriate key and trust store locations and passwords. No custom encryption options are currently enabled. The available options are:

- enabled: (Default: false) To enable, set to true.
- keystore: (Default: `conf/.keystore`) The location of a Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.
- keystore_password: (Default: *cassandra*) Password for the keystore. This must match the password used when generating the keystore and truststore.
- require_client_auth: (Default: false) Enables or disables certificate authentication. (Available starting with Cassandra 1.2.3.)
- truststore: (Default: `conf/.truststore`) Set if require_client_auth is true.
- truststore_password: *<truststore_password>* Set if require_client_auth is true.

The advanced settings are:

- protocol: (Default: TLS)
- algorithm: (Default: SunX509)
- store_type: (Default: JKS)
- cipher_suites: (Default: TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA)

Related information: Client-to-node encryption

ssl_storage_port

(Default: 7001) The SSL port for encrypted communication. Unused unless enabled in encryption_options.

Configuring gossip settings

About this task

When a node first starts up, it looks at its cassandra.yaml configuration file to determine the name of the Cassandra cluster it belongs to; which nodes (called *seeds*) to contact to obtain information about the other nodes in the cluster; and other parameters for determining port and range information.

Procedure

In the `cassandra.yaml` file, set the following parameters:

Property	Description
cluster_name	Name of the cluster that this node is joining. Must be the same for every node in the cluster.
listen_address	The IP address or hostname that Cassandra binds to for listening to other Cassandra nodes.
(Optional) broadcast_address	The IP address a node tells other nodes in the cluster to contact it by. It allows public and private address to be different. For example, use the broadcast_address parameter in topologies where not all nodes have access to other nodes by their private IP addresses. The default is the listen_address.
seed_provider	A -seeds list is comma-delimited list of hosts (IP addresses) that gossip uses to learn the topology of the ring. Every node should have the same list of seeds. In multiple data-center clusters, the seed list should include at least one node from each data center (replication group). More than a single seed node per data center is recommended for fault tolerance. Otherwise, gossip has to communicate with another data center when bootstrapping a node. Making every node a seed node is **not** recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per data center).
storage_port	The inter-node communication port (default is 7000). Must be the same for every node in the cluster.
initial_token	For legacy clusters. Used in the single-node-per-token architecture, where a node owns exactly one contiguous range in the ring space.
num_tokens	For new clusters. Defines the number of tokens randomly assigned to this node on the ring when using virtual nodes (vnodes).

Configuring the heap dump directory

About this task

Analyzing the heap dump file can help troubleshoot memory problems. Cassandra starts Java with the option `-XX:-HeapDumpOnOutOfMemoryError`. Using this option triggers a heap dump in the event of an out-of-memory condition. The heap dump file consists of references to objects that cause the heap to overflow. By default, Cassandra puts the file a subdirectory of the working, root directory when running as a service. If Cassandra does not have write permission to the root directory, the heap dump fails. If the root directory is too small to accommodate the heap dump, the server crashes.

For a heap dump to succeed and to prevent crashes, configure a heap dump directory that meets these requirements:

- Accessible to Cassandra for writing
- Large enough to accommodate a heap dump

This file is located in:

- Packaged installs: `/etc/dse/cassandra`
- Tarball installs: `install_location/resources/cassandra/conf`

Base the size of the directory on the value of the Java `-mx` option.

Procedure

1. Open the `cassandra-env.sh` file for editing.

   ```
   # set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR
   ```
2. Scroll down to the comment about the heap dump path:

   ```
   # set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR
   ```
3. On the line after the comment, set the `CASSANDRA_HEAPDUMP_DIR` to the path you want to use:

   ```
   # set jvm HeapDumpPath with CASSANDRA_HEAPDUMP_DIR
   CASSANDRA_HEAPDUMP_DIR =<path>
   ```
4. Save the `cassandra-env.sh` file and restart.

Generating tokens

If not using virtual nodes (vnodes), you still need to calculate tokens for your cluster.

The following topics in the Cassandra 1.1 documentation provide conceptual information about tokens:

- Data Distribution in the Ring
- Replication Strategy

About calculating tokens for single or multiple data centers in Cassandra 1.2 and later

- Single data center deployments: calculate tokens by dividing the hash range by the number of nodes in the cluster.
- Multiple data center deployments: calculate the tokens for each data center so that the hash range is evenly divided for the nodes in each data center.

For more explanation, see be sure to read the conceptual information mentioned above.

The method used for calculating tokens depends on the type of partitioner:

Calculating tokens for the Murmur3Partitioner

Use this method for generating tokens when you are **not** using virtual nodes (vnodes) and using the Murmur3Partitioner (default). This partitioner uses a maximum possible range of hash values from -2^{63} to $+2^{63} -1$. To calculate tokens for this partitioner:

```
python -c 'print [str(((2**64 / number_of_tokens) * i) - 2**63) for i in
 range(number_of_tokens)]'
```

For example, to generate tokens for 6 nodes:

```
python -c 'print [str(((2**64 / 6) * i) - 2**63) for i in range(6)]'
```

The command displays the token for each node:

```
[ '-9223372036854775808', '-6148914691236517206', '-3074457345618258604',
  '-2', '3074457345618258600', '6148914691236517202' ]
```

Calculating tokens for the RandomPartitioner

To calculate tokens when using the RandomPartitioner in Cassandra 1.2 clusters, use the Cassandra 1.1 Token Generating Tool.

Configuring virtual nodes

Enabling virtual nodes on a new cluster

About this task

Generally when all nodes have equal hardware capability, they should have the same number of virtual nodes (vnodes). If the hardware capabilities vary among the nodes in your cluster, assign a proportional number of vnodes to the larger machines. For example, you could designate your older machines to use 128 vnodes and your new machines (that are twice as powerful) with 256 vnodes.

Procedure

Set the number of tokens on each node in your cluster with the num_tokens parameter in the `cassandra.yaml` file.

The recommended value is 256. Do not set the initial_token parameter.

Enabling virtual nodes on an existing production cluster

About this task

For production clusters, enabling virtual nodes (vnodes) has less impact on performance if you bring up a another data center configured with vnodes already enabled and let Cassandra automatic mechanisms distribute the existing data into the new nodes.

Procedure

1. Add a new data center to the cluster.
2. Once the new data center with vnodes enabled is up, switch your clients to use the new data center.
3. Run a full repair with nodetool repair.

This step ensures that after you move the client to the new data center that any previous writes are added to the new data center and that nothing else, such as hints, is dropped when you remove the old data center.

4. Update your schema to no longer reference the old data center.

5. Remove the old data center from the cluster.

See Decommissioning a data center.

Logging configuration

Logging configuration

To get more diagnostic information about the runtime behavior of a specific Cassandra node than what is provided by Cassandra's JMX MBeans and the nodetool utility, you can increase the logging levels on specific portions of the system using log4j.

Cassandra provides logging functionality using Simple Logging Facade for Java (SLF4J) with a log4j backend. Additionally, the output.log captures the stdout of the Cassandra process, which is configurable using the standard Linux logrotate facility. You can also change logging levels via JMX using the JConsole tool.

The logging levels from most to least verbose are:

* TRACE
* DEBUG
* INFO
* WARN
* ERROR
* FATAL

Note: Be aware that increasing logging levels can generate a lot of logging output on even a moderately trafficked cluster.

Changing Logging Levels

About this task

The default logging level is determined by the following line in the `log4j-server.properties` file:

`log4j.rootLogger=INFO,stdout,R`

To exert more fine-grained control over your logging, you can specify the logging level for specific categories. The categories usually (but not always) correspond to the package and class name of the code doing the logging. For example, the following setting logs DEBUG messages from all classes in the `org.apache.cassandra.db` package:

`log4j.logger.org.apache.cassandra.db=DEBUG`

In this example, DEBUG messages are logged specifically from the `StorageProxy` class in the `org.apache.cassandra.service` package:

`log4j.logger.org.apache.cassandra.service.StorageProxy=DEBUG`

To determine which category a particular message in the log belongs to, you change the following line:

`log4j.appender.R.layout.ConversionPattern=%5p [%t] %d{ISO8601} %F (line %L) %m %n`

Procedure

1. Add `%c` at the beginning of the conversion pattern:

   ```
   log4j.appender.R.layout.ConversionPattern=%c %5p [%t] %d{ISO8601} %F (line
    %L) %m%n
   ```

   Each log message is now prefixed with the category.

2. After Cassandra runs for a while, use the following command to determine which categories are logging the most messages:

   ```
   cat system.log.* | egrep 'TRACE|DEBUG|INFO|WARN|ERROR|FATAL' | awk '{ print
    $1 }' | sort | uniq -c | sort -n
   ```

3. If you find that a particular class logs too many messages, use the following format to set a less verbose logging level for that class by adding a line for that class:

   ```
   loggerog4j.logger.package.class=WARN
   ```

   For example a busy Solr node can log numerous INFO messages from the SolrCore, LogUpdateProcessorFactory, and SolrIndexSearcher classes. To suppress these messages, add the following lines:

   ```
   log4j.logger.org.apache.solr.core.SolrCore = WARN
   log4j.logger.org.apache.solr.update.processor.LogUpdateProcessorFactory=WARN
   log4j.logger.org.apache.solr.search.SolrIndexSearcher=WARN
   ```

4. After determining which category a particular message belongs to you may want to revert the messages back to the default format. Do this by removing `%c` from the `ConversionPattern`.

Changing the rotation and size of the Cassandra output.log

About this task

Cassandra's `output.log` logging configuration is controlled by the `log4j-server.properties` file in the following directories:

file in the following directories:

- Packaged installs: `/etc/dse/cassandra`
- Tarball installs: *install_location*`/resources/cassandra/conf`

The `output.log` stores the stdout of the Cassandra process; it is not controllable from log4j. However, you can rotate it using the standard Linux logrotate facility.

The `copytruncate` directive is critical because it allows the log to be rotated without any support from Cassandra for closing and reopening the file. For more information, refer to the logrotate man page.

To configure logrotate to work with Cassandra, create a file called `/etc/logrotate.d/cassandra` with the following contents:

```
/var/log/cassandra/output.log {
  size 10M
  rotate 9
  missingok
  copytruncate
  compress
}
```

Changing the rotation and size of the Cassandra system.log

About this task

Cassandra's system.log logging configuration is controlled by the `log4j-server.properties` file in the following directories:

file in the following directories:

- Packaged installs: `/etc/cassandra`
- Tarball installs: `install_location/conf`

About this task

The maximum log file size and number of backup copies are controlled by the following lines:

```
log4j.appender.R.maxFileSize=20MB
log4j.appender.R.maxBackupIndex=50
```

The default configuration rolls the log file once the size exceeds 20MB and maintains up to 50 backups. When the `maxFileSize` is reached, the current log file is renamed to `system.log.1` and a new `system.log` is started. Any previous backups are renumbered from `system.log.n` to `system.log.n+1`, which means the higher the number, the older the file. When the maximum number of backups is reached, the oldest file is deleted.

- By default, logging output is placed the `/var/log/cassandra/system.log`. You can change the location of the output by editing the `log4j.appender.R.File` path. Be sure that the directory exists and is writable by the process running Cassandra.
- If an issue occurred but has already been rotated out of the current `system.log`, check to see if it is captured in an older backup. If you want to keep more history, increase the `maxFileSize`, `maxBackupIndex`, or both. Make sure you have enough space to store the additional logs.

Commit log archive configuration

About this task

Cassandra provides commitlog archiving and point-in-time recovery. You configure this feature in the `commitlog_archiving.properties` configuration file, which is located in the following directories:

- Cassandra package installations: `/etc/cassandra`
- Cassandra tarball installations: `install_location/conf`
- DataStax Enterprise package installations: `/etc/dse/cassandra`
- DataStax Enterprise tarball installations: `install_location/resources/cassandra/conf`

The commands `archive_command` and `restore_command` expect only a single command with arguments. The parameters must be entered verbatim. STDOUT and STDIN or multiple commands cannot be executed. To workaround, you can script multiple commands and add a pointer to this file. To disable a command, leave it blank.

Procedure

- Archive a commitlog segment:

Command	archive_command=	
Parameters	`%path`	Fully qualified path of the segment to archive.

Command	archive_command=	
	`%name`	Name of the commit log.
Example	`archive_command=/bin/ln %path /` `backup/%name`	

- Restore an archived commitlog:

Command	restore_command=	
Parameters	`%from`	Fully qualified path of the an archived commitlog segment from the *restore_directories*.
	`%t`	Name of live commit log directory.
Example	`restore_command=cp -f %from %to`	

- Set the restore directory location:

Command	restore_directories=
Format	`restore_directories=`*`restore_directory_location`*

- Restore mutations created up to and including the specified timestamp:

Command	restore_point_in_time=
Format	`<timestamp>` (YYYY:MM:DD HH:MM:SS)
Example	`restore_point_in_time=2013:12:11` `17:00:00`

Restore stops when the first client-supplied timestamp is greater than the restore point timestamp. Because the order in which Cassandra receives mutations does not strictly follow the timestamp order, this can leave some mutations unrecovered.

Using multiple network interfaces

About this task

How to configure Cassandra for multiple network interfaces or when using different regions in cloud implementations.

You must configure settings in both the `cassandra.yaml` file and the property file (`cassandra-rackdc.properties` or `cassandra-topology.properties`) used by the snitch.

Configuring cassandra.yaml for multiple networks or across regions in cloud implementations

In multiple networks or cross-region cloud scenarios, communication between data centers can only take place using an external IP address. The external IP address is defined in the `cassandra.yaml` file using the broadcast_address setting. Configure each node as follows:

1. In the cassandra.yaml, set the listen_address to the *private* IP address of the node, and the broadcast_address to the *public* address of the node.

   This allows Cassandra nodes to bind to nodes in another network or region, thus enabling multiple data-center support. For intra- network or region traffic, Cassandra switches to the private IP after establishing a connection.

2. Set the addresses of the seed nodes in the `cassandra.yaml` file to that of the *public* IP. Private IP are not routable between networks. For example:

```
seeds: 50.34.16.33, 60.247.70.52
```

Note: Do not make all nodes seeds, see Internode communications (gossip).

3. Be sure that the storage_port or ssl_storage_port is open on the public IP firewall.

Caution: Be sure to enable encryption and authentication when using public IP's. See Node-to-node encryption. Another option is to use a custom VPN to have local, inter-region/datacenter IP's.

Configuring the snitch for multiple networks

External communication between the data centers can only happen when using the broadcast_address (public IP).

The GossipingPropertyFileSnitch is recommended for production. The `cassandra-rackdc.properties` file defines the data centers used by this snitch.

For each node in the network, specify its data center in cassandra-rackdc.properties file.

In the example below, there are two cassandra data centers and each data center is named for its workload. The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (Data center names are case-sensitive.)

Network A	Network B
Node and data center:	Node and data center:
• **node0**	• **node0**

dc=DC_A_cassandra
  rack=RAC1
	dc=DC_A_cassandra
	  rack=RAC1

• **node1**	• **node1**

dc=DC_A_cassandra
  rack=RAC1
	dc=DC_A_cassandra
	  rack=RAC1

• **node2**	• **node2**

dc=DC_B_cassandra
  rack=RAC1
	dc=DC_B_cassandra
	  rack=RAC1

• **node3**	• **node3**

dc=DC_B_cassandra
  rack=RAC1
	dc=DC_B_cassandra
	  rack=RAC1

• **node4**	• **node4**

dc=DC_A_analytics
  rack=RAC1
	dc=DC_A_analytics
	  rack=RAC1

• **node5**	• **node5**

dc=DC_A_search
  rack=RAC1
	dc=DC_A_search
	  rack=RAC1

Configuring the snitch for cross-region communication in cloud implementations

Note: Be sure to use the appropriate snitch for your implementation. If your deploying on Amazon EC2, see the instructions in Ec2MultiRegionSnitch.

In cloud deployments, the The region name is treated as the data center name and availability zones are treated as racks within a data center. For example, if a node is in the us-east-1 region, us-east is the data center name and 1 is the rack location. (Racks are important for distributing replicas, but not for data center naming.)

In the example below, there are two cassandra data centers and each data center is named for its workload. The data center naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (Data center names are case-sensitive.)

For each node, specify its data center in the `cassandra-rackdc.properties`. The dc_suffix option defines the data centers used by the snitch. Any other lines are ignored.

Region: us-east	Region: us-west
Node and data center:	Node and data center:
• **node0**	• **node0**
`dc_suffix=_1_cassandra`	`dc_suffix=_1_cassandra`
• **node1**	• **node1**
`dc_suffix=_1_cassandra`	`dc_suffix=_1_cassandra`
• **node2**	• **node2**
`dc_suffix=_2_cassandra`	`dc_suffix=_2_cassandra`
• **node3**	• **node3**
`dc_suffix=_2_cassandra`	`dc_suffix=_2_cassandra`
• **node4**	• **node4**
`dc_suffix=_1_analytics`	`dc_suffix=_1_analytics`
• **node5**	• **node5**
`dc_suffix=_1_search`	`dc_suffix=_1_search`
This results in four us-east data centers:	This results in four us-west data centers:

`us-east_1_cassandra`
`us-east_2_cassandra`
`us-east_1_analytics`
`us-east_1_search`
	`us-west_1_cassandra`
	`us-west_2_cassandra`
	`us-west_1_analytics`
	`us-west_1_search`

Hadoop support

Cassandra 2.0.11 improves support for integrating Hadoop with Cassandra:

- MapReduce
- Apache Pig
- Apache Hive

You can use Cassandra 2.0.11 and later with Hadoop 2.x or 1.x with some restrictions.

- Isolate Cassandra and Hadoop nodes in separate data centers.
- Before starting the data centers of Cassandra/Hadoop nodes, disable virtual nodes (vnodes).

To disable virtual nodes:

1. In the cassandra.yaml file, set num_tokens to 1.
2. Uncomment the initial_token property and set it to 1 or to the value of a generated token for a multi-node cluster.
3. Start the cluster for the first time.

Do not disable or enable vnodes on an existing cluster.

Setup and configuration, described in the Apache docs, involves overlaying a Hadoop cluster on Cassandra nodes, configuring a separate server for the Hadoop NameNode/JobTracker, and installing a Hadoop TaskTracker and Data Node on each Cassandra node. The nodes in the Cassandra data center can draw from data in the HDFS Data Node as well as from Cassandra. The Job Tracker/Resource Manager (JT/RM) receives MapReduce input from the client application. The JT/RM sends a MapReduce job request to the Task Trackers/Node Managers (TT/NM) and optional clients, MapReduce, Hive, and Pig. The data is written to Cassandra and results sent back to the client.

MapReduce Process in a Cassandra/Hadoop Cluster
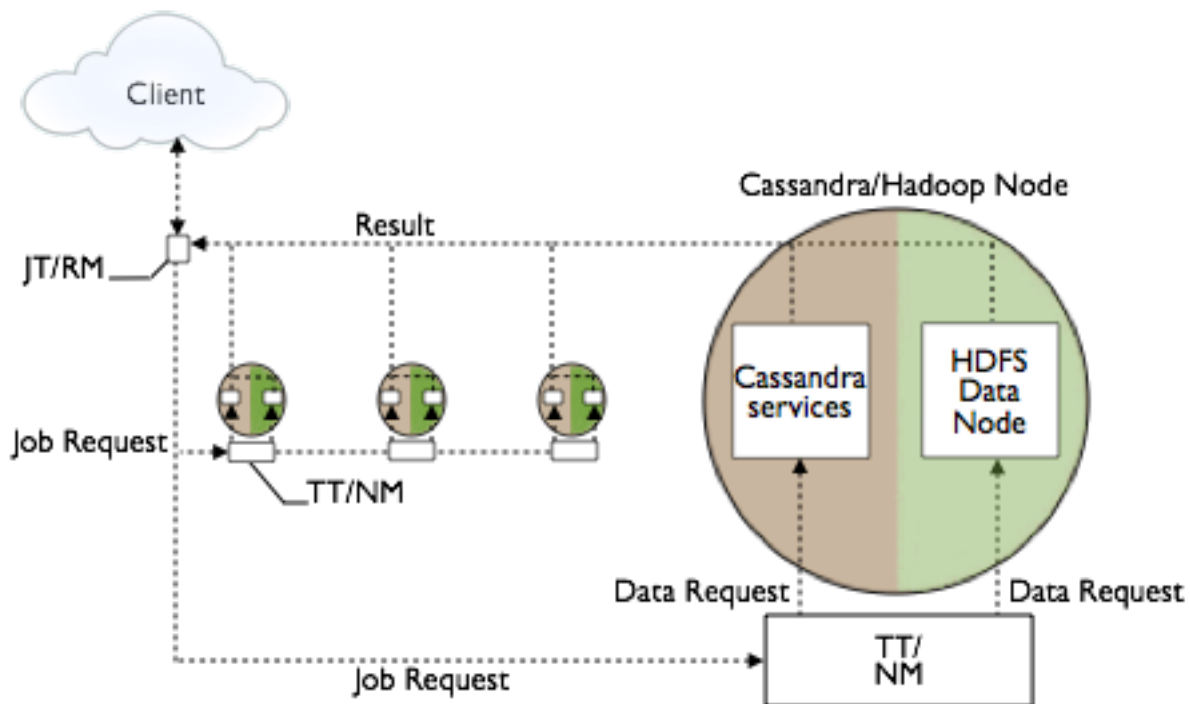
The Apache docs also cover how to get configuration and integration support.

Input and Output Formats

Hadoop jobs can receive data from CQL tables and indexes and you can load data into Cassandra from a Hadoop job. Cassandra 2.0.11 supports the following formats for these tasks:

* CQL partition input format: ColumnFamilyInputFormat class.
* BulkOutputFormat class introduced in Cassandra 1.1

Cassandra 2.0.11 and later supports the CqlOutputFormat, which is the CQL-compatible version of the BulkOutputFormat class. The CQLOutputFormat acts as a Hadoop-specific OutputFormat. Reduce tasks can store keys (and corresponding bound variable values) as CQL rows (and respective columns) in a given CQL table.

Cassandra 2.0.11 supports using the CQLOutputFormat with Apache Pig.

Running the wordcount example

Wordcount example JARs are located in the `examples` directory of the Cassandra source code installation. There are CQL and legacy examples in the `hadoop_cql3_word_count` and `hadoop_word_count` subdirectories, respectively. Follow instructions in the readme files.

Isolating Hadoop and Cassandra workloads

When you create a keyspace using CQL, Cassandra creates a virtual data center for a cluster, even a one-node cluster, automatically. You assign nodes that run the same type of workload to the same data center. The separate, virtual data centers for different types of nodes segregate workloads running Hadoop from those running Cassandra. Segregating workloads ensures that only one type of workload is active per data center. Separating nodes running a sequential data load, from nodes running any other type of workload, such as Cassandra real-time OLTP queries is a best practice.

Operations

Monitoring Cassandra

Monitoring a Cassandra cluster

Understanding the performance characteristics of your Cassandra cluster is critical to diagnosing issues and planning capacity.

Cassandra exposes a number of statistics and management operations via Java Management Extensions (JMX). Java Management Extensions (JMX) is a Java technology that supplies tools for managing and monitoring Java applications and services. Any statistic or operation that a Java application has exposed as an MBean can then be monitored or manipulated using JMX.

During normal operation, Cassandra outputs information and statistics that you can monitor using JMX-compliant tools, such as:

*   The Cassandra nodetool utility
*   DataStax OpsCenter management console
*   JConsole

Using the same tools, you can perform certain administrative commands and operations such as flushing caches or doing a node repair.

Monitoring using nodetool utility

The nodetool utility is a command-line interface for monitoring Cassandra and performing routine database operations. Included in the Cassandra distribution, nodetool and is typically run directly from an operational Cassandra node.

The nodetool utility supports the most important JMX metrics and operations, and includes other useful commands for Cassandra administration. This utility is commonly used to output a quick summary of the ring and its current state of general health with the status command. For example:

```
paul@ubuntu:~/cassandra-1.2.0$ bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens  Owns   Host ID                               Rack
UN  10.194.171.160  53.98 KB   256     0.8%   a9fa31c7-f3c0-44d1-b8e7-a2628867840c  rack1
UN  10.196.14.48    93.62 KB   256     9.9%   f5bb146c-db51-475c-a44f-9facf2f1ad6e  rack1
DN  10.196.14.239   ?          256     8.2%   b8e6748f-ec11-410d-c94f-9b67d88a28e7  rack1
```

The nodetool utility provides commands for viewing detailed metrics for tables, server metrics, and compaction statistics:

*   `nodetool cfstats` displays statistics for each table and keyspace.
*   `nodetool cfhistograms` provides statistics about a table, including read/write latency, row size, column count, and number of SSTables.
*   `nodetool netstats` provides statistics about network operations and connections.
*   `nodetool tpstats` provides statistics about the number of active, pending, and completed tasks for each stage of Cassandra operations by thread pool.

DataStax OpsCenter

DataStax OpsCenter is a graphical user interface for monitoring and administering all nodes in a Cassandra cluster from one centralized console. DataStax OpsCenter is bundled with DataStax support offerings. You can register for a free version for development or non-production use.

OpsCenter provides a graphical representation of performance trends in a summary view that is hard to obtain with other monitoring tools. The GUI provides views for different time periods as well as the capability to drill down on single data points. Both real-time and historical performance data for a Cassandra or DataStax Enterprise cluster are available in OpsCenter. OpsCenter metrics are captured and stored within Cassandra.
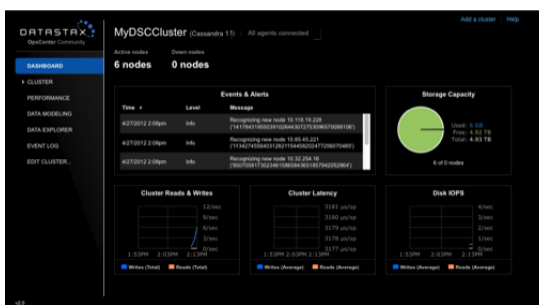
Within OpsCenter you can customize the performance metrics viewed to meet your monitoring needs. Administrators can also perform routine node administration tasks from OpsCenter. Metrics within OpsCenter are divided into three general categories: table metrics, cluster metrics, and OS metrics. For many of the available metrics, you can view aggregated cluster-wide information or view information on a per-node basis.
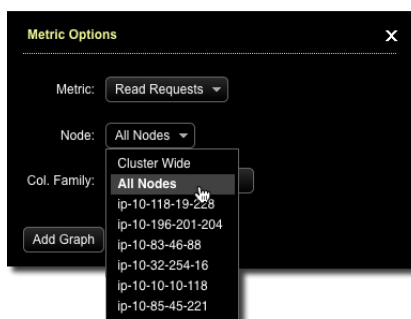
Monitoring using JConsole

JConsole is a JMX-compliant tool for monitoring Java applications such as Cassandra. It is included with Sun JDK 5.0 and higher. JConsole consumes the JMX metrics and operations exposed by Cassandra and displays them in a well-organized GUI. For each node monitored, JConsole provides these six separate tab views:

• Overview

  Displays overview information about the Java VM and monitored values.
• Memory

  Displays information about memory use.
• Threads

  Displays information about thread use.

- Classes

  Displays information about class loading.
- VM Summary

  Displays information about the Java Virtual Machine (VM).
- Mbeans

  Displays information about MBeans.

The Overview and Memory tabs contain information that is very useful for Cassandra developers. The Memory tab allows you to compare heap and non-heap memory usage, and provides a control to immediately perform Java garbage collection.

For specific Cassandra metrics and operations, the most important area of JConsole is the MBeans tab. This tab lists the following Cassandra MBeans:

- org.apache.cassandra.db

  Includes caching, table metrics, and compaction.
- org.apache.cassandra.internal

  Internal server operations such as gossip and hinted handoff.
- org.apache.cassandra.net

  Inter-node communication including FailureDetector, MessagingService and StreamingService.
- org.apache.cassandra.request

  Tasks related to read, write, and replication operations.

When you select an MBean in the tree, its MBeanInfo and MBean Descriptor are displayed on the right, and any attributes, operations or notifications appear in the tree below it. For example, selecting and expanding the org.apache.cassandra.db MBean to view available actions for a table results in a display like the following:
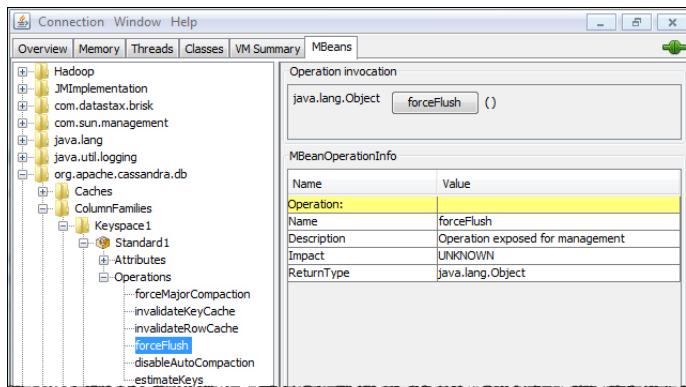
If you choose to monitor Cassandra using JConsole, keep in mind that JConsole consumes a significant amount of system resources. For this reason, DataStax recommends running JConsole on a remote machine rather than on the same host as a Cassandra node.

The JConsole CompactionManagerMBean exposes compaction metrics that can indicate when you need to add capacity to your cluster.

Compaction metrics

Monitoring compaction performance is an important aspect of knowing when to add capacity to your cluster. The following attributes are exposed through CompactionManagerMBean:

Table 7: Compaction Metrics

Attribute	Description
CompletedTasks	Number of completed compactions since the last start of this Cassandra instance
PendingTasks	Number of estimated tasks remaining to perform
ColumnFamilyInProgress	The table currently being compacted. This attribute is null if no compactions are in progress.
BytesTotalInProgress	Total number of data bytes (index and filter are not included) being compacted. This attribute is null if no compactions are in progress.
BytesCompacted	The progress of the current compaction. This attribute is null if no compactions are in progress.

Thread pool and read/write latency statistics

Cassandra maintains distinct thread pools for different stages of execution. Each of the thread pools provide statistics on the number of tasks that are active, pending, and completed. Trends on these pools for increases in the pending tasks column indicate when to add additional capacity. After a baseline is established, configure alarms for any increases above normal in the pending tasks column. Use nodetool tpstats on the command line to view the thread pool details shown in the following table.

Table 8: Compaction Metrics

Thread Pool	Description
AE_SERVICE_STAGE	Shows anti-entropy tasks.
CONSISTENCY-MANAGER	Handles the background consistency checks if they were triggered from the client's consistency level.
FLUSH-SORTER-POOL	Sorts flushes that have been submitted.
FLUSH-WRITER-POOL	Writes the sorted flushes.
GOSSIP_STAGE	Activity of the Gossip protocol on the ring.
LB-OPERATIONS	The number of load balancing operations.
LB-TARGET	Used by nodes leaving the ring.
MEMTABLE-POST-FLUSHER	Memtable flushes that are waiting to be written to the commit log.
MESSAGE-STREAMING-POOL	Streaming operations. Usually triggered by bootstrapping or decommissioning nodes.
MIGRATION_STAGE	Tasks resulting from the call of system_* methods in the API that have modified the schema.
MISC_STAGE	
MUTATION_STAGE	API calls that are modifying data.
READ_STAGE	API calls that have read data.
RESPONSE_STAGE	Response tasks from other nodes to message streaming from this node.

Thread Pool	Description
STREAM_STAGE	Stream tasks from this node.

Read/Write latency metrics

Cassandra tracks latency (averages and totals) of read, write, and slicing operations at the server level through StorageProxyMBean.

Table statistics

For individual tables, ColumnFamilyStoreMBean provides the same general latency attributes as StorageProxyMBean. Unlike StorageProxyMBean, ColumnFamilyStoreMBean has a number of other statistics that are important to monitor for performance trends. The most important of these are:

Table 9: Compaction Metrics

Attribute	Description
MemtableDataSize	The total size consumed by this table's data (not including metadata).
MemtableColumnsCount	Returns the total number of columns present in the memtable (across all keys).
MemtableSwitchCount	How many times the memtable has been flushed out.
RecentReadLatencyMicros	The average read latency since the last call to this bean.
RecentWriterLatencyMicros	The average write latency since the last call to this bean.
LiveSSTableCount	The number of live SSTables for this table.

The recent read latency and write latency counters are important in making sure operations are happening in a consistent manner. If these counters start to increase after a period of staying flat, you probably need to add capacity to the cluster.

You can set a threshold and monitor LiveSSTableCount to ensure that the number of SSTables for a given table does not become too great.

Tuning Bloom filters

Cassandra uses Bloom filters to determine whether an SSTable has data for a particular row. Bloom filters are unused for range scans, but are used for index scans. Bloom filters are probabilistic sets that allow you to trade memory for accuracy. This means that higher Bloom filter attribute settings bloom_filter_fp_chance use less memory, but will result in more disk I/O if the SSTables are highly fragmented. Bloom filter settings range from 0 to 1.0 (disabled). The default value of bloom_filter_fp_chance depends on the compaction strategy. The LeveledCompactionStrategy uses a higher default value (0.1) than the SizeTieredCompactionStrategy or DateTieredCompactionStrategy, which have a default of 0.01. Memory savings are nonlinear; going from 0.01 to 0.1 saves about one third of the memory. SSTables using LCS contain a relatively smaller ranges of keys than those using STCS, which facilitates efficient exclusion of the SSTables even without a bloom filter; however, adding a small bloom filter helps when there are many levels in LCS.

The settings you choose depend the type of workload. For example, to run an analytics application that heavily scans a particular table, you would want to inhibit the Bloom filter on the table by setting it high.

To view the observed Bloom filters false positive rate and the number of SSTables consulted per read use cfstats in the nodetool utility.

Bloom filters are stored off-heap so you don't need include it when determining the -Xmx settings (the maximum memory size that the heap can reach for the JVM).

To change the bloom_filter_attribute on a table, use CQL. For example:

```
ALTER TABLE addamsFamily WITH bloom_filter_fp_chance = 0.1;
```

After updating the value of bloom_filter_fp_chance on a table, Bloom filters need to be regenerated in one of these ways:

* Initiate compaction
* Upgrade SSTables

You do not have to restart Cassandra after regenerating SSTables.

Data caching

Configuring data caches

Cassandra includes integrated caching and distributes cache data around the cluster for you. When a node goes down, the client can read from another cached replica of the data. The integrated architecture also facilitates troubleshooting because there is no separate caching tier, and cached data matches what's in the database exactly. The integrated cache solves the cold start problem by virtue of saving your cache to disk periodically and being able to read contents back in when it restarts—you never have to start with a cold cache.

About the partition key cache

The partition key cache is a cache of the partition index for a Cassandra table. Using the key cache instead of relying on the OS page cache saves CPU time and memory. However, enabling just the key cache results in disk (or OS page cache) activity to actually read the requested data rows.

About the row cache

The row cache is similar to a traditional cache like memcached. When a row is accessed, the entire row is pulled into memory, merging from multiple SSTables if necessary, and cached, so that further reads against that row can be satisfied without hitting disk at all.

Important: Cassandra caches all rows in a partition when reading the partition.

While storing the row cache *off-heap*, Cassandra has to deserialize a partition into heap to read from it. Consequently, use the row cache under these conditions only:

* The partition you will cache is small.
* Your users or applications will read most of the partition all at once.

Do *not* use the row cache unless you fully understand how to prevent misusing the row cache. Misuse exhausts the JVM heap and causes Cassandra to fail.

Typically, you enable either the partition key or row cache for a table--*not* both. The main exception is for caching archive tables that are infrequently read. Disable caching entirely for archive tables.

Enabling and configuring caching

About this task

Use CQL to enable or disable caching by configuring the caching table property. Set parameters in the `cassandra.yaml` file to configure other caching properties:

- Partition key cache size
- Row cache size
- How often Cassandra saves partition key caches to disk
- How often Cassandra saves row caches to disk

Procedure

Set the table caching property in the WITH clause of the table definition. Enclose the table property in single quotation marks. For example, configure Cassandra to disable the partition key cache and the row cache.

```
CREATE TABLE users (
  userid text PRIMARY KEY,
  first_name text,
  last_name text,
)
WITH caching = 'none';
```

How caching works

When both row cache and partition key cache are configured, the row cache returns results whenever possible. In the event of a row cache miss, the partition key cache might still provide a hit that makes the disk seek much more efficient. This diagram depicts two read operations on a table with both caches already populated.
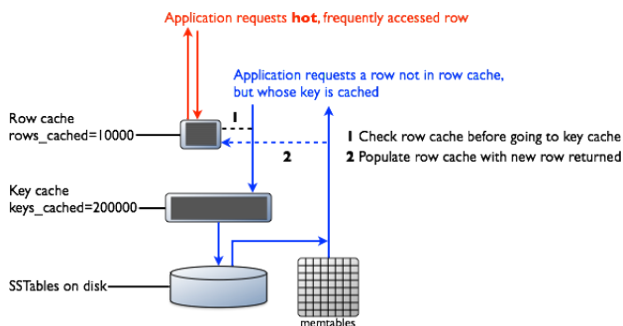
One read operation hits the row cache, returning the requested row without a disk seek. The other read operation requests a row that is not present in the row cache but is present in the partition key cache. After accessing the row in the SSTable, the system returns the data and populates the row cache with this read operation.

Tips for efficient cache use

Some tips for efficient cache use are:

- Store lower-demand data or data with extremely long rows in a table with minimal or no caching.
- Deploy a large number of Cassandra nodes under a relatively light load per node.
- Logically separate heavily-read data into discrete tables.

Cassandra memtables have overhead for index structures on top of the actual data they store. If the size of the values stored in the heavily-read columns is small compared to the number of columns and rows themselves, this overhead can be substantial. Rows having this type of data do not lend themselves to efficient row caching and caching rows can cause serious problems.

Monitoring and adjusting caching

Make changes to cache options in small, incremental adjustments, then monitor the effects of each change using DataStax OpsCenter or the nodetool utility. The output of the `nodetool info` command shows the following row cache and key cache metrics, which are configured in the `cassandra.yaml` file:

- Cache size in bytes
- Capacity in bytes
- Number of hits
- Number of requests
- Recent hit rate
- Duration in seconds after which Cassandra saves the key cache.

For example, on start-up, the information from `nodetool info` might look something like this:

```
Token                 : (invoke with -T/--tokens to see all 256 tokens)
ID                    : 387d15ba-7103-491b-9327-1a691dbb504a
Gossip active         : true
Thrift active         : true
Native Transport active: true
Load                  : 11.35 KB
Generation No         : 1384180190
Uptime (seconds)      : 437
Heap Memory (MB)      : 136.33 / 1996.81
Data Center           : datacenter1
Rack                  : rack1
Exceptions            : 0
Key Cache             : size 360 (bytes), capacity 103809024 (bytes), 15 hits, 19
 requests, 0.789 recent hit rate, 14400 save period in seconds
Row Cache             : size 0 (bytes), capacity 0 (bytes), 0 hits, 0 requests, NaN
 recent hit rate, 0 save period in seconds
```

In the event of high memory consumption, consider tuning data caches.

Configuring memtable throughput

Configuring memtable throughput can improve write performance. Cassandra flushes memtables to disk, creating SSTables when the commit log space threshold has been exceeded. Configure the commit log space threshold per node in the cassandra.yaml. How you tune memtable thresholds depends on your data and write load. Increase memtable throughput under either of these conditions:

- The write load includes a high volume of updates on a smaller set of data.
- A steady stream of continuous writes occurs. This action leads to more efficient compaction.

Allocating memory for memtables reduces the memory available for caching and other internal Cassandra structures, so tune carefully and in small increments.

Configuring compaction

About this task

As discussed earlier, the compaction process merges keys, combines columns, evicts tombstones, consolidates SSTables, and creates a new index in the merged SSTable.

In the cassandra.yaml file, you configure these global compaction parameters:

- snapshot_before_compaction
- in_memory_compaction_limit_in_mb
- multithreaded_compaction
- compaction_preheat_key_cache
- concurrent_compactors
- compaction_throughput_mb_per_sec

The compaction_throughput_mb_per_sec parameter is designed for use with large partitions because compaction is throttled to the specified total throughput across the entire system.

Cassandra provides a start-up option for testing compaction strategies without affecting the production workload.

Using CQL, you configure a compaction strategy:

- SizeTieredCompactionStrategy (STCS)
- DateTieredCompactionStrategy (DTCS)
- LeveledCompactionStrategy (LCS)

Note: When using DTCS disabling read repair is recommended. Use full repair as necessary.

To configure the compaction strategy property and CQL compaction subproperties, such as the maximum number of SSTables to compact and minimum SSTable size, use CREATE TABLE or ALTER TABLE.

Procedure

1. Update a table to set the compaction strategy using the ALTER TABLE statement.

```
ALTER TABLE users WITH
  compaction = { 'class' :  'LeveledCompactionStrategy'  }
```

2. Change the compaction strategy property to SizeTieredCompactionStrategy and specify the minimum number of SSTables to trigger a compaction using the CQL min_threshold attribute.

```
ALTER TABLE users
  WITH compaction =
  {'class' : 'SizeTieredCompactionStrategy', 'min_threshold' : 6 }
```

Results

You can monitor the results of your configuration using compaction metrics, see Compaction metrics.

Compression

Compression maximizes the storage capacity of Cassandra nodes by reducing the volume of data on disk and disk I/O, particularly for read-dominated workloads. Cassandra quickly finds the location of rows in the SSTable index and decompresses the relevant row chunks.

Write performance is not negatively impacted by compression in Cassandra as it is in traditional databases. In traditional relational databases, writes require overwrites to existing data files on disk. The

database has to locate the relevant pages on disk, decompress them, overwrite the relevant data, and finally recompress. In a relational database, compression is an expensive operation in terms of CPU cycles and disk I/O. Because Cassandra SSTable data files are immutable (they are not written to again after they have been flushed to disk), there is no recompression cycle necessary in order to process writes. SSTables are compressed only once when they are written to disk. Writes on compressed tables can show up to a 10 percent performance improvement.

When to compress data

Compression is best suited for tables that have many rows and each row has the same columns, or at least as many columns, as other rows. For example, a table containing user data such as username, email, and state, is a good candidate for compression. The greater the similarity of the data across rows, the greater the compression ratio and gain in read performance.

A table that has rows of different sets of columns is not well-suited for compression. Dynamic tables do not yield good compression ratios.

Don't confuse table compression with compact storage of columns, which is used for backward compatibility of old applications with CQL.

Depending on the data characteristics of the table, compressing its data can result in:

- 2x-4x reduction in data size
- 25-35% performance improvement on reads
- 5-10% performance improvement on writes

After configuring compression on an existing table, subsequently created SSTables are compressed. Existing SSTables on disk are not compressed immediately. Cassandra compresses existing SSTables when the normal Cassandra compaction process occurs. Force existing SSTables to be rewritten and compressed by using nodetool upgradesstables (Cassandra 1.0.4 or later) or nodetool scrub.

Configuring compression

About this task

You configure a table property and subproperties to manage compression. The CQL table properties documentation describes the types of compression options that are available. Compression is enabled by default in Cassandra 1.1 and later.

Procedure

1. Disable compression, using CQL to set the compression parameters to an empty string.

```
CREATE TABLE DogTypes (
           block_id uuid,
           species text,
           alias text,
           population varint,
           PRIMARY KEY (block_id)
         )
           WITH compression = { 'sstable_compression' : '' };
```

2. Enable compression on an existing table, using ALTER TABLE to set the compression algorithm sstable_compression to LZ4Compressor (Cassandra 1.2.2 and later), SnappyCompressor, or DeflateCompressor.

```
CREATE TABLE DogTypes (
           block_id uuid,
           species text,
           alias text,
           population varint,
```

```
            PRIMARY KEY (block_id)
        )
        WITH compression = { 'sstable_compression' : 'LZ4Compressor' };
```

3. Change compression on an existing table, using ALTER TABLE and setting the compression algorithm sstable_compression to DeflateCompressor.

```
ALTER TABLE CatTypes
  WITH compression = { 'sstable_compression' : 'DeflateCompressor',
'chunk_length_kb' : 64 }
```

You tune data compression on a per-table basis using CQL to alter a table.

Testing compaction and compression

About this task

Write survey mode is a Cassandra startup option for testing new compaction and compression strategies. In write survey mode, you can test out new compaction and compression strategies on that node and benchmark the write performance differences, without affecting the production cluster.

Write survey mode adds a node to a database cluster. The node accepts all write traffic as if it were part of the normal Cassandra cluster, but the node does not officially join the ring.

Also use write survey mode to try out a new Cassandra version. The nodes you add in write survey mode to a cluster must be of the same major release version as other nodes in the cluster. The write survey mode relies on the streaming subsystem that transfers data between nodes in bulk and differs from one major release to another.

If you want to see how read performance is affected by modifications, stop the node, bring it up as a standalone machine, and then benchmark read operations on the node.

Procedure

Enable write survey mode by starting a Cassandra node using the write_survey option.

```
bin/cassandra – Dcassandra.write_survey=true
```

This example shows how to start a tarball installation of Cassandra.

Tuning Java resources

Consider tuning Java resources in the event of a performance degradation or high memory consumption.

There are two files that control environment settings for Cassandra:

• conf/cassandra-env.sh

Java Virtual Machine (JVM) configuration settings

• bin/cassandra-in.sh

Sets up Cassandra environment variables such as CLASSPATH and JAVA_HOME.

Heap sizing options

If you decide to change the Java heap sizing, both MAX_HEAP_SIZE and HEAP_NEWSIZE should should be set together in conf/cassandra-env.sh.

• MAX_HEAP_SIZE

Sets the maximum heap size for the JVM. The same value is also used for the minimum heap size. This allows the heap to be locked in memory at process start to keep it from being swapped out by the OS.

• HEAP_NEWSIZE

The size of the young generation. The larger this is, the longer GC pause times will be. The shorter it is, the more expensive GC will be (usually). A good guideline is 100 MB per CPU core.

Tuning the Java heap

Because Cassandra is a database, it spends significant time interacting with the operating system's I/O infrastructure through the JVM, so a well-tuned Java heap size is important. Cassandra's default configuration opens the JVM with a heap size that is based on the total amount of system memory:

System Memory	Heap Size
Less than 2GB	1/2 of system memory
2GB to 4GB	1GB
Greater than 4GB	1/4 system memory, but not more than 8GB

Many users new to Cassandra are tempted to turn up Java heap size too high, which consumes the majority of the underlying system's RAM. In most cases, increasing the Java heap size is actually detrimental for these reasons:

• In most cases, the capability of Java to gracefully handle garbage collection above 8GB quickly diminishes.
• Modern operating systems maintain the OS page cache for frequently accessed data and are *very* good at keeping this data in memory, but can be prevented from doing its job by an elevated Java heap size.

If you have more than 2GB of system memory, which is typical, keep the size of the Java heap relatively small to allow more memory for the page cache.

Some Solr users have reported that increasing the stack size improves performance under Tomcat. To increase the stack size, uncomment and modify the default -Xss128k setting in the cassandra-env.sh file. Also, decreasing the memtable space to make room for Solr caches might improve performance. Modify the memtable space using the memtable_total_space_in_mb property in the cassandra.yaml file.

Because MapReduce runs outside the JVM, changes to the JVM do not affect Analytics/Hadoop operations directly.

How Cassandra uses memory

Using a java-based system like Cassandra, you can typically allocate about 8GB of memory on the heap before garbage collection pause time starts to become a problem. Modern machines have much more memory than that and Cassandra can make use of additional memory as page cache when files on disk are accessed. Allocating more than 8GB of memory on the heap poses a problem due to the amount of Cassandra metadata about data on disk. The Cassandra metadata resides in memory and is proportional to total data. Some of the components grow proportionally to the size of total memory.

In Cassandra 1.2 and later, the Bloom filter and compression offset map that store this metadata reside off-heap, greatly increasing the capacity per node of data that Cassandra can handle efficiently. In Cassandra 2.0, the partition summary also resides off-heap.

About the off-heap row cache

Cassandra can store cached rows in native memory, outside the Java heap. This results in both a smaller per-row memory footprint and reduced JVM heap requirements, which helps keep the heap size in the sweet spot for JVM garbage collection performance.

Tuning Java garbage collection

Cassandra's GCInspector class logs information about garbage collection whenever a garbage collection takes longer than 200ms. Garbage collections that occur frequently and take a moderate length of time to complete (such as ConcurrentMarkSweep taking a few seconds), indicate that there is a lot of garbage collection pressure on the JVM. Remedies include adding nodes, lowering cache sizes, or adjusting the JVM options regarding garbage collection.

JMX options

Cassandra exposes a number of statistics and management operations via Java Management Extensions (JMX). Java Management Extensions (JMX) is a Java technology that supplies tools for managing and monitoring Java applications and services. Any statistic or operation that a Java application has exposed as an MBean can then be monitored or manipulated using JMX. JConsole, the nodetool utility, and DataStax OpsCenter are examples of JMX-compliant management tools.

By default, you can modify the following properties in the conf/cassandra-env.sh file to configure JMX to listen on port 7199 without authentication.

- com.sun.management.jmxremote.port

  The port on which Cassandra listens from JMX connections.
- com.sun.management.jmxremote.ssl

  Enable/disable SSL for JMX.
- com.sun.management.jmxremote.authenticate

  Enable/disable remote authentication for JMX.
- -Djava.rmi.server.hostname

  Sets the interface hostname or IP that JMX should use to connect. Uncomment and set if you are having trouble connecting.

Purging gossip state on a node

About this task

Gossip information is persisted locally by each node to use immediately on node restart without having to wait for gossip communications.

Procedure

In the unlikely event you need to correct a problem in the gossip state:

1. Using MX4J or JConsole, connect to the node's JMX port and then use the JMX method `Gossiper.unsafeAssassinateEndpoints(`*ip_address*`)` to assassinate the problem node.

   This takes a few seconds to complete so wait for confirmation that the node is deleted.
2. If the JMX method above doesn't solve the problem, stop your client application from sending writes to the cluster.
3. Take the entire cluster offline:

   a) Drain each node.

   ```
   $ nodetool options drain
   ```
   b) Stop each node:

   - Packaged installs:

     ```
     $ sudo service cassandra stop
     ```

- Tarball installs:

  ```
  $ sudo service cassandra stop
  ```

4. Clear the data from the `peers` directory:

```
$ sudo rm -r /var/lib/cassandra/data/system/peers/*
```

5. Clear the gossip state when the node starts:

- For tarball installations, you can use a command line option or edit the `cassandra-env.sh`. To use the command line:

  ```
  $ install_location/bin/cassandra -Dcassandra.load_ring_state=false
  ```

- For package installations or if you are not using the command line option above, add the following line to the `cassandra-env.sh` file:

  ```
  JVM_OPTS="$JVM_OPTS -Dcassandra.load_ring_state=false"
  ```

  - Packaged installs: `/usr/share/cassandra/cassandra-env.sh`
  - Tarball installs: `install_location/conf/cassandra-env.sh`

6. Bring the cluster online one node at a time, starting with the seed nodes.

- Packaged installs:

  ```
  $ sudo service cassandra start
  ```

- Tarball installs:

  ```
  $ cd install_location
  $ bin/cassandra
  ```

What to do next

Remove the line you added in the `cassandra-env.sh` file.

Repairing nodes

Node repair makes data on a replica consistent with data on other nodes. To understand how repair works and the information described in this topic, please read the blog article Advanced repair techniques.

The nodetool repair command repairs inconsistencies across all of the replicas for a given range of data. Run repair in these situations:

- As a best practice, you should schedule repairs weekly.

  Note: If deletions never occur, you should still schedule regular repairs. Be aware that setting a column to null is a delete.
- During node recovery. For example, when bringing a node back into the cluster after a failure.
- On nodes containing data that is not read frequently.
- To update data on a node that has been down.

Guidelines for running routine node repair include:

- The hard requirement for routine repair frequency is the value of gc_grace_seconds. Run a repair operation at least once on each node within this time period. Following this important guideline ensures that deletes are properly handled in the cluster.
- Schedule regular repair operations for low-usage hours.
- To minimize impact, use nodetool repair and do not invoke more than one repair at a time.

- In systems that seldom delete or overwrite data, you can raise the value of gc_grace with minimal impact to disk space. This allows wider intervals for scheduling repair operations with the nodetool utility.

Repair requires intensive disk I/O. This occurs because of the validation compaction used for building the Merkle tree. To mitigate heavy disk usage:

- Use the nodetool compaction throttling options (setcompactionthroughput and setcompactionthreshold).
- Use `nodetool repair`.

    The repair command takes a snapshot of each replica immediately and then sequentially repairs each replica from the snapshots. For example, if you have RF=3 and A, B and C represents three replicas, this command takes a snapshot of each replica immediately and then sequentially repairs each replica from the snapshots (A<->B, A<->C, B<->C) instead of repairing A, B, and C all at once. This allows the dynamic snitch to maintain performance for your application via the other replicas, because at least one replica in the snapshot is not undergoing repair.

    Recall that snapshots are hardlinks to existing SSTables, immutable, and require almost no disk space. This means that for any given replica set, only one replica at a time performs the validation compaction. This allows the dynamic snitch to maintain performance for your application via the other replicas.

    Note: Using the nodetool repair -pr (–partitioner-range) option repairs only the primary range for that node, the other replicas for that range still have to perform the Merkle tree calculation, causing a validation compaction. Because all the replicas are compacting at the same time, all the nodes may be slow to respond for that portion of the data.

This happens because the Merkle trees don't have infinite resolution and Cassandra makes a tradeoff between the size and space. Currently, Cassandra uses a fixed depth of 15 for the tree (32K leaf nodes). For a node containing a million partitions with one damaged partition, about 30 partitions are streamed, which is the number that fall into each of the *leaves* of the tree. Of course, the problem gets worse when more partitions exist per node, and results in a lot of disk space usage and needless compaction.

To mitigate overstreaming, you can use subrange repair. Subrange repair allows for repairing only a portion of the data belonging to the node. Because the Merkle tree precision is fixed, this effectively increases the overall precision.

To use subrange repair:

1. Use the Java `describe_splits` call to ask for a split containing 32K partitions.
2. Iterate throughout the entire range incrementally or in parallel. This completely eliminates the overstreaming behavior and wasted disk usage overhead.
3. Pass the tokens you received for the split to the `nodetool repair` -st (–start-token) and -et (–end-token) options.
4. Pass the -local (–in-local-dc) option to nodetool to repair only within the local data center. This reduces the cross data-center transfer load.

Adding or removing nodes, data centers, or clusters

Adding nodes to an existing cluster

About this task

Virtual nodes (vnodes) greatly simplify adding nodes to an existing cluster:

- Calculating tokens and assigning them to each node is no longer required.
- Rebalancing a cluster is no longer necessary because a node joining the cluster assumes responsibility for an even portion of the data.

For a detailed explanation about how this works, see Virtual nodes.

If you are using racks, you can safely bootstrap two nodes at a time when both nodes are on the same rack.

Note: If you do not use vnodes, see Adding or replacing single-token nodes.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See Installing prior releases.

1. Install Cassandra on the new nodes, but do not start Cassandra.

   If you used the Debian install, Cassandra starts automatically and you must and stop the node and clear the data.

2. Set the following properties in the `cassandra.yaml` and, depending on the snitch, the `cassandra-topology.properties` or `cassandra-rackdc.properties` configuration files:

   - auto_bootstrap - If this option has been set to false, you must set it to true. This option is not listed in the default `cassandra.yaml` configuration file and defaults to true.
   - cluster_name - The name of the cluster the new node is joining.
   - listen_address/broadcast_address - May usually be left blank. Otherwise, use IP address or host name that other Cassandra nodes use to connect to the new node.
   - endpoint_snitch - The snitch Cassandra uses for locating nodes and routing requests.
   - num_tokens - The number of vnodes to assign to the node. If the hardware capabilities vary among the nodes in your cluster, you can assign a proportional number of vnodes to the larger machines.
   - seed_provider - The -seeds list in this setting determines which nodes the new node should contact to learn about the cluster and establish the gossip process.

     Note: Seed nodes cannot bootstrap. Make sure the new node is not listed in the -seeds list. **Do not make all nodes seed nodes.** Please read Internode communications (gossip).
   - Change any other non-default settings you have made to your existing cluster in the `cassandra.yaml` file and `cassandra-topology.properties` or `cassandra-rackdc.properties` files. Use the `diff` command to find and merge (by head) any differences between existing and new nodes.

3. Start Cassandra on each new node. Allow two minutes between node initializations. You can monitor the startup and data streaming process using nodetool netstats.

4. After all new nodes are running, run nodetool cleanup on each of the previously existing nodes to remove the keys no longer belonging to those nodes. Wait for cleanup to complete on one node before doing the next.

   Cleanup may be safely postponed for low-usage hours.

Adding a data center to a cluster

About this task

Steps to add a data center to an existing cluster.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See Installing prior releases.

1. Ensure that you are using NetworkTopologyStrategy for all of your keyspaces.

2. For each node, set the following properties in the cassandra.yaml file:.

   a) Add (or edit) auto_bootstrap: false.

By default, this setting is true and not listed in the `cassandra.yaml` file. Setting this parameter to false prevents the new nodes from attempting to get all the data from the other nodes in the data center. When you run nodetool rebuild in the last step, each node is properly mapped.

b) Set other properties, such as -seeds and endpoint_snitch, to match the cluster settings.

For more guidance, see Initializing a multiple node cluster (multiple data centers).

Note: Do not make all nodes seeds, see Internode communications (gossip).

c) If you want to enable vnodes, set num_tokens.

The recommended value is 256. Do not set the initial_token parameter.

3. Update the relevant property file for snitch used on all servers to include the new nodes. You do not need to restart.

- GossipingPropertyFileSnitch: `cassandra-rackdc.properties`
- PropertyFileSnitch: `cassandra-topology.properties`

4. Ensure that your clients are configured correctly for the new cluster:

- If your client uses the DataStax Java, C#, or Python driver, set the load-balancing policy to `DCAwareRoundRobinPolicy` (Java, C#, Python).
- If you are using another client such as Hector, make sure it does not auto-detect the new nodes so that they aren't contacted by the client until explicitly directed. For example if you are using Hector, use `sethostConfig.setAutoDiscoverHosts(false);`. If you are using Astyanax, use `ConnectionPoolConfigurationImpl.setLocalDatacenter("<data center name">)` to ensure you are connecting to the specified data center.
- If you are using Astyanax 2.x, with integration with the DataStax Java Driver 2.0, you can set the load-balancing policy to `DCAwareRoundRobinPolicy` by calling `JavaDriverConfigBuilder.withLoadBalancingPolicy()`.

```
AstyanaxContext<Keyspace> context = new AstyanaxContext.Builder()
    ...
    .withConnectionPoolConfiguration(new JavaDriverConfigBuilder()
        .withLoadBalancingPolicy(new TokenAwarePolicy(new
DCAwareRoundRobinPolicy())))
        .build())
    ...
```

5. If using a QUORUM consistency level for reads or writes, check the LOCAL_QUORUM or EACH_QUORUM consistency level to see if the level meets your requirements for multiple data centers.

6. Start Cassandra on the new nodes.

7. After all nodes are running in the cluster:

a) Change the keyspace properties to specify the desired replication factor for the new data center.

For example, set strategy options to DC1:2, DC2:2.

For more information, see ALTER KEYSPACE.

b) Run nodetool rebuild specifying the existing data center on all nodes in the new data center:

`nodetool rebuild -- name_of_existing_data_center`

Otherwise, requests to the new data center with LOCAL_ONE or ONE consistency levels may fail if the existing data centers are not completely in-sync.

You can run rebuild on one or more nodes at the same time. The choices depends on whether your cluster can handle the extra IO and network pressure of running on multiple nodes. Running on one node at a time has the least impact on the existing cluster.

Attention: If you don't specify the existing data center in the command line, the new nodes will appear to rebuild successfully, but will not contain any data.

8. For each new node, change to true or remove auto_bootstrap: false in the `cassandra.yaml` file.

   Returns this parameter to its normal setting so the nodes can get all the data from the other nodes in the data center if restarted.

Replacing a dead node

About this task

Steps to replace a node that has died for some reason, such as hardware failure.

You must prepare and start the replacement node, integrate it into the cluster, and then remove the dead node. If the node is a seed node, see Replacing a dead seed node.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See Installing prior releases.

1. Confirm that the node is dead using nodetool status:

   The nodetool command shows a down status for the dead node (DN):

2. Note the `Address` of the dead node; it is used in step 5.
3. Install Cassandra on the new node, but do not start Cassandra.

   If you used the Debian/Ubuntu install, Cassandra starts automatically and you must and stop the node and clear the data.

4. Set the following properties in the `cassandra.yaml` and, depending on the snitch, the `cassandra-topology.properties` or `cassandra-rackdc.properties` configuration files:

   - auto_bootstrap - If this option has been set to false, you must set it to true. This option is not listed in the default `cassandra.yaml` configuration file and defaults to true.
   - cluster_name - The name of the cluster the new node is joining.
   - listen_address/broadcast_address - May usually be left blank. Otherwise, use IP address or host name that other Cassandra nodes use to connect to the new node.
   - endpoint_snitch - The snitch Cassandra uses for locating nodes and routing requests.
   - num_tokens - The number of vnodes to assign to the node. If the hardware capabilities vary among the nodes in your cluster, you can assign a proportional number of vnodes to the larger machines.
   - seed_provider - The -seeds list in this setting determines which nodes the new node should contact to learn about the cluster and establish the gossip process.

     Note: Seed nodes cannot bootstrap. Make sure the new node is not listed in the -seeds list. **Do not make all nodes seed nodes.** Please read Internode communications (gossip).
   - Change any other non-default settings you have made to your existing cluster in the `cassandra.yaml` file and `cassandra-topology.properties` or `cassandra-rackdc.properties` files. Use the `diff` command to find and merge (by head) any differences between existing and new nodes.

5. Start the replacement node with the replace_address option:

   - Packaged installs: Add the following option to `/usr/share/cassandra/cassandra-env.sh` file:

     `JVM_OPTS="$JVM_OPTS -Dcassandra.replace_address=address_of_dead_node`
   - Tarball installs: Start Cassandra with this option:

```
$ sudo bin/cassandra –Dcassandra.replace_address=address_of_dead_node
```

6. If using a packaged install, after the new node finishes bootstrapping, remove the option you added in step 5.

What to do next

Remove the old node's IP address from the `cassandra-topology.properties` or `cassandra-rackdc.properties` file

Caution: Wait at least 72 hours to ensure that old node information is removed from gossip. If removed from the property file too soon, problems may result.

Replacing a dead seed node

About this task

Steps to replace a seed node.

Because Cassandra doesn't allow a seed node to be bootstrapped, use the following steps for replacing the dead node:

Procedure

1. Promote an existing node to a seed node by adding its IP address to -seeds list and remove (demote) the IP address of the dead seed node from the cassandra.yaml file for each node in the cluster.

2. Replace the dead node, as described in Replacing a dead node.

Replacing a running node

About this task

Steps to replace a node with a new node, such as when updating to newer hardware or performing proactive maintenance.

You must prepare and start the replacement node, integrate it into the cluster, and then decommission the old node.

Note: To change the IP address of a node, simply change the IP of node and then restart Cassandra. If you change the IP address of a seed node, you must update the - seeds parameter in the seed_provider for each node in the `cassandra.yaml` file.

Procedure

Be sure to install the same version of Cassandra as is installed on the other nodes in the cluster. See Installing prior releases.

1. Prepare and start the replacement node, as described in Adding nodes to an existing cluster.

   Note: If not using vnodes, see Adding or replacing single-token nodes.

2. Confirm that the replacement node is alive:

   • Run nodetool ring if not using vnodes.
   • Run nodetool status if using vnodes.

   The status should show:

   • nodetool ring: Up
   • nodetool status: UN

3. Note the `Host ID` of the node; it is used in the next step.

4. Using the Host ID of the original node, decommission the original node from the cluster using the nodetool decommission command.

Moving a node from one rack to another

A common task is moving a node from one rack to another. For example, when using GossipPropertyFileSnitch, a common error is mistakenly placing a node in the wrong rack. To correct the error, use one of the following procedures.

- The preferred method is to decommission the node and re-add it to the correct rack and data center.

  - This method takes longer to complete than the alternative method. Data is moved that the decommissioned node doesn't need anymore. Then the node gets new data while bootstrapping. The alternative method does both operations simultaneously.

- An alternative method is to update the node's topology, restart the node. Once the node is up, run a full repair on the cluster.

  Note: This method is not preferred because until the repair is completed. the node might blindly handle requests for data the node doesn't yet have.

Decommissioning a data center

About this task

Steps to properly remove a data center so no information is lost.

Procedure

1. Make sure no clients are still writing to any nodes in the data center.

2. Run a full repair with nodetool repair.

   This ensures that all data is propagated from the data center being decommissioned.

3. Change all keyspaces so they no longer reference the data center being removed.

4. Run nodetool decommission on every node in the data center being removed.

Removing a node

About this task

Use these instructions when you want to remove nodes to reduce the size of your cluster, not for replacing a dead node.

.

Attention: If you are not using virtual nodes (vnodes), you must rebalance the cluster.

Procedure

1. Check whether the node is up or down using nodetool status:

   The nodetool command shows the status of the node (UN=up, DN=down):

   ```
   paul@ubuntu:~/cassandra-1.2.0$ bin/nodetool status
   Datacenter: datacenter1
   =======================
   Status=Up/Down
   |/ State=Normal/Leaving/Joining/Moving
   --  Address         Load      Tokens  Owns   Host ID                               Rack
   UN  10.194.171.160  53.98 KB  256     0.8%   a9fa31c7-f3c0-44d1-b8e7-a2628867840c  rack1
   UN  10.196.14.48    93.62 KB  256     9.9%   f5bb146c-db51-475c-a44f-9facf2f1ad6e  rack1
   DN  10.196.14.239   ?         256     8.2%   null
   ```

2. If the node is up, run nodetool decommission.

   This assigns the ranges that the node was responsible for to other nodes and replicates the data appropriately.

   Use nodetool netstats to monitor the progress.

3. If the node is down:

   - If the cluster uses vnodes, remove the node using the nodetool removenode command.
   - If the cluster does not use vnodes, before running the nodetool removenode command, adjust your tokens to evenly distribute the data across the remaining nodes to avoid creating a hot spot.

Switching snitches

About this task

Because snitches determine how Cassandra distributes replicas, the procedure to switch snitches depends on whether or not the topology of the cluster will change:

- If data has not been inserted into the cluster, there is no change in the network topology. This means that you only need to set the snitch; no other steps are necessary.
- If data has been inserted into the cluster, it's possible that the topology has changed and you will need to perform additional steps.

A change in topology means that there is a change in the data centers and/or racks where the nodes are placed. Topology changes may occur when the replicas are placed in different places by the new snitch. Specifically, the replication strategy places the replicas based on the information provided by the new snitch. The following examples demonstrate the differences:

- **No topology change**

  Suppose you have 5 nodes using the SimpleSnitch in a single data center and you change to 5 nodes in 1 data center and 1 rack using a network snitch such as the GossipingPropertyFileSnitch.

- **Topology change**

  Suppose you have 5 nodes using the SimpleSnitch in a single data center and you change to 5 nodes in 2 data centers using the PropertyFileSnitch.

  Note: If splitting from one data center to two, you need to change the schema for the keyspace that are splitting. Additionally, the data center names must change accordingly.

- **Topology change**

  Suppose you have 5 nodes using the SimpleSnitch in a single data center and you change to 5 nodes in 1 data center and 2 racks using the RackInferringSnitch.

The configuration files for snitches are located in:

- Tarball installs: *install_location*/conf
- Packaged installs: /etc/cassandra

Procedure

1. Create a properties file with data center and rack information.

   - cassandra-rackdc.properties - GossipingPropertyFileSnitch Ec2Snitch and Ec2MultiRegionSnitch only
   - cassandra-topology.properties - all other network snitches

2. Copy the cassandra-topology.properties or cassandra-rackdc.properties file to the Cassandra configuration directory on all the cluster's nodes. They won't be used until the new snitch is enabled.

3. Change the snitch for each node in the cluster in the node's `cassandra.yaml` file. For example:

   `endpoint_snitch: GossipingPropertyFileSnitch`

4. If the topology has not changed, you can restart each node one at a time.

   Any change in the `cassandra.yaml` file requires a node restart.

5. If the topology of the network has changed:
   a) Shut down all the nodes, then restart them.
   b) Run a sequential repair and nodetool cleanup on each node.

Edge cases for transitioning or migrating a cluster

About this task

The information in this topic is intended for the following types of scenarios (without any interruption of service):

- Transition a cluster on EC2 to a cluster on Amazon virtual private cloud (VPC).
- Migrate from a cluster when the network separates the current cluster from the future location.
- Migrate from an early Cassandra cluster to a recent major version.

Procedure

The following method ensures that if something goes wrong with the new cluster, you still have the existing cluster until you no longer need it.

1. Set up and configure the new cluster as described in Provisioning a new cluster.

   Note: If you're not using vnodes, be sure to configure the token ranges in the new nodes to match the ranges in the old cluster.

2. Set up the schema for the new cluster using CQL.

3. Configure your client to write to both clusters.

   Depending on how the writes are done, code changes may be needed. Be sure to use identical consistency levels.

4. Ensure that the data is flowing to the new nodes so you won't have any gaps when you copy the snapshots to the new cluster in step 6.

5. Snapshot the old EC2 cluster.

6. Copy the data files from your keyspaces to the nodes.

   - If not using vnodes and the if the node ratio is 1:1, it's simpler and more efficient to simply copy the data files to their matching nodes.
   - If the clusters are different sizes or if you are using vnodes, use the Cassandra bulk loader (sstableloader) (sstableloader).

7. You can either switch to the new cluster all at once or perform an incremental migration.

   For example, to perform an incremental migration, you can set your client to designate a percentage of the reads that go to the new cluster. This allows you to test the new cluster before decommissioning the old cluster.

8. Decommission the old cluster:
   a) Remove the cluster from the OpsCenter.
   b) Remove the nodes.

Adding or replacing single-token nodes

About this task

This topic applies only to clusters using single-token architecture, not vnodes.

About adding Capacity to an Existing Cluster

Cassandra allows you to add capacity to a cluster by introducing new nodes to the cluster in stages and by adding an entire data center. When a new node joins an existing cluster, it needs to know:

- Its position in the ring and the range of data it is responsible for, which is assigned by the initial_token and the partitioner.
- The seed nodes to contact for learning about the cluster and establish the gossip process.
- The name of the cluster it is joining and how the node should be addressed within the cluster.
- Any other non-default settings made to cassandra.yaml on your existing cluster.

When you add one or more nodes to a cluster, you must calculate the tokens for the new nodes. Use one of the following approaches:

Add capacity by doubling the cluster size

Adding capacity by doubling (or tripling or quadrupling) the number of nodes is less complicated when assigning tokens. Existing nodes can keep their existing token assignments, and new nodes are assigned tokens that bisect (or trisect) the existing token ranges. For example, when you generate tokens for six nodes, three of the generated token values will be the same as if you generated for three nodes. To clarify, you first obtain the token values that are already in use, and then assign the newly calculated token values to the newly added nodes.

Recalculate new tokens for all nodes and move nodes around the ring

When increases capacity by a non-uniform number of nodes, you must recalculate tokens for the entire cluster, and then use nodetool move to assign the new tokens to the existing nodes. After all nodes are restarted with their new token assignments, run a nodetool cleanup to remove unused keys on all nodes. These operations are resource intensive and should be done during low-usage times.

Add one node at a time and leave the initial_token property empty

When the initial_token is empty, Cassandra splits the token range of the heaviest loaded node and places the new node into the ring at that position. This approach is unlikely to result in a perfectly balanced ring, but will alleviate hot spots.

Adding Nodes to a Cluster

1. Install Cassandra on the new nodes, but do not start them.
2. Calculate the tokens for the nodes based on the expansion strategy you are using the Token Generating Tool. *You can skip this step if you want the new nodes to automatically pick a token range when joining the cluster.*
3. Set the cassandra.yaml for the new nodes.
4. Set the initial_token according to your token calculations (or leave it unset if you want the new nodes to automatically pick a token range when joining the cluster).
5. Start Cassandra on each new node. Allow **two** minutes between node initializations. You can monitor the startup and data streaming process using nodetool netstats.
6. After the new nodes are fully bootstrapped, some token assignments must be changed on the old nodes to balance the token assignment across the cluster. Assign the new initial_token property value to all nodes that require new tokens, and run `nodetool move` *`new_token`* one node at a time.
7. After all nodes have their new tokens assigned, run nodetool cleanup one node at a time for each node. Wait for cleanup to complete before doing the next node. This step removes the keys that no longer belong to the previously existing nodes.

Note: Cleanup may be safely postponed for low-usage hours.

Adding a Data Center to a Cluster

Before starting this procedure, please read the guidelines in Adding Capacity to an Existing Cluster above.

1. Ensure that you are using NetworkTopologyStrategy for all of your keyspaces.
2. For each new node, edit the configuration properties in the cassandra.yaml file:

   - Set `auto_bootstrap` to `False`.
   - Set the `initial_token`. Be sure to offset the tokens in the new data center, see Generating tokens.
   - Set the `cluster name`.
   - Set any other non-default settings.
   - Set the seed lists. Every node in the cluster must have the same list of seeds and include at least one node from each data center. Typically one to three seeds are used per data center.

3. Update either the `cassandra-topology.properties` (PropertyFileSnitch) or `cassandra-rackdc.properties` (GossipingPropertyFileSnitch) on all servers to include the new nodes. You do not need to restart.

   The location of the `conf` directory depends on the type of installation:

   - Packaged installs: `/etc/cassandra/conf`
   - Tarball installs: `install_location/conf`

4. Ensure that your client does not auto-detect the new nodes so that they aren't contacted by the client until explicitly directed. For example in Hector, set

   ```
   hostConfig.setAutoDiscoverHosts(false);
   ```

5. If using a QUORUM consistency level for reads or writes, check the LOCAL_QUORUM or EACH_QUORUM consistency level to make sure that the level meets the requirements for multiple data centers.
6. Start the new nodes.
7. After all nodes are running in the cluster:

   a. Change the replication factor for your keyspace for the expanded cluster.
   b. Run nodetool rebuild on each node in the new data center.

Replacing a Dead Node

1. Confirm that the node is dead using the nodetool ring command on any live node in the cluster.

   The nodetool ring command shows a *Down* status for the token value of the dead node:

   ```
   $ nodetool ring -h localhost

   Address          DC           Rack     Status State   Load         Owns     Token
   10.46.123.11     datacenter1  rack1    Up     Normal  179.58 KB    16.67%   0
   10.46.123.12     datacenter1  rack1    Down   Normal  315.21 KB    16.67%   28356863910078205288614550619314017621
   10.46.123.13     datacenter1  rack1    Up     Normal  267.71 KB    16.67%   56713727820156418577229101238628035242
   10.46.123.14     datacenter1  rack1    Up     Normal  315.21 KB    16.67%   85070591730234615865843651857942052863
   10.46.123.15     datacenter1  rack1    Up     Normal  292.36 KB    16.67%   113427455640312821155448202477256070485
   10.46.123.16     datacenter1  rack1    Up     Normal  300.02 KB    16.67%   141784319550391026443072753096570088106
   ```

2. Install Cassandra on the replacement node.
3. Remove any preexisting Cassandra data on the replacement node:

   ```
   $ sudo rm -rf /var/lib/cassandra/*
   ```

4. Set `auto_bootstrap: true`. (If `auto_bootstrap` is not in the cassandra.yaml file, it automatically defaults to `true`.)
5. Set the `initial_token` in the `cassandra.yaml` file to the value of the dead node's token -1. Using the value from the above graphic, this is `28356863910078205288614550619314017621-1`:

   ```
   initial_token: 28356863910078205288614550619314017620
   ```

6. Configure any non-default settings in the node's `cassandra.yaml` to match your existing cluster.
7. Start the new node.
8. After the new node has finished bootstrapping, check that it is marked up using the `nodetool ring` command.
9. Run nodetool repair on each keyspace to ensure the node is fully consistent. For example:

   ```
   $ nodetool repair -h 10.46.123.12 keyspace_name
   ```
10. Remove the dead node.

Backing up and restoring data

Cassandra backs up data by taking a snapshot of all on-disk data files (SSTable files) stored in the data directory. You can take a snapshot of all keyspaces, a single keyspace, or a single table while the system is online.

Using a parallel ssh tool (such as pssh), you can snapshot an entire cluster. This provides an *eventually consistent* backup. Although no one node is guaranteed to be consistent with its replica nodes at the time a snapshot is taken, a restored snapshot resumes consistency using Cassandra's built-in consistency mechanisms.

After a system-wide snapshot is performed, you can enable incremental backups on each node to backup data that has changed since the last snapshot: each time an SSTable is flushed, a hard link is copied into a `/backups` subdirectory of the data directory (provided JNA is enabled).

Note: If JNA is enabled, snapshots are performed by hard links. If not enabled, I/O activity increases as the files are copied from one location to another, which significantly reduces efficiency.

Taking a snapshot

About this task

Snapshots are taken per node using the nodetool snapshot command. To take a global snapshot, run the nodetool snapshot command using a parallel ssh utility, such as pssh.

A snapshot first flushes all in-memory writes to disk, then makes a hard link of the SSTable files for each keyspace. You must have enough free disk space on the node to accommodate making snapshots of your data files. A single snapshot requires little disk space. However, snapshots can cause your disk usage to grow more quickly over time because a snapshot prevents old obsolete data files from being deleted. After the snapshot is complete, you can move the backup files to another location if needed, or you can leave them in place.

Note: Cassandra can only restore data from a snapshot when the table schema exists. It is recommended that you also backup the schema.

Procedure

Run the nodetool snapshot command, specifying the hostname, JMX port, and keyspace. For example:

```
$ nodetool -h localhost -p 7199 snapshot mykeyspace
```

Results

The snapshot is created in *data_directory_location*/*keyspace_name*/*table_name*/ `snapshots`/*snapshot_name* directory. Each snapshot directory contains numerous `.db` files that contain the data at the time of the snapshot.

For example:

Packaged installs:

```
/var/lib/cassandra/data/mykeyspace/mytable/snapshots/1406227071618/
mykeyspace-1-Data.db
```

Tarball installs:

```
install_location/data/data/mykeyspace/mytable/snapshots/1406227071618/
mykeyspace-1-Data.db
```

Deleting snapshot files

About this task

When taking a snapshot, previous snapshot files are not automatically deleted. You should remove old snapshots that are no longer needed.

The nodetool clearsnapshot command removes all existing snapshot files from the snapshot directory of each keyspace. You should make it part of your back-up process to clear old snapshots before taking a new one.

Procedure

To delete all snapshots for a node, run the nodetool `clearsnapshot` command. For example:

```
$ nodetool -h localhost -p 7199 clearsnapshot
```

To delete snapshots on all nodes at once, run the nodetool clearsnapshot command using a parallel ssh utility.

Enabling incremental backups

About this task

When incremental backups are enabled (disabled by default), Cassandra hard-links each flushed SSTable to a backups directory under the keyspace data directory. This allows storing backups offsite without transferring entire snapshots. Also, incremental backups combine with snapshots to provide a dependable, up-to-date backup mechanism.

As with snapshots, Cassandra does not automatically clear incremental backup files. DataStax recommends setting up a process to clear incremental backup hard-links each time a new snapshot is created.

Procedure

Edit the `cassandra.yaml` configuration file on each node in the cluster and change the value of incremental_backups to true.

Restoring from a Snapshot

About this task

Restoring a keyspace from a snapshot requires all snapshot files for the table, and if using incremental backups, any incremental backup files created after the snapshot was taken.

Generally, before restoring a snapshot, you should truncate the table. If the backup occurs before the delete and you restore the backup after the delete without first truncating, you do not get back the original data (row). Until compaction, the tombstone is in a different SSTable than the original row, so restoring the SSTable containing the original row does not remove the tombstone and the data still appears to be deleted.

Cassandra can only restore data from a snapshot when the table schema exists. If you have not backed up the schema, you can do the either of the following:

- Method 1

    1. Restore the snapshot, as described below.
    2. Recreate the schema.
- Method 2

    1. Recreate the schema.
    2. Restore the snapshot, as described below.
    3. Run nodetool refresh.

Procedure

You can restore a snapshot in several ways:

- Use the sstableloader tool.
- Copy the snapshot SSTable directory (see Taking a snapshot) to the `data/`*`keyspace`*`/`*`table_name`* `directory` and then call the JMX method `loadNewSSTables()` in the column family MBean for each column family through JConsole. You can use nodetool refresh instead of the `loadNewSSTables()` call.

    The location of the data directory depends on the type of installation:

    - Packaged installs: `/var/lib/cassandra/data`
    - Tarball installs: *`install_location`*`/data/data`

    The tokens for the cluster you are restoring must match exactly the tokens of the backed-up cluster at the time of the snapshot. Furthermore, the snapshot must be copied to the correct node with matching tokens matching. If the tokens do not match, or the number of nodes do not match, use the `sstableloader` procedure.
- Use the Node Restart Method described below.

Node restart method

About this task

If restoring a single node, you must first shutdown the node. If restoring an entire cluster, you must shut down all nodes, restore the snapshot data, and then start all nodes again.

Note: Restoring from snapshots and incremental backups temporarily causes intensive CPU and I/O activity on the node being restored.

Procedure

1. Shut down the node.
2. Clear all files in the `commitlog` directory:

    - Packaged installs: `/var/lib/cassandra/commitlog`
    - Tarball installs: *`install_location`*`/data/commitlog`

    This prevents the commitlog replay from putting data back, which would defeat the purpose of restoring data to a particular point in time.
3. Delete all `*.db` files in *data_directory_location*/*keyspace_name*/*table_name* directory, but **DO NOT** delete the */snapshots* and */backups* subdirectories.

    where *data_directory_location* is

    - Packaged installs: `/var/lib/cassandra/data`

- Tarball installs: *install_location*/data/data

4. Locate the most recent snapshot folder in this directory:

   data_directory_location/*keyspace_name*/*table_name*/snapshots/*snapshot_name*

5. Copy its contents into this directory:

   data_directory_location/*keyspace_name*/*table_name* directory.

6. If using incremental backups, copy all contents of this directory:

   data_directory_location/*keyspace_name*/*table_name*/backups

7. Paste it into this directory:

   data_directory_location/*keyspace_name*/*table_name*

8. Restart the node.

   Restarting causes a temporary burst of I/O activity and consumes a large amount of CPU resources.

9. Run nodetool repair.

Restoring a snapshot into a new cluster

About this task

Suppose you want to copy a snapshot of SSTable data files from a three node Cassandra cluster with vnodes enabled (256 tokens) and recover it on another newly created three node cluster (256 tokens). The token ranges will not match so you need to specify the tokens for the new cluster that were used in the old cluster.

Note: This procedure assumes you are familiar with restoring a snapshot and configuring and initializing a cluster. If not, see Initializing a cluster.

Procedure

To recover the snapshot on the new cluster:

1. From the old cluster, retrieve the list of tokens associated with each node's IP:

   ```
   $ nodetool ring | grep ip_address_of_node | awk '{print $NF ","}' | xargs
   ```

2. In the `cassandra.yaml` file for each node in the new cluster, add the list of tokens you obtained in the previous step to the initial_token parameter using the same num_tokens setting as in the old cluster.

3. Make any other necessary changes in the `cassandra.yaml` and property files so that the new nodes match the old cluster settings.

4. Clear the system table data from each new node:

   ```
   $ sudo rm -rf /var/lib/cassandra/data/system/*
   ```

   This allows the new nodes to use the initial tokens defined in the `cassandra.yaml` when they restart.

5. Restore the SSTable files snapshotted from the old cluster onto the new cluster using the same directories. Otherwise the new cluster does not have data to read in when you restart the nodes.

6. Start each node using the specified list of token ranges in `cassandra.yaml`:

   ```
   initial_token: -9211270970129494930, -9138351317258731895,
    -8980763462514965928, ...
   ```

   This allows Cassandra to read the SSTable snapshot from the old cluster.

Recovering from a single disk failure using JBOD

About this task

How to recover from a single disk failure in a disk array using JBOD (just a bunch of disks).

Node can restart

1. Stop Cassandra and shut down the node.
2. Replace the failed disk.
3. Start the node and Cassandra.
4. Run nodetool repair on the node.

Node cannot restart

If the node cannot restart, it is possible the system directory is corrupted. If the node cannot restart after completing these steps, see Replacing a dead node.

If using the node uses vnodes:

1. Stop Cassandra and shut down the node.
2. Replace the failed disk.
3. On a healthy node run the following command:

   ```
   $ nodetool ring | grep ip_address_of_node | awk ' {print $NF ","}' | xargs
   ```
4. On the node with the new disk, add the list of tokens from the previous step (separated by commas), under initial_token in the cassandra.yaml file.
5. Clear each `system` directory for every functioning drive:

   Assuming disk1 has failed and the data_file_directories setting in the `cassandra.yaml` for each drive is:

   ```
   -/mnt1/cassandra/data
   -/mnt2/cassandra/data
   -/mnt3/cassandra/data
   ```

   Run the following commands:

   ```
   $ rm -fr /mnt2/cassandra/data/system
   $ rm -fr /mnt3/cassandra/data/system
   ```
6. Start the node and Cassandra.
7. Run nodetool repair.
8. After the node is fully integrated into the cluster, it is recommended to return to normal vnode settings:

   - num_tokens: *number_of_tokens*
   - #initial_token

If the node uses assigned tokens (single-token architecture):

1. Stop Cassandra and shut down the node.
2. Replace the failed disk.
3. Clear each `system` directory for every functioning drive:

   Assuming disk1 has failed and the data_file_directories setting in the `cassandra.yaml` for each drive is:

   ```
   -/mnt1/cassandra/data
   -/mnt2/cassandra/data
   ```

```
-/mnt3/cassandra/data
```

Run the following commands:

```
$ rm -fr /mnt2/cassandra/data/system
$ rm -fr /mnt3/cassandra/data/system
```

4. Start the node and Cassandra.
5. Run nodetool repair on the node.

Cassandra tools

The nodetool utility

The nodetool utility is a command line interface for managing a cluster.

Command format

- Packaged installs: `nodetool -h HOSTNAME [-p JMX_PORT ] COMMAND`
- Tarball installs: `install_location/bin/nodetool -h HOSTNAME [-p JMX_PORT ] COMMAND`
- Remote Method Invocation: `nodetool -h HOSTNAME [-p JMX_PORT -u JMX_USERNAME -pw JMX_PASSWORD ] COMMAND`

  If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.

Most nodetool commands operate on a single node in the cluster if -h is not used to identify one or more other nodes. These commands operate cluster-wide:

- rebuild
- repair
- taketoken

If the node from which you issue the command is the intended target, you do not need the -h option to identify the target; otherwise, identify the target node, or nodes, using the -h option.

Getting nodetool help

nodetool help

  Provides a listing of nodetool commands.

nodetool help *command name*

  Provides help on a specific command. For example

```
$ nodetool help upgradesstables
```

cfhistograms

Provides statistics about a table that could be used to plot a frequency function.

Synopsis

```
nodetool <options> cfhistograms -- <keyspace> <table>
```

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -- Separates an option from an argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is the name of a table.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The `nodetool cfhistograms` command provides statistics about a table, including number of SSTables, read/write latency, partition (row) size, and cell count.

Example

If you are using DataStax Enterprise, performthe Wikipedia demo, and then run this command to get statistics about the solr table in the wiki keyspace.

```
$ nodetool cfhistograms wiki solr
```

The output shows latencies in microseconds (μs), partition (formerly called row) size in bytes, and the number of SSTables and the cell count. The Offset column corresponds to the x-axis in a histogram. It represents buckets of values, which are a series of ranges. Each offset includes the range of values greater than the previous offset and less than or equal to the current offset. The offsets start at 1 and each subsequent offset is calculated by multiplying the previous offset by 1.2, rounding up, and removing duplicates. The offsets can range from 1 to approximately 25 million, with less precision as the offsets get larger.

For example:

- Offset 1 shows that 3579 requests only had to look at one SSTable. The SSTables column corresponds to how many SSTables were involved in a read request.
- Offset 86 shows that there were 663 requests with a write latency between 73 and 86 μs. The range falls into the 73 to 86 bucket.

On some versions of Cassandra, the output looks like this:

```
wiki/solr histograms
Offset          SSTables        Write Latency        Read Latency                Row Size
   Column Count
1               3579                        0                   0                       0
                        0
2                   0                        0                   0                       0
                        0
. . .
35                  0                        0                   0                       0
                        0
42                  0                        0                  27                       0
                        0
50                  0                        0                 187                       0
                        0
60                  0                       10                 460                       0
                        0
72                  0                      200                 689                       0
                        0
86                  0                      663                 552                       0
                        0
103                 0                      796                 367                       0
                        0
124                 0                      297                 736                       0
                        0
```

149	0	265	243	0
	0			
179	0	460	263	0
	0			
. . .				
25109160	0	0	0	0
	0			

On other versions of Cassandra 2.0.x, the output does not label the columns. The offset is on the left, the latency on the right:

```
SSTables per Read
1 sstables: 3579

Write Latency (microseconds)
  50 us: 1
  60 us: 195
  72 us: 1029
  86 us: 876
 103 us: 433
 124 us: 170
 149 us: 208
 179 us: 247
 215 us: 216
 258 us: 137
 310 us: 43
 372 us: 9
 446 us: 4
 535 us: 1
 642 us: 2
 770 us: 2
 924 us: 1
1109 us: 1
1331 us: 2
1597 us: 0
1916 us: 0
2299 us: 1
2759 us: 1

Read Latency (microseconds)
  50 us: 4
  60 us: 69
  72 us: 384
  86 us: 802
 103 us: 936
 124 us: 523
 149 us: 452
 179 us: 271
 215 us: 106
 258 us: 13
 310 us: 4
 372 us: 6
 446 us: 1
 535 us: 3
 642 us: 3
 770 us: 0
 924 us: 0
1109 us: 0
1331 us: 1
1597 us: 0
1916 us: 0
2299 us: 0
2759 us: 0
3311 us: 0
```

```
 3973 us: 0
 4768 us: 0
 5722 us: 0
 6866 us: 0
 8239 us: 0
 9887 us: 0
11864 us: 0
14237 us: 1

Partition Size (bytes)
No Data

Cell Count per Partition
No Data
```

OpsCenter displays the same information in a better format for understanding the statistics.

cfstats

Provides statistics about tables.

Synopsis

`nodetool <options> cfstats -i -- (<keyspace>.<table> ... )`

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option from an argument that could be mistaken for a option.
- keyspace.table is one or more keyspace and table names in dot notation.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The `nodetool cfstats` command provides statistics about one or more tables. You use dot notation to specify one or more keyspace and table names. If you do not specify a keyspace and table, Cassandra provides statistics about all tables.

This table describes the `nodetool cfstats` output.

Table 10: nodetool cfstats output

Name of statistic	Example value	Brief description	Related information
Keyspace	libdata	Name of the keyspace	Keyspace and table
Read count	11207	Number of requests to	

Name of statistic	Example value	Brief description	Related information
		read tables in the libdata keyspace since startup	
Read latency	0.0479311144820020164 ms	Latency reading the tables in the libdata keyspace	OpsCenter alert metrics
Write count	17598	Number of requests to update tables in the libdata keyspace since startup	Same as above
Write latency	0.0535029548812965506 ms	Latency writing tables in the libdata keyspace	Same as above
Pending tasks	0	Tasks in the queue for reads, writes, and cluster operations of tables in the keyspace	OpsCenter pending task metrics
Table	libout	Name of the Cassandra table	
SSTable count	3	Number of SSTables containing data from the table	How to use the SSTable counts metric and OpsCenter alert metrics
Space used (live), bytes:	9592399	Space that is measured depends on operating system	Advanced system alert metrics
Space used (total), bytes:	9592399	Same as above	Same as above
SSTable compression ratio	0.3675136388821150946	Fraction of data-representation size resulting from compression	Types of compression option)

Name of statistic	Example value	Brief description	Related information
Memtable cell count	0	Number of cells (storage engine rows x columns) of data in the memtable	Cassandra memtable structure in memory
Memtable data size, bytes:	0	Size of the memtable data	Same as above
Memtable switch count	3	Number of times a full memtable was swapped for an empty one that increases each time the memtable for a table is flushed to disk	How memtables are measured article
Local read count	11207	Number of local read requests for the libout table since startup	OpsCenter alert documentation
Local read latency	0.048 ms	Round trip time in milliseconds to complete a request to read the libout table	Factors that affect read latency
Local write count	17598	Number of local requests to update the libout the table since startup	OpsCenter alert documentation
Local write latency	0.054 ms	Round trip time in milliseconds to complete an update	Factors that affect write latency

Name of statistic	Example value	Brief description	Related information
		to the libout table	
Pending tasks	0	Number of read, write, and cluster operations that are pending	OpsCenter pending task metrics documentation
Bloom filter false positives	0	Number of false positives, which occur when the bloom filter said the row existed, but it actually did not exist in absolute numbers	Tuning bloom filters
Bloom filter false ratio	0.00000	Fraction of all bloom filter checks resulting in a false positive	Same as above
Bloom filter space used, bytes	11688	Bytes of bloom filter data	Same as above
Compacted partition minimum bytes	1110	Lower size limit for the partition being compacted in memory	Used to calculate what the approximate row cache size should be. Multiply the reported row cache size, which is the number of rows in the cache, by the compacted row mean size for every table and sum them.
Compacted partition maximum bytes	126934	Upper size limit for compacted table rows, configurable in the cassandra.yaml in_memory_compaction _limit_in_mb	Same as above
Compacted partition mean bytes	2730	The average size of compacted table rows	Same as above
Average live cells per	0.0		

Name of statistic	Example value	Brief description	Related information
slice (last five minutes)			
Average tombstones per slice (last five minutes)	0.0		

Example

This example shows an excerpt of the output of the command after flushing a table of library data to disk.

```
Keyspace: libdata
 Read Count: 11207
 Read Latency: 0.047931114482020164 ms.
 Write Count: 17598
 Write Latency: 0.053502954881236506 ms.
 Pending Tasks: 0
  Table: libout
  SSTable count: 3
  Space used (live), bytes: 9088955
  Space used (total), bytes: 9088955
  Space used by snapshots (total), bytes: 0
  SSTable Compression Ratio: 0.36751363892150946
  Memtable cell count: 0
  Memtable data size, bytes: 0
  Memtable switch count: 3
  Local read count: 11207
  Local read latency: 0.048 ms
  Local write count: 17598
  Local write latency: 0.054 ms
  Pending tasks: 0
  Bloom filter false positives: 0
  Bloom filter false ratio: 0.00000
  Bloom filter space used, bytes: 11688
  Compacted partition minimum bytes: 1110
  Compacted partition maximum bytes: 126934
  Compacted partition mean bytes: 2730
  Average live cells per slice (last five minutes): 0.0
  Average tombstones per slice (last five minutes): 0.0
```

cleanup

Cleans up keyspaces and partition keys no longer belonging to a node.

Synopsis

```
nodetool <options> cleanup -- <keyspace > ( <table> ... )
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option from an argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Use this command to remove unwanted data after adding a new node to the cluster. Cassandra does not automatically remove data from nodes that lose part of their partition range to a newly added node. Run `nodetool cleanup` on the source node and on neighboring nodes that shared the same subrange after the new node is up and running. Failure to run this command after adding a node causes Cassandra to include the old data to rebalance the load on that node. Running the `nodetool cleanup` command causes a temporary increase in disk space usage proportional to the size of your largest SSTable. Disk I/O occurs when running this command.

Running this command affects nodes that use a counter column in a table. Cassandra assigns a new counter ID to the node.

Optionally, this command takes a list of table names. If you do not specify a keyspace, this command cleans all keyspaces no longer belonging to a node.

clearsnapshot

Removes one or more snapshots.

Synopsis

```
nodetool <options> clearsnapshot -t <snapshot > -- ( <keyspace> ... )
```

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -t means the following file contains the snapshot.
- -- Separates an option from an argument that could be mistaken for a option.
- keyspace is one or more keyspace names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Deletes snapshots for one or more keyspaces. To remove all snapshots, omit the snapshot name.

Caution: Removing all snapshots or all specific keyspace snapshot also removes OpsCenter backups.

compact

Forces a major compaction on one or more tables.

Synopsis

```
nodetool <options> compact <keyspace> ( <table> ... )
```

- options are:

    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>

- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command starts the compaction process on tables that use the SizeTieredCompactionStrategy or DateTieredCompactionStrategy. You can specify a keyspace for compaction. If you do not specify a keyspace, the nodetool command uses the current keyspace. You can specify one or more tables for compaction. If you do not specify a table(s), compaction of all tables in the keyspace occurs. This is called a major compaction. If you do specify a table(s), compaction of the specified table(s) occurs. This is called a minor compaction. A major compaction consolidates all existing SSTables into a single SSTable. During compaction, there is a temporary spike in disk space usage and disk I/O because the old and new SSTables co-exist. A major compaction can cause considerable disk I/O.

compactionhistory

Provides the history of compaction operations.

Synopsis

```
nodetool <options> compactionhistory
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR

- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

compactionstats

Provide statistics about a compaction.

Synopsis

```
nodetool <options> compactionstats
```

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- separates an option and argument that could be mistaken for a option.
- data center is the name of an arbitrarily chosen data center from which to select sources for streaming.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The total column shows the total number of uncompressed bytes of SSTables being compacted. The system log lists the names of the SSTables compacted.

Example

```
$ bin/nodetool compactionstats
pending tasks: 5
        compaction type        keyspace        table        completed
    total      unit  progress
            Compaction      Keyspace1      Standard1      282310680
  302170540    bytes    93.43%
            Compaction      Keyspace1      Standard1      58457931
  307520780    bytes    19.01%
Active compaction remaining time :    0h00m16s
```

decommission

Deactivates a node by streaming its data to another node.

Synopsis

```
nodetool <options> decommission
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >

- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Causes a live node to decommission itself, streaming its data to the next node on the ring. Use netstats to monitor the progress, as described on http://wiki.apache.org/cassandra/NodeProbe#Decommission and http://wiki.apache.org/cassandra/Operations#Removing_nodes_entirely.

describering

Provides the partition ranges of a keyspace.

Synopsis

`nodetool <options> describering -- <keyspace>`

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -- Separates an option from an argument that could be mistaken for a option.
- keyspace is a keyspace name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

disableautocompaction

Disables autocompaction for a keyspace and one or more tables.

Synopsis

`nodetool <options> disableautocompaction -- <keyspace> ( <table> ... )`

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.

- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The keyspace can be followed by one or more tables.

disablebackup

Disables incremental backup.

Synopsis

```
nodetool <options> disablebackup
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

disablebinary

Disables the native transport.

Synopsis

```
nodetool <options> disablebinary
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR

- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Disables the binary protocol, also known as the native transport.

disablegossip

Disables the gossip protocol.

Synopsis

```
nodetool <options> disablegossip
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command effectively marks the node as being down.

disablehandoff

Disables storing of future hints on the current node.

Synopsis

```
nodetool <options> disablehandoff
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

disablethrift

Disables the Thrift server.

Synopsis

```
nodetool <options> disablethrift
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

drain

Drains the node.

Synopsis

```
nodetool <options> drain
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Flushes all memtables from the node to SSTables on disk. Cassandra stops listening for connections from the client and other nodes. You need to restart Cassandra after running `nodetool drain`. You typically use this command before upgrading a node to a new version of Cassandra. To simply flush memtables to disk, use `nodetool flush`.

enableautocompaction

Enables autocompaction for a keyspace and one or more tables.

Synopsis

```
nodetool <options> enableautocompaction -- <keyspace> ( <table> ... )
```

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is the name of one or more keyspaces, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The keyspace can be followed by one or more tables.

enablebackup

Enables incremental backup.

Synopsis

```
nodetool <options> enablebackup
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

enablebinary

Re-enables native transport.

Synopsis

```
nodetool <options> enablebinary
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Re-enables the binary protocol, also known as native transport.

enablegossip

Re-enables gossip.

Synopsis

```
nodetool <options> enablegossip
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

enablehandoff

Re-enables the storing of future hints on the current node.

Synopsis

```
nodetool <options> enablehandoff
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

enablethrift

Re-enables the Thrift server.

Synopsis

```
nodetool <options> enablethrift
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

flush

Flushes one or more tables from the memtable.

Synopsis

```
 nodetool <options> flush -- <keyspace> ( <table> ... )
```

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is the name of one or more tables, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

You can specify a keyspace followed by one or more tables that you want to flush from the memtable to SSTables on disk.

getcompactionthreshold

Provides the minimum and maximum compaction thresholds in megabytes for a table.

Synopsis

```
nodetool <options> getcompactionthreshold -- <keyspace> <table>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is the name of a table.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

getendpoints

Provides the end points that own the partition key. <note>The partitioner returns a token for the key. Cassandra will return an endpoint whether or not data exists on the identified node for that token. </note>

Synopsis

```
nodetool <options> getendpoints -- <keyspace> <table> key
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is a table name.
- key is the partition key of the end points you want to get.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable

- Orange (and) means not literal, indicates scope

getsstables

Provides the SSTables that own the partition key.

Synopsis

`nodetool <options> getsstables -- <keyspace> <table> key`

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is a table name.
- key is the partition key of the SSTables.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

getstreamthroughput

Provides the megabytes per second throughput limit for streaming in the system.

Synopsis

`nodetool <options> getstreamthroughput`

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

gossipinfo

Provides the gossip information for the cluster.

Synopsis

```
nodetool <options> gossipinfo
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

info

Provides node information, such as load and uptime.

Synopsis

```
nodetool <options> info ( -T | --tokens )
```

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -T or --tokens means provide all token information.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Provides node information including the token and on disk storage (load) information, times started (generation), uptime in seconds, and heap memory usage.

invalidatekeycache

Resets the global key cache parameter to the default, which saves all keys.

Synopsis

```
nodetool <options> invalidatekeycache
```

options are:

- (-h | --host) <host name> | <ip address>

- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

By default the key_cache_keys_to_save is disabled in the cassandra.yaml. This command resets the parameter to the default.

invalidaterowcache

Resets the global key cache parameter, row_cache_keys_to_save, to the default (not set), which saves all keys.

Synopsis

```
nodetool <options> invalidaterowcache
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

join

Causes the node to join the ring.

Synopsis

```
nodetool <options> join
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Causes the node to join the ring, assuming the node was initially *not* started in the ring using the -Djoin_ring=false cassandra utility option. The joining node should be properly configured with the desired options for seed list, initial token, and auto-bootstrapping.

move

Moves the node on the token ring to a new token.

Synopsis

```
nodetool <options> move -- <new token>
```

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- new token is a number in the range 0 to 2^{127} -1 for negative tokens.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Escape negative tokens using \\ . For example: move \\-123 . This command essentially combines decommission and bootstrapoperations.

netstats

Provides network information about the host.

Synopsis

```
nodetool <options> netstats
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >

- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The default host is the connected host if the user does not include a host name or IP address in the command. The output includes the following information:

- JVM settings
- Mode
- Read repair statistics
- Attempted

  The number of successfully completed read repair operations
- Mismatch (blocking)

  The number of read repair operations since server restart that blocked a query.
- Mismatch (background)

  The number of read repair operations since server restart performed in the background.
- Pool name

  Information about client read and write requests by thread pool.

Example

Get the network information for a node 10.171.147.128:

```
nodetool –h 10.171.147.128 netstats
```

An example of output is:

```
Mode: NORMAL
Not sending any streams.
Not receiving any streams.
Read Repair Statistics:
Attempted: 1
Mismatch (Blocking): 0
Mismatch (Background): 0
Pool Name                       Active    Pending      Completed
Commands                          n/a          0              0
Responses                         n/a          0              0
```

pausehandoff

Pauses the hints delivery process.

Synopsis

```
nodetool <options> pausehandoff
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>

- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

proxyhistograms

Provides a histogram of network statistics.

Synopsis

```
nodetool <options> proxyhistograms
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The output of this command shows the full request latency recorded by the coordinator. The output includes the percentile rank of read and write latency values for inter-node communication. Range latency is the latency of range scans, such as getting all partitions, executing an IN statement, or performing any selection that requires ALLOW FILTERING or index lookups. Typically, you use the command to see if requests encounter a slow node.

Examples

This example shows the output from nodetool proxyhistograms after running 4,500 insert statements and 45,000 select statements on a three ccm node-cluster on a local computer. The output shows an offset on the left and latencies in microseconds (μs) on the right. The offset corresponds to the x-axis in a histogram. It represents buckets of values, which are a series of ranges. Each offset includes the range of values greater than the previous offset and less than or equal to the current offset. Each offset is calculated by multiplying the previous offset by 1.2, rounding up, and removing duplicates.

```
$ nodetool proxyhistograms
proxy histograms

Read Latency (microseconds)
61214 us: 1

Write Latency (microseconds)
```

```
    103 us: 22
    124 us: 142
    149 us: 297
    179 us: 1190
    215 us: 1823
    258 us: 2091
    310 us: 1291
    372 us: 753
    446 us: 297
    535 us: 72
    642 us: 26
    770 us: 15
    924 us: 4
   1109 us: 0
   1331 us: 0
   1597 us: 0
   1916 us: 0
   2299 us: 0
   2759 us: 0
   3311 us: 1
   3973 us: 0
   4768 us: 0
   5722 us: 0
   6866 us: 0
   8239 us: 0
   9887 us: 0
  11864 us: 0
  14237 us: 0
  17084 us: 0
  20501 us: 0
  24601 us: 0
  29521 us: 0
  35425 us: 0
  42510 us: 0
  51012 us: 1

Range Latency (microseconds)
    310 us: 1
    372 us: 139
    446 us: 1824
    535 us: 8933
    642 us: 6123
    770 us: 3672
    924 us: 2178
   1109 us: 1152
   1331 us: 673
   1597 us: 112
   1916 us: 15
   2299 us: 2
   2759 us: 1
   3311 us: 1
   3973 us: 0
   4768 us: 0
   5722 us: 0
   6866 us: 0
   8239 us: 1
   9887 us: 0
  11864 us: 0
  14237 us: 2
  17084 us: 0
  20501 us: 0
  24601 us: 0
  29521 us: 1
  35425 us: 0
```

```
42510 us: 1
51012 us: 1
```

rangekeysample

Provides the sampled keys held across all keyspaces.

Synopsis

`nodetool <`*`options`*`> rangekeysample`

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

rebuild

Rebuilds data by streaming from other nodes.

Synopsis

`nodetool <`*`options`*`> rebuild -- <data center>`

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- data center is the name of an arbitrarily chosen data center from which to select sources for streaming.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command operates on multiple nodes in a cluster. Like bootstrap, rebuild only streams data from a single source replica per range. Use this command to bring up a new data center in an existing cluster. See Adding a data center to a cluster .

For example, when adding a new data center, you would run the following on all nodes in the new data center:

```
nodetool rebuild -- name_of_existing_data_center
```

Attention: If you don't specify the existing data center in the command line, the new nodes will appear to rebuild successfully, but will not contain any data.

You can run rebuild on one or more nodes at the same time. The choices depends on whether your cluster can handle the extra IO and network pressure of running on multiple nodes. Running on one node at a time has the least impact on the existing cluster.

rebuild_index

Performs a full rebuild of the index for a table.

Synopsis

```
nodetool <options> rebuild_index -- <keyspace> <table> ( <index> ... )
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is a table name.
- index is a list of index names separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Fully rebuilds one or more indexes for a table.

refresh

Loads newly placed SSTables onto the system without a restart.

Synopsis

```
nodetool <options> refresh -- <keyspace> <table>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>

- -- Separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.
- table is a table name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

removenode

Provides the status of current node removal, forces completion of pending removal, or removes the identified node.

Synopsis

```
nodetool <options> removenode --<status> | <force> | <ID>
```

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- status provides status information.
- force forces completion of the pending removal.
- ID is the host ID, in UUID format.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command removes a node, shows the status of a removal operation, or forces the completion of a pending removal. When the node is down and `nodetool decommission` cannot be used, use `nodetool removenode`. Run this command only on nodes that are down. If the cluster does not use vnodes, before running the `nodetool removenode` command, adjust the tokens.

Examples

Determine the UUID of the node to remove by running `nodetool status`. Use the UUID of the node that is down to remove the node.

```
$ nodetool status

Datacenter: DC1
===============
Status=Up/Down
```

```
|/ State=Normal/Leaving/Joining/Moving
--  Address          Load        Tokens  Owns (effective)  Host ID
                 Rack
UN  192.168.2.101  112.82 KB   256     31.7%             420129fc-0d84-42b0-
be41-ef7dd3a8ad06   RAC1
DN  192.168.2.103  91.11 KB    256     33.9%             d0844a21-3698-4883-
ab66-9e2fd5150edd   RAC1
UN  192.168.2.102  124.42 KB   256     32.6%             8d5ed9f4-7764-4dbd-
bad8-43fddce94b7c   RAC1

$ nodetool removenode d0844a21-3698-4883-ab66-9e2fd5150edd
```

View the status of the operation to remove the node:

```
$ nodetool removenode status

RemovalStatus: No token removals in process.
```

Confirm that the node has been removed.

```
$ nodetool status

Datacenter: DC1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address          Load        Tokens  Owns (effective)  Host ID
                 Rack
UN  192.168.2.101  112.82 KB   256     37.7%             420129fc-0d84-42b0-
be41-ef7dd3a8ad06   RAC1
UN  192.168.2.102  124.42 KB   256     38.3%             8d5ed9f4-7764-4dbd-
bad8-43fddce94b7c   RAC1
```

repair

Repairs one or more tables.

Synopsis

```
nodetool <options> repair
  ( -dc <dc_name> | --in-dc <dc_name> )
  ( -et <end_token> | --end-token <end_token> )
  ( -local | --in-local-dc )
  ( -par | --parallel )
  ( -pr | --partitioner-range )
  ( -st <start_token> --start-token <start_token> )
-- <keyspace> ( <table> ... )
```

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -dc, or --in-dc, followed by dc_name, or means restrict repair to nodes in the named data center, which must be the local data center.
- -et, or --end-token, followed by the UUID of a token means stop repairing a range of nodes after repairing this token.
- -local, or --in-local-dc, means repair nodes in the same data center only.
- -par, or --parallel, means carry out a parallel repair.
- pr, or --partitioner-range, means repair only the first range returned by the partitioner.

- -st, or --start-token, followed by the UUID of a token means start repairing a range of nodes at this token.
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the keyspace name.
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command operates on multiple nodes in a cluster, and includes an option to restrict repair to a set of nodes. This command begins an anti-entropy node repair operation. If the -pr option is specified, only the first range returned by the partitioner for a node is repaired. This allows you to repair each node in the cluster in succession without duplicating work. If the -pr option is not specified, Cassandra repairs all replica ranges that fall within the responsibility of the node.

By default, the repair command takes a snapshot of each replica immediately and then sequentially repairs each replica from the snapshots. For example, if you have RF=3 and A, B and C represents three replicas, this command takes a snapshot of each replica immediately and then sequentially repairs each replica from the snapshots (A<->B, A<->C, B<->C) instead of repairing A, B, and C all at once. This allows the dynamic snitch to maintain performance for your application via the other replicas, because at least one replica in the snapshot is not undergoing repair.

Unlike sequential repair (described above), parallel repair constructs the Merkle tables for all nodes at the same time. Therfore, no snapshots are required (or generated). Use a parallel repair to complete the repair quickly or when you have operational downtime that allows the resources to be completely consumed during the repair.

To restrict the repair to the local data center, use the -dc option followed by the name of the data center. Issue the command from a node in the data center you want to repair. Issuing the command from a data center other than the named one returns an error. Do not use -pr with this option to repair only a local data center.

```
$ nodetool repair -dc DC1
[2014-07-24 21:59:55,326] Nothing to repair for keyspace 'system'
[2014-07-24 21:59:55,617] Starting repair command #2, repairing 490 ranges
  for keyspace system_traces (seq=true, full=true)
[2014-07-24 22:23:14,299] Repair session 323b9490-137e-11e4-88e3-c972e09793ca
  for range (820981369067266915,822627736366088177] finished
[2014-07-24 22:23:14,320] Repair session 38496a61-137e-11e4-88e3-c972e09793ca
  for range (2506042417712465541,2515941262699962473] finished
. . .
```

An inspection of the system.log shows repair taking place only on IP addresses in DC1.

```
. . .
INFO  [AntiEntropyStage:1] 2014-07-24 22:23:10,708 RepairSession.java:171
  - [repair #16499ef0-1381-11e4-88e3-c972e09793ca] Received merkle tree
  for sessions from /192.168.2.101
INFO  [RepairJobTask:1] 2014-07-24 22:23:10,740 RepairJob.java:145
  - [repair #16499ef0-1381-11e4-88e3-c972e09793ca] requesting merkle trees
  for events (to [/192.168.2.103, /192.168.2.101])
. . .
```

For more information about the repair operation, see "Repairing nodes."

resetlocalschema

Reset the node's local schema and resynchronizes.

Synopsis

```
nodetool <options> resetlocalschema
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password>
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

resumehandoff

Resume hints delivery process.

Synopsis

```
nodetool <options> resumehandoff
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password>
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

ring

Provides node status and information about the ring.

Synopsis

```
nodetool <options> ring ( -r |  --resolve-ip ) -- <keyspace>
```

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>

- (-pw | --password) <password >
- (-u | --username) <user name>
- -r, or --resolve-ip, means to provide node names instead of IP addresses.
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Displays node status and information about the ring as determined by the node being queried. This information can give you an idea of the load balance and if any nodes are down. If your cluster is not properly configured, different nodes may show a different ring. Check that the node appears the same way in the ring.If you use virtual nodes (vnodes), use `nodetool status` for succinct output.

- Address

  The node's URL.
- DC (data center)

  The data center containing the node.
- Rack

  The rack or, in the case of Amazon EC2, the availability zone of the node.
- Status - Up or Down

  Indicates whether the node is functioning or not.
- State - N (normal), L (leaving), J (joining), M (moving)

  The state of the node in relation to the cluster.
- Load - updates every 90 seconds

  The amount of file system data under the cassandra data directory after excluding all content in the snapshots subdirectories. Because all SSTable data files are included, any data that is not cleaned up, such as TTL-expired cell or tombstoned data) is counted.
- Token

  The end of the token range up to and including the value listed. For an explanation of token ranges, see Data Distribution in the Ring .
- Owns

  The percentage of the data owned by the node per data center times the replication factor. For example, a node can own 33% of the ring, but show100% if the replication factor is 3.
- Host ID

  The network ID of the node.

scrub

Rebuild SSTables for one or more Cassandra tables.

Synopsis

```
nodetool <options> scrub <keyspace> -- ( -ns | --no-snapshot ) ( -s | --skip-
corrupted ) ( <table> ... )
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
  - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- -ns, or --no-snapshot, triggers a snapshot of the scrubbed table first assuming snapshots are not disabled (the default).
- - s, or --skip-corrupted skips corrupted partitions even when scrubbing counter tables. (default false)
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) means not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Rebuilds SSTables on a node for the named tables and snapshots data files before rebuilding as a safety measure. If possible use upgradesstables. While scrub rebuilds SSTables, it also discards data that it deems broken and creates a snapshot, which you have to remove manually. If scrub can't validate the column value against the column definition's data type, it logs the partition key and skips to the next partition. If the -ns option is specified, snapshot creation is disabled.

Skipping corrupt rows in tables having counter columns results in undercounts. By default the scrub operation stops if you attempt to skip such a row. To force the scrub to skip the row and continue scrubbing, re-run nodetool scrub using the --skip-corrupted option.

setcachecapacity

Set global key and row cache capacities in megabytes.

Synopsis

```
nodetool <options> setcachecapacity -- <key-cache-capacity> <row-cache-
capacity>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- key-cache-capacity is the maximum size in MB of the key cache in memory.
- row-cache-capacity corresponds to the maximum size in MB of the row cache in memory.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The key-cache-capacity argument corresponds to the key_cache_size_in_mb parameter in the cassandra.yaml. Each key cache hit saves one seek and each row cache hit saves a minimum of two seeks. Devoting some memory to the key cache is usually a good tradeoff considering the positive effect on the response time. The default value is empty, which means a minimum of five percent of the heap in MB or 100 MB.

The row-cache-capacity argument corresponds to the row_cache_size_in_mb parameter in the cassandra.yaml. By default, row caching is zero (disabled).

setcachekeystosave

Sets the number of keys saved by each cache for faster post-restart warmup.

Synopsis

```
nodetool <options> setcachekeystosave -- <key-cache-keys-to-save> <row-cache-keys-to-save>
```

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- key-cache-keys-to-save is the number of keys from the key cache to save to the saved caches directory.
- row-cache-keys-to-save is the number of keys from the row cache to save to the saved caches directory.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command saves the specified number of key and row caches to the saved caches directory, which you specify in the cassandra.yaml. The key-cache-keys-to-save argument corresponds to the key_cache_keys_to_save in the cassandra.yaml, which is disabled by default, meaning all keys will be saved. The row-cache-keys-to-save argument corresponds to the row_cache_keys_to_save in the cassandra.yaml, which is disabled by default.

setcompactionthreshold

Sets minimum and maximum compaction thresholds for a table.

Synopsis

```
nodetool <options> setcompactionthreshold -- <keyspace> <table> <minthreshold>
 <maxthreshold>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is the name of a keyspace.
- table is a table name.
- minthreshold sets the minimum number of SSTables to trigger a minor compaction when using SizeTieredCompactionStrategy or DateTieredCompactionStrategy.
- maxthreshold sets the maximum number of SSTables to allow in a minor compaction when using SizeTieredCompactionStrategy or DateTieredCompactionStrategy.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This parameter controls how many SSTables of a similar size must be present before a minor compaction is scheduled. The max_threshold sets an upper bound on the number of SSTables that may be compacted in a single minor compaction, as described in http://wiki.apache.org/cassandra/MemtableSSTable .

When using LeveledCompactionStrategy, maxthreshold sets the MAX_COMPACTING_L0, which limits the number of L0 SSTables that are compacted concurrently to avoid wasting memory or running out of memory when compacting highly overlapping SSTables.

setcompactionthroughput

Sets the throughput capacity for compaction in the system, or disables throttling.

Synopsis

```
nodetool <options> setcompactionthroughput -- <value_in_mb>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- value_in_mb is the throughput capacity in MB per second for compaction.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Set value_in_mb to 0 to disable throttling.

sethintedhandoffthrottlekb

Sets hinted handoff throttle in kb/sec per delivery thread. (Cassandra 2.0.11 and later)

Synopsis

```
nodetool <options> sethintedhandoffthrottlekb <value_in_kb/sec>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
  - (-pwf <passwordFilePath | --password-file <passwordFilePath>)
- value_in_kb/sec is the throttle time.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

When a node detects that a node for which it is holding hints has recovered, it begins sending the hints to that node. This setting specifies the maximum sleep interval per delivery thread in kilobytes per second after delivering each hint. The interval shrinks proportionally to the number of nodes in the cluster. For example, if there are two nodes in the cluster, each delivery thread uses the maximum interval; if there are three nodes, each node throttles to half of the maximum interval, because the two nodes are expected to deliver hints simultaneously.

Example

```
nodetool sethintedhandoffthrottlekb 2048
```

setstreamthroughput

Sets the throughput capacity in MB for streaming in the system, or disable throttling.

Synopsis

```
nodetool <options> setstreamthroughput -- <value_in_mb>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- value_in_mb is the throughput capacity in MB per second for streaming.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Set value_in_MB to 0 to disable throttling.

settraceprobability

Sets the probability for tracing a request.

Synopsis

```
nodetool <options> settraceprobability -- <value>
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- value is a probability between 0 and 1.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Probabilistic tracing is useful to determine the cause of intermittent query performance problems by identifying which queries are responsible. This option traces some or all statements sent to a cluster. Tracing a request usually requires at least 10 rows to be inserted.

A probability of 1.0 will trace everything whereas lesser amounts (for example, 0.10) only sample a certain percentage of statements. Care should be taken on large and active systems, as system-wide tracing will have a performance impact. Unless you are under very light load, tracing all requests (probability 1.0) will probably overwhelm your system. Start with a small fraction, for example, 0.001 and increase only if necessary. The trace information is stored in a system_traces keyspace that holds two tables –

sessions and events, which can be easily queried to answer questions, such as what the most time-consuming query has been since a trace was started. Query the parameters map and thread column in the system_traces.sessions and events tables for probabilistic tracing information.

snapshot

Take a snapshot of one or more keyspaces, or of a table, to backup data.

Synopsis

```
nodetool <options> snapshot (
   ( -cf <table> | --column-family <table> )
   ( -t <tag> | --tag <tag> )
   -- ( <keyspace> ) | ( <keyspace> ... )
)
```

- options are:
    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -cf, or --column-family, followed by the name of the table to be backed up.
- -t or --tag, followed by the snapshot name.
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is one keyspace name that is required when using the -cf option, or one or more optional keyspace names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Use this command to back up data using a snapshot. Depending on how you use the command, the following data is included:

- All keyspaces on a node.

    Omit the optional keyspace and table parameters, as shown in the first example.
- One or more keyspaces and all tables in the named keyspaces.

    Include one or more names of the keyspaces, as shown in the second and third examples.
- A single table.

    Include the name of a single keyspace and a table using the -cf option, as shown in the last example.

Cassandra flushes the node before taking a snapshot, takes the snapshot, and stores the data in the snapshots directory of each keyspace in the data directory. If you do not specify the name of a snapshot directory using the -t option, Cassandra names the directory using the timestamp of the snapshot, for example 1391460334889. Follow the procedure for taking a snapshot before upgrading Cassandra. When upgrading, backup all keyspaces. For more information about snapshots, see Apache documentation .

Example: All keyspaces

Take a snapshot of all keyspaces on the node. On Linux, in the Cassandra `bin` directory, for example:

```
$ ./nodetool snapshot
```

The following message appears:

```
Requested creating snapshot(s) for [all keyspaces] with snapshot name
 [1391464041163]
Snapshot directory: 1391464041163
```

Because you did not specify a snapshot name, Cassandra names snapshot directories using the timestamp of the snapshot. If the keyspace contains no data, empty directories are not created.

Example: Single keyspace snapshot

Assuming you created the keyspace and tables in the music service example, take a snapshot of the music keyspace and name the snapshot 2014.06.24. On Linux, in the Cassandra `bin` directory, for example:

```
$ ./nodetool snapshot -t 2014.06.24 music
```

The following message appears:

```
Requested creating snapshot(s) for [music] with snapshot name [2014.06.24]
Snapshot directory: 2014.06.24
```

Assuming the music keyspace contains two tables, songs and playlists, taking a snapshot of the keyspace creates multiple snapshot directories named 2014.06.24. A number of .db files containing the data are located in these directories. For example:

```
$ cd /var/lib/cassandra/data/music/playlists-bf8118508cfd11e3972273ded3cb6170/
snapshots/2014.06.24
$ ls
music-playlists-ka-1-CompressionInfo.db  music-playlists-ka-1-Index.db
 music-playlists-ka-1-TOC.txt
music-playlists-ka-1-Data.db             music-playlists-ka-1-Statistics.db
music-playlists-ka-1-Filter.db           music-playlists-ka-1-Summary.db
$ cd /var/lib/cassandra/data/music/songs-
b8e385a08cfd11e3972273ded3cb6170/2014.06.24/snapshots/2014.06.24
music-songs-ka-1-CompressionInfo.db music-songs-ka-1-Index.db  music-songs-
ka-1-TOC.txt
music-songs-ka-1-Data.db   music-songs-ka-1-Statistics.db
music-songs-ka-1-Filter.db  music-songs-ka-1-Summary.db
```

Example: Multiple keyspaces snapshot

Assuming you created a keyspace named mykeyspace in addition to the music keyspace, take a snapshot of both keyspaces. On Linux, in the Cassandra `bin` directory, for example:

```
$ ./nodetool snapshot mykeyspace music
```

The following message appears:

```
Requested creating snapshot(s) for [mykeyspace, music] with snapshot name
 [1391460334889]
Snapshot directory: 1391460334889
```

Example: Single table snapshot

Take a snapshot of only the playlists table in the music keyspace. On Linux, in the Cassandra `bin` directory, for example:

```
$ ./nodetool snapshot -cf playlists music
Requested creating snapshot(s) for [music] with snapshot name [1391461910600]
Snapshot directory: 1391461910600
```

Cassandra creates the snapshot directory named 1391461910600 that contains the backup data of playlists table in `/var/lib/cassandra/data/music/playlists-bf8118508cfd11e3972273ded3cb6170/snapshots`, for example.

status

Provide information about the cluster, such as the state, load, and IDs.

Synopsis

`nodetool <options> status (-r | --resolve-ip ) -- <keyspace>`

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -r, or --resolve-ip, means to provide node names instead of IP addresses.
- -- Separates an option and argument that could be mistaken for a option.
- keyspace is a keyspace name.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

The status command provides the following information:

- Status - U (up) or D (down)

  Indicates whether the node is functioning or not.
- State - N (normal), L (leaving), J (joining), M (moving)

  The state of the node in relation to the cluster.
- Address

  The node's URL.
- Load - updates every 90 seconds

  The amount of file system data under the cassandra data directory after excluding all content in the snapshots subdirectories. Because all SSTable data files are included, any data that is not cleaned up, such as TTL-expired cell or tombstoned data) is counted.
- Tokens

  The number of tokens set for the node.
- Owns

  The percentage of the data owned by the node per data center times the replication factor. For example, a node can own 33% of the ring, but show100% if the replication factor is 3.

  Attention: If your cluster uses multiple data centers with different keyspaces that use different replication strategies or replication factors, you must specify a keyspace to get meaningful ownship information.
- Host ID

The network ID of the node.

- Rack

  The rack or, in the case of Amazon EC2, the availability zone of the node.

statusbinary

Provide the status of native transport.

Synopsis

`nodetool <`*`options`*`> statusbinary`

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Provides the status of the binary protocol, also known as the native transport.

statusthrift

Provide the status of the Thrift server.

Synopsis

`nodetool <`*`options`*`> statusthrift`

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

stop

Stops the compaction process.

Synopsis

```
nodetool <options> stop  -- <compaction_type>
```

- options are:
  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>
  - (-pw | --password) <password >
  - (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- A compaction type: COMPACTION, VALIDATION, CLEANUP, SCRUB, INDEX_BUILD

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Stops an operation from continuing to run. This command is typically used to stop a compaction that has a negative impact on the performance of a node. After the compaction stops, Cassandra continues with the remaining operations in the queue. Eventually, Cassandra restarts the compaction.

stopdaemon

Stops the cassandra daemon.

Synopsis

```
nodetool <options> stopdaemon
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

taketoken

Deprecated. Do *not* use. Using this command can result in data loss.

Synopsis

```
nodetool <options> taketoken -- ( <token>, ... )
```

- options are:

    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -- Separates an option from an argument that could be mistaken for a option.
- token is a token to move.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

This command operates on multiple nodes in a cluster and was made available in Cassandra 2.0.6 for moving virtual nodes (vnodes). This command is being removed in the next release. Using this command can result in data loss.

tpstats

Provides usage statistics of thread pools.

Synopsis

```
nodetool <options> tpstats
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password>
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) means not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Run the nodetool tpstats command on the local node. The nodetool tpstats command provides statistics about the number of active, pending, and completed tasks for each stage of Cassandra operations by thread pool. A high number of pending tasks for any pool can indicate performance problems, as described in http://wiki.apache.org/cassandra/Operations#Monitoring .

This table describes the indicators:

Table 11: nodetool tpstats output

Name of statistic	Task	Related information
ReadStage	Local reads	
RequestResponse	Handle responses from other nodes	
MutationStage	Local writes	A high number of pending write requests indicates a problem handling them. Tune hardware or Cassandra configuration.
ReadRepairStage	A digest query and update of replicas of a key	
ReplicateOnWriteStage	Counter writes, replications after a local write	
GossipStage	Handle gossip rounds every second	
AntiEntropyStage	Repair consistency	Nodetool repair
MigrationStage	Make schema changes	
MemoryMeter	Actual object memory including JVM overhead	
MemtablePostFlusher	Flush the commit log and other operations after flushing the memtable	
FlushWriter	Flush the memtable to disk, the status of the sort and write-to-disk operations	Tune hardware or Cassandra configuration.
MiscStage	Miscellaneous operations	
PendingRangeCalculator	Calculate pending ranges per bootstraps and departed nodes	Developer notes
commitlog_archiver	Save the commit log	
InternalResponseStage	Respond to non-client initiated messages, including bootstrapping and schema checking	
HintedHandoff	Send missed mutations to other nodes	

Example

Run the command every two seconds.

```
nodetool -h labcluster tpstats
```

Example output is:

Pool Name	Active	Pending	Completed	Blocked	All time blocked
ReadStage	3	2	19570606	0	0
RequestResponseStage	0	1	10552500	0	0
MutationStage	0	0	11554725	0	0
ReadRepairStage	0	0	124792	0	0
ReplicateOnWriteStage	0	0	0	0	0
GossipStage	0	0	2713	0	0
AntiEntropyStage	0	0	50	0	0
MigrationStage	0	0	0	0	0
MemoryMeter	1	4	545	0	0
MemtablePostFlusher	0	0	417	0	0
FlushWriter	0	0	371	0	0
MiscStage	0	0	25	0	0
PendingRangeCalculator	0	0	16	0	0
commitlog_archiver	0	0	0	0	0
InternalResponseStage	0	0	0	0	0
HintedHandoff	0	0	10	0	0

Message type	Dropped
RANGE_SLICE	0
READ_REPAIR	0
BINARY	0
READ	0
MUTATION	0
_TRACE	0
REQUEST_RESPONSE	0

truncatehints

Truncates all hints on the local node, or truncates hints for the one or more endpoints.

Synopsis

```
nodetool <options> truncatehints -- ( <endpoint> ... )
```

- options are:

  - (-h | --host) <host name> | <ip address>
  - (-p | --port) <port number>

- (-pw | --password) <password >
- (-u | --username) <user name>
- -- Separates an option and argument that could be mistaken for a option.
- endpoint is one or more endpoint IP addresses or host names which hints are to be deleted.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

upgradesstables

Rewrites SSTables for tables that are not running the current version of Cassandra.

Synopsis

```
nodetool <options> upgradesstables
 (-a | --include-all-sstables )
 -- ( <keyspace> <table>  ... )
```

- options are:

    - (-h | --host) <host name> | <ip address>
    - (-p | --port) <port number>
    - (-pw | --password) <password >
    - (-u | --username) <user name>
- -a or --include-all-sstables, followed by the snapshot name.
- -- Separates an option and argument that could be mistaken for a option.
- keyspace a keyspace name.
- table is one or more table names, separated by a space.

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Description

Rebuilds SSTables on a node that are not compatible with the current version. Use this command when upgrading your server or changing compression options.

version

Provides the version number of Cassandra running on the specified node.

Synopsis

```
nodetool <options> version
```

options are:

- (-h | --host) <host name> | <ip address>
- (-p | --port) <port number>
- (-pw | --password) <password >
- (-u | --username) <user name>

Synopsis Legend

- Angle brackets (< >) mean not literal, a variable
- Italics mean optional
- The pipe (|) symbol means OR or AND/OR
- Ellipsis (...) means repeatable
- Orange (and) means not literal, indicates scope

Cassandra bulk loader (sstableloader)

About this task

The Cassandra bulk loader, also called the sstableloader, provides the ability to:

- Bulk load external data into a cluster.
- Load existing SSTables into another cluster with a different number of nodes or replication strategy.
- Restore snapshots.

The sstableloader streams a set of SSTable data files to a live cluster; it does not simply copy the set of SSTables to every node, but transfers the relevant part of the data to each node, conforming to the replication strategy of the cluster. The table into which the data is loaded does not need to be empty.

If tables are repaired in a different cluster, after being loaded, the tables will be unrepaired.

Prerequisites

Because sstableloader uses Cassandra gossip, make sure of the following:

- The `cassandra.yaml` configuration file is in the classpath and properly configured.
- At least one node in the cluster is configured as seed.
- In the `cassandra.yaml` file, the following properties are properly configured for the cluster that you are importing into:
    - cluster_name
    - listen_address
    - storage_port
    - rpc_address
    - rpc_port

When using sstableloader to load external data, you must first generate SSTables.

If using DataStax Enterprise, you can use Sqoop to migrate external data to Cassandra.

Generating SSTables

SSTableWriter is the API to create raw Cassandra data files locally for bulk load into your cluster. The Cassandra source code includes the CQLSSTableWriter implementation for creating SSTable files from external data without needing to understand the details of how those map to the underlying storage engine. Import the `org.apache.cassandra.io.sstable.CQLSSTableWriter` class, and define the schema for the data you want to import, a writer for the schema, and a prepared insert statement, as shown in Cassandra 2.0.1, 2.0.2, and a quick peek at 2.0.3.

Using sstableloader

Before loading the data, you must define the schema of the tables with CQL or Thrift.

To get the best throughput from SSTable loading, you can use multiple instances of sstableloader to stream across multiple machines. No hard limit exists on the number of SSTables that sstableloader can run at the same time, so you can add additional loaders until you see no further improvement.

If you use sstableloader on the same machine as the Cassandra node, you can't use the same network interface as the Cassandra node. However, you can use the JMX **StorageService** > **bulkload() call** from that node. This method takes the absolute path to the directory where the SSTables are located, and loads them just as sstableloader does. However, because the node is both source and destination for the streaming, it increases the load on that node. This means that you should load data from machines that are not Cassandra nodes when loading into a live cluster.

Usage:

Packaged installs:

```
$ sstableloader [options] path_to_keyspace
```

Tarball installs:

```
$ cd install_location/bin
$ sstableloader [options] path_to_keyspace
```

The sstableloader bulk loads the SSTables found in the keyspace directory to the configured target cluster, where the parent directories of the directory path are used as the target keyspace/table name.

1. Go to the location of the SSTables:

   ```
   $ cd /var/lib/cassandra/data/Keyspace1/Standard1/
   ```
2. To view the contents of the keyspace:

   ```
   $ ls

   Keyspace1-Standard1-jb-60-CRC.db
   Keyspace1-Standard1-jb-60-Data.db
   ...
   Keyspace1-Standard1-jb-60-TOC.txt
   ```
3. To bulk load the files, specify the path to `Keyspace1/Standard1/` in the target cluster:

   ```
   $ sstableloader -d 110.82.155.1 /var/lib/cassandra/data/Keyspace1/Standard1/
    ##  package installation

   $ install_location/bin/sstableloader -d 110.82.155.1 /var/lib/cassandra/
   data/Keyspace1/Standard1/ ## tarball installation
   ```

   This bulk loads all files.

Table 12: sstableloader options

Short option	Long option	Description
-alg	--ssl-alg <ALGORITHM>	Client SSL algorithm (default: SunX509).
-ciphers	--ssl-ciphers <CIPHER-SUITES>	Client SSL. Comma-separated list of encryption suites.
-d	--nodes <initial_hosts>	Required. Connect to a list of (comma separated) hosts for initial cluster information.
-f	--conf-path <path_to_config_file>	Path to the `cassandra.yaml` path for streaming throughput and client/server SSL.

Short option	Long option	Description
-h	--help	Display help.
-i	--ignore <NODES>	Do not stream to this comma separated list of nodes.
-ks	--keystore <KEYSTORE>	Client SSL. Full path to the keystore.
-kspw	--keystore-password <KEYSTORE-PASSWORD>	Client SSL. Password for the keystore.
	--no-progress	Do not display progress.
-p	--port <rpc port>	RPC port (default: 9160).
-prtcl	--ssl-protocol <PROTOCOL>	Client SSL. Connections protocol to use (default: TLS).
-pw	--password <password>	Password for Cassandra authentication.
-st	--store-type <STORE-TYPE>	Client SSL. Type of store.
-t	--throttle <throttle>	Throttle speed in Mbits (default: unlimited).
-tf	--transport-factory <transport factory>	Fully-qualified `ITransportFactory` class name for creating a connection to Cassandra.
-ts	--truststore <TRUSTSTORE>	Client SSL. Full path to truststore.
-tspw	--truststore-password <TRUSTSTORE-PASSWORD>	Client SSL. Password of the truststore.
-u	--username <username>	User name for Cassandra authentication.
-v	--verbose	Verbose output.

The following `cassandra.yaml` options can be over-ridden from the command line:

Option in cassandra.yaml	Command line example
stream_throughput_outbound_megabits_per_sec	--throttle 300
server_encryption_options	--ssl-protocol none
client_encryption_options	--keystore-password MyPassword

The cassandra utility

About this task

Cassandra start-up parameters can be run from the command line (in Tarball installations) or specified in the cassandra-env.sh file (Package or Tarball installations).

Usage

For tarballs, run the following from the command line:

`cassandra [OPTIONS]`

For package installs, add the following to the `/etc/cassandra/cassandra-env.sh` file:

`JVM_OPTS="$JVM_OPTS -D[PARAMETER]`

Include file

For convenience on Linux installations, Cassandra uses an include file, `cassandra.in.sh`, to source these environment variables. Use the following locations for this file:

- Tarball installs: `install_location/bin`
- Packaged installs: `/usr/share/cassandra`

Cassandra also uses the Java options set in `cassandra-env.sh`. If you want to pass additional options to the Java virtual machine, such as maximum and minimum heap size, edit the options in this file rather than setting them in the environment.

General options

Option	Description
-f	Start the cassandra process in foreground. The default is to start as background process.
-h	Help.
-p *filename*	Log the process ID in the named file. Useful for stopping Cassandra by killing its PID.
-v	Print the version and exit.

Start-up parameters

-D *parameter*

Specifies the following start-up parameters:

cassandra.available_processors=*number_of_processors*

In a multi-instance deployment, multiple Cassandra instances will independently assume that all CPU processors are available to it. This setting allows you to specify a smaller set of processors and perhaps have affinity.

cassandra.config=*directory*

The directory location of the `cassandra.yaml` file.

cassandra.initial_token=*token*

Sets the initial partitioner token for a node the first time the node is started.

cassandra.join_ring=*true|false*

Set to false to start Cassandra on a node but not have the node join the cluster.

cassandra.load_ring_state=*true* |false

Set to false to clear all gossip state for the node on restart. Use when you have changed node information in `cassandra.yaml` (such as listen_address).

cassandra.metricsReporterConfigFile=*file*

Enable pluggable metrics reporter. See Pluggable metrics reporting in Cassandra 2.0.2.

cassandra.native_transport_port=port

Set the port on which the CQL native transport listens for clients. (Default: 9042)

cassandra.partitioner=*partitioner*

Set the partitioner. (Default: `org.apache.cassandra.dht.Murmur3Partitioner`)

cassandra.renew_counter_id=*true/false*

Set to true to reset local counter info on a node. Used to recover from data loss to a counter table.

1. Remove all SSTables for counter tables on the node.
2. Restart the node with the value of this parameter set to true.
3. Run nodetool repair once the node is up again.

cassandra.replace_address=*listen_address or broadcast_address of dead node*

To replace a node that has died, restart a new node in its place specifying the listen_address or broadcast_address that the new node is assuming. The new node must not have any data in its data directory, that is, it must be in the same state as before bootstrapping.

Note: The broadcast_address defaults to the listen_address except when using the Ec2MultiRegionSnitch.

cassandra.replayList=*table*

Allow restoring specific tables from an archived commit log.

cassandra.ring_delay_ms=*ms*

Allows overriding of the default RING_DELAY (1000ms), which is the amount of time a node waits before joining the ring.

cassandra.rpc_port=*port*

Set the port for the Thrift RPC service, which is used for client connections. (Default: 9160).

cassandra.ssl_storage_port=*port*

Set the SSL port for encrypted communication. (Default: 7001)

cassandra.start_native_transport=*true/false*

Enable or disable the native transport server. See start_native_transport in `cassandra.yaml`.

cassandra.start_rpc=*true/false*

Enable or disable the Thrift RPC server. (Default: true)

cassandra.storage_port=*port*

Set the port for inter-node communication. (Default: 7000)

cassandra.triggers_dir=*directory*

Set the default location for the trigger JARs. (Default: `conf/triggers`)

cassandra.write_survey=true

For testing new compaction and compression strategies. It allows you to experiment with different strategies and benchmark write performance differences without affecting the production workload. See Testing compaction and compression.

Example

- **Clear gossip state when starting a node. (Useful when the node has changed its configuration, such as its listen IP address.)**

  Command line: `bin/cassandra -Dcassandra.load_ring_state=false`

  cassandra-env.sh: `JVM_OPTS="$JVM_OPTS -Dcassandra.load_ring_state=false"`

- **Start Cassandra on a node and do not join the cluster when already configured in the `cassandra.yaml` file:**

  Command line: `bin/cassandra -Dcassandra.join_ring=false`

  cassandra-env.sh: `JVM_OPTS="$JVM_OPTS -Dcassandra.join_ring=false"`

- **Replacing a dead node:**

  Command line: `bin/cassandra -Dcassandra.replace_address=10.91.176.160`

cassandra-env.sh: `JVM_OPTS="$JVM_OPTS -Dcassandra.replace_address=10.91.176.160"`

Related topics

The cassandra.yaml configuration file

The cassandra-stress tool

About this task

The cassandra-stress tool is a Java-based stress testing utility for benchmarking and load testing a Cassandra cluster.

The binary installation of the tool also includes a daemon, which in larger-scale testing can prevent potential skews in the test results by keeping the JVM warm.

Modes of operation:

- **Inserting**: Loads test data.
- **Reading**: Reads test data.
- **Indexed range slicing**: Works with RandomPartitioner on indexed tables.

The cassandra-stress tool creates a keyspace called Keyspace1 and within that, tables named Standard1, Super1, Counter1, and SuperCounter1, depending on what type of table is being tested. These are automatically created the first time you run the stress test and will be reused on subsequent runs unless you drop the keyspace using CQL. It is not possible to change the names; they are hard-coded.

Commands:

- Packaged installs: `cassandra-stress [options]`
- Tarball installs: `install_location/tools/bin/cassandra-stress [options]`

You can use these modes with or without the cassandra-stress daemon running (binary installs only).

Options for cassandra-stress

Short option	Long option	Description
-V	--average-size-values	Generate column values of average rather than specific size.
-C <CARDINALITY>	--cardinality <CARDINALITY>	Number of unique values stored in columns. Default is 50.
-c <COLUMNS>	--columns <COLUMNS>	Number of columns per key. Default is 5.
-S <COLUMN-SIZE>	--column-size <COLUMN-SIZE>	Size of column values in bytes. Default is 34.
-Z <COMPACTION-STRATEGY>	--compaction-strategy <COMPACTION-STRATEGY>	Specifies which compaction strategy to use.
-U <COMPARATOR>	--comparator <COMPARATOR>	Specifies which column comparator to use. Supported types are: TimeUUIDType, AsciiType, and UTF8Type.
-I <COMPRESSION>	--compression <COMPRESSION>	Specifies the compression to use for SSTables. Default is no compression.

Short option	Long option	Description
-e <CONSISTENCY-LEVEL>	--consistency-level <CONSISTENCY-LEVEL>	Consistency level to use.
-x <CREATE-INDEX>	--create-index <CREATE-INDEX>	Type of index to create on columns (KEYS).
-L l	--enable-cql	Perform queries using CQL (Cassandra Query Language).
-y <TYPE>	--family-type <TYPE>	Sets the table type.
-f <FILE>	--file <FILE>	Write output to a given file.
-h	--help	Show help.
-k	--keep-going	Ignore errors when inserting or reading. When set, --keep-trying has no effect. Default is false.
-K <KEEP-TRYING>	--keep-trying <KEEP-TRYING>	Retry on-going operation N times (in case of failure). Use a positive integer. The default is 10.
-g <KEYS-PER-CALL>	-g, --keys-per-call <KEYS-PER-CALL>	Number of keys to per call. Default is 1000.
-d <NODES>	--nodes <NODES>	Nodes to perform the test against. Must be comma separated with no spaces. Default is localhost.
-D <NODESFILE>	--nodesfile <NODESFILE>	File containing host nodes (one per line).
-W	--no-replicate-on-write	Set replicate_on_write to false for counters. Only for counters with a consistency level of ONE (CL=ONE).
-F <NUM-DIFFERENT-KEYS>	--num-different-keys <NUM-DIFFERENT-KEYS>	Number of different keys. If less than NUM-KEYS, the same key is re-used multiple times. Default is NUM-KEYS.
-n <NUMKEYS>	--num-keys <NUMKEYS>	Number of keys to write or read. Default is 1,000,000.
-o <OPERATION>	--operation <OPERATION>	Operation to perform: INSERT, READ, INDEXED_RANGE_SLICE, MULTI_GET, COUNTER_ADD, COUNTER_GET. Default is INSERT.
-p <PORT>	--port <PORT>	Thrift port. Default is 9160.
-i <PROGRESS-INTERVAL>	--progress-interval <PROGRESS-INTERVAL>	The interval, in seconds, at which progress is output. Default is 10 seconds.
-Q <QUERY-NAMES>	--query-names <QUERY-NAMES>	Comma-separated list of column names to retrieve from each row.
-r	--random	Use random key generator. When used --stdev has no effect. Default is false.
-l <REPLICATION-FACTOR>	--replication-factor <REPLICATION-FACTOR>	Replication Factor to use when creating tables. Default is 1.

Short option	Long option	Description
-R <REPLICATION-STRATEGY>	--replication-strategy <REPLICATION-STRATEGY>	Replication strategy to use (only on insert and when a keyspace does not exist.) The default is SimpleStrategy.
-T <SEND-TO>	--send-to <SEND-TO>	Sends the command as a request to the cassandra-stressd daemon at the specified IP address. The daemon must already be running at that address.
-N <SKIP-KEYS>	--skip-keys <SKIP-KEYS>	Fraction of keys to skip initially. Default is 0.
-s <STDEV>	--stdev <STDEV>	Standard deviation. Default is 0.1.
-O <STRATEGY-PROPERTIES>	--strategy-properties <STRATEGY-PROPERTIES>	Replication strategy properties in the following format: <dc_name>:<num>,<dc_name>:<num>, ... For use with NetworkTopologyStrategy.
-t <THREADS>	--threads <THREADS>	Number of threads to use. Default is 50.
-m	--unframed	Use unframed transport. Default is false.
-P	--use-prepared-statements	(CQL only) Perform queries using prepared statements.

Using the Daemon Mode

About this task

The daemon in larger-scale testing can prevent potential skews in the test results by keeping the JVM warm.

Run the daemon from:

install_location/tools/bin/cassandra-stressd start|stop|status [-h <host>]

During stress testing, you can keep the daemon running and send it commands through it using the --send-to option.

Example

- Insert 1,000,000 rows to given host:

  /tools/bin/cassandra-stress –d 192.168.1.101

  When the number of rows is not specified, one million rows are inserted.
- Read 1,000,000 rows from given host:

  tools/bin/cassandra-stress –d 192.168.1.101 –o read
- Insert 10,000,000 rows across two nodes:

  /tools/bin/cassandra-stress –d 192.168.1.101,192.168.1.102 –n 10000000
- Insert 10,000,000 rows across two nodes using the daemon mode:

  /tools/bin/cassandra-stress –d 192.168.1.101,192.168.1.102 –n 10000000 --send-to 54.0.0.1

Interpreting the output of cassandra-stress

Each line reports data for the interval between the last elapsed time and current elapsed time, which is set by the --progress-interval option (default 10 seconds).

```
7251,725,725,56.1,95.1,191.8,10
19523,1227,1227,41.6,86.1,189.1,21
41348,2182,2182,22.5,75.7,176.0,31
...
```

Data	Description
total	Total number of operations since the start of the test.
interval_op_rate	Number of operations per second performed during the interval.
interval_key_rate	Number of keys/rows read or written per second during the interval (normally be the same as interval_op_rate unless doing range slices).
latency	Average latency in milliseconds for each operation during that interval.
95th	95% of the time the latency was less than the number displayed in the column (Cassandra 1.2 or later).
99th	99% of the time the latency was less than the number displayed in the column (Cassandra 1.2 or later).
elapsed	Number of seconds elapsed since the beginning of the test.

The sstablescrub utility

About this task

The sstablescrub utility is an offline version of nodetool scrub. It attempts to remove the corrupted parts while preserving non-corrupted data.

If scrubbing results in dropping rows, new SSTables become unrepaired. However, if no bad rows are detected, the SSTable keeps its origin repairedAt field, which denotes the time of repair. If the entire table is unreadable, it will not be able to repair it.

Procedure

1. Before using sstablescrub, try rebuilding the tables using nodetool scrub.

   If nodetool scrub does not fix the problem, use this utility. Because it runs offline, sstablescrub can correct errors that nodetool scrub cannot.
2. Shut down the node.
3. Run the utility:

   - Packaged installs: sstablescrub [options] <keyspace> <table>
   - Tarball installs: *install_location*/bin/sstablescrub [options] <keyspace> <table>

Table 13: Options

Flag	Option	Description
	--debug	Display stack traces.
-h	--help	Display help.

Flag	Option	Description
-m	--manifest-check	Only check and repair the leveled manifest, without actually scrubbing the SSTables.
-v	--verbose	Verbose output.

4. After running this utility, you must perform a repair because corrupted rows are thrown away.

The sstablesplit utility

About this task

Use this tool to split SSTables files into multiple SSTables of a maximum designated size. For example, if SizeTieredCompactionStrategy was used for a major compaction and results in a excessively large SSTable, it's a good idea to split the table because won't get compacted again until the next huge compaction.

Cassandra must be stopped to use this tool:

- Packaged installs:

```
$ sudo service cassandra stop
```
- Tarball installs:

```
$ ps auwx | grep cassandra
$ sudo kill pid
```

Usage:

- Packaged installs: `sstablessplit [options] <filename> [<filename>]*`
- Tarball installs: *install_location*`/bin/sstablessplit [options] <filename> [<filename>]*`

Example:

```
$ sstablesplit -s 40 /var/lib/cassandra/data/Keyspace1/Standard1/*
```

Table 14: Options

Flag	Option	Description
	--debug	Display stack traces.
-h	--help	Display help.
	--no-snapshot	Do not snapshot the SSTables before splitting.
-s	--size <size>	Maximum size in MB for the output SSTables (default: 50).
-v	--verbose	Verbose output.

sstablekeys

About this task

The sstablekeys utility dumps table keys. To list the keys in an SSTable, find the name of the SSTable file. The file is located in the data directory and has a `.db` extension. The location of the data directory, listed in

the "Install locations" section, depends on the type of installation. After finding the name of the file, use the name as an argument to the `sstablekeys` command.

```
$ bin/sstablekeys <sstable_name>
```

Procedure

1. Create the playlists table in the music keyspace as shown in Data modeling.

2. Insert the row of data about ZZ Top in playlists:

```
INSERT INTO music.playlists (id, song_order, song_id, title, artist, album)
   VALUES (62c36092-82a1-3a00-93d1-46196ee77204,
   1,
   a3e64f8f-bd44-4f28-b8d9-6938726e34d4,
   'La Grange',
   'ZZ Top',
   'Tres Hombres');
```

3. Flush the data to disk.

```
$ nodetool flush music playlists
```

4. Look at keys in the SSTable data. For example, use sstablekeys followed by the path to the data. Use the path to data for your Cassandra installation:

```
$ sstablekeys <path to data>/data/data/music/
   playlists-8b9f4cc0229211e4b02073ded3cb6170/music-playlists-ka-1-Data.db
```

The output appears, for example:

```
62c3609282a13a0093d146196ee77204
```

The sstableupgrade tool

About this task

This tool rewrites the SSTables in the specified table or snapshot to match the currently installed version of Cassandra.

If restoring with sstableloader, you must upgrade your snapshots before restoring for any snapshot taken in a major version older than the major version that Cassandra is currently running.

Usage:

- Packaged installs: `sstableupgrade [options] <keyspace> <cf> [snapshot]`
- Tarball installs: `install_location/bin/sstableupgrade [options] <keyspace> <cf> [snapshot]`

The snapshot option only upgrades the specified snapshot.

Table 15: Options

Flag	Option	Description
	--debug	Display stack traces.
-h	--help	Display help.

References

Starting and stopping Cassandra

Starting Cassandra as a service

About this task

Start the Cassandra Java server process for packaged installations. Startup scripts are provided in `/etc/init.d`. The service runs as the *cassandra* user.

Procedure

You must have root or sudo permissions to start Cassandra as a service.

On initial start-up, each node must be started one at a time, starting with your seed nodes:

```
$ sudo service cassandra start
```

On Enterprise Linux systems, the Cassandra service runs as a java process.

Starting Cassandra as a stand-alone process

About this task

Start the Cassandra Java server process for tarball installations.

Procedure

On initial start-up, each node must be started one at a time, starting with your seed nodes.

- To start Cassandra in the background:

```
$ cd install_location
$ bin/cassandra
```

- To start Cassandra in the foreground:

```
$ cd install_location
$ bin/cassandra -f
```

Stopping Cassandra as a service

About this task

Stop the Cassandra Java server process on packaged installations.

Procedure

You must have root or sudo permissions to stop the Cassandra service:

197

```
$ sudo service cassandra stop
```

Stopping Cassandra as a stand-alone process

About this task

Stop the Cassandra Java server process on packaged installations.

Procedure

Find the Cassandra Java process ID (PID), and then kill the process using its PID number:

```
$ ps auwx | grep cassandra
$ sudo kill pid
```

Clearing the data as a service

About this task

Remove all data from a package installation.

Procedure

To clear the data from the **default** directories:

After stopping the service, run the following command:

```
$ sudo rm -rf /var/lib/cassandra/*
```

Note: If you are clearing data from an AMI installation for restart, you need to preserve the log files.

Clearing the data as a stand-alone process

About this task

Removes all data from a tarball installation.

Procedure

To clear all data from the **default** directories, including the commitlog and saved_caches:

1. After stopping the process, run the following command from the install directory:

```
$ cd install_location
$ sudo rm -rf data/*
```

2. To clear the only the data directory:

```
$ cd install_location
$ sudo rm -rf data/data/*
```

Install locations

Tarball installation directories

The configuration files are located in the following directories:

Configuration Files	Locations
cassandra.yaml	*install_location*/conf
cassandra-topology.properties	*install_location*/conf
cassandra-rackdc.properties	*install_location*/conf
cassandra-env.sh	*install_location*/conf
cassandra.in.sh	*install_location*/bin

The binary tarball releases install into the installation directory.

Directories	Description
bin	Utilities and start scripts
conf	Configuration files and environment settings
interface	Thrift and Avro client APIs
javadoc	Cassandra Java API documentation
lib	JAR and license files

For DataStax Enterprise installs, see Configuration File Locations.

Package installation directories

The configuration files are located in the following directories:

Configuration Files	Locations
cassandra.yaml	/etc/cassandra
cassandra-topology.properties	/etc/cassandra
cassandra-rackdc.properties	/etc/cassandra
cassandra-env.sh	/etc/cassandra
cassandra.in.sh	/usr/share/cassandra

The packaged releases install into these directories:

Directories	Description
/var/lib/cassandra	Data directories
/var/log/cassandra	Log directory
/var/run/cassandra	Runtime files
/usr/share/cassandra	Environment settings
/usr/share/cassandra/lib	JAR files
/usr/bin	Binary files

199

Directories	Description
/usr/sbin	
/etc/cassandra	Configuration files
/etc/init.d	Service startup script
/etc/security/limits.d	Cassandra user limits
/etc/default	

For DataStax Enterprise installs, see Configuration File Locations.

Cassandra-CLI utility (deprecated)

Cassandra stores storage configuration attributes in the system keyspace. You set storage configuration attributes on a per-keyspace or per-table basis programmatically or using a client application, such as CLI or Thrift.

Important: The CLI utility is **deprecated** and will be removed in Cassandra 3.0. For ease of use and performance, switch from Thrift and CLI to CQL and cqlsh.

Keyspace attributes

A keyspace must have a user-defined name, a replica placement strategy, and options that specify the number of copies per data center or node.

name

Required. The name for the keyspace.

placement_strategy

Required. Determines how Cassandra distributes replicas for a keyspace among nodes in the ring. Values are:

- SimpleStrategy or org.apache.cassandra.locator.SimpleStrategy
- NetworkTopologyStrategy or org.apache.cassandra.locator.NetworkTopologyStrategy

NetworkTopologyStrategy requires a snitch to be able to determine rack and data center locations of a node. For more information about replication placement strategy, see Data replication.

strategy_options

Specifies configuration options for the chosen replication strategy class. The replication factor option is the total number of replicas across the cluster. A replication factor of 1 means that there is only one copy of each row on one node. A replication factor of 2 means there are two copies of each row, where each copy is on a different node. All replicas are equally important; there is no primary or master replica. As a general rule, the replication factor should not exceed the number of nodes in the cluster. However, you can increase the replication factor and then add the desired number of nodes.

When the replication factor exceeds the number of nodes, writes are rejected, but reads are served as long as the desired consistency level can be met.

For more information about configuring the replication placement strategy for a cluster and data centers, see Choosing keyspace replication options.

durable_writes

(Default: true) When set to false, data written to the keyspace bypasses the commit log. Be careful using this option because you risk losing data.

Table attributes

The following attributes can be declared per table.

bloom_filter_fp_chance

See CQL properties in *CQL for Cassandra 2.0.*

bucket_high

See CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

bucket_low

See CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

caching

See CQL properties in *CQL for Cassandra 2.0.*

chunk_length_kb

See CQLCompression Subproperties in *CQL for Cassandra 2.0.*

column_metadata

(Default: N/A - container attribute) Column metadata defines these attributes of a column:

- name: Binds a validation_class and (optionally) an index to a column.
- validation_class: Type used to check the column value.
- index_name: Name of the index.
- index_type: Type of index. Currently the only supported value is KEYS.

Setting a value for the name option is required. The validation_class is set to the default_validation_class of the table if you do not set the validation_class option explicitly. The value of index_type must be set to create an index for a column. The value of index_name is not valid unless index_type is also set.

Setting and updating column metadata with the Cassandra-CLI utility requires a slightly different command syntax than other attributes; note the brackets and curly braces in this example:

```
[default@demo ] UPDATE COLUMN FAMILY users WITH  comparator =UTF8Type
AND  column_metadata =[{column_name: full_name, validation_class: UTF8Type,
 index_type: KEYS }];
```

column_type

(Default: Standard) The standard type of table contains regular columns.

comment

See CQL properties in *CQL for Cassandra 2.0.*

compaction_strategy

See `compaction` in CQL properties in *CQL for Cassandra 2.0.*

comparator

(Default: BytesType) Defines the data types used to validate and sort column names. There are several built-in column comparators available. The comparator cannot be changed after you create a table.

compare_subcolumns_with

(Default: BytesType) Required when the column_type attribute is set to Super. Same as comparator but for the sub-columns of a super column. Deprecated as of Cassandra 1.0, but can still be declared for backward compatibility.

compression_options

(Default: N/A - container attribute) Sets the compression algorithm and sub-properties for the table. Choices are:

- sstable_compression
- chunk_length_kb
- crc_check_chance

crc_check_chance

See CQLCompression Subproperties in *CQL for Cassandra 2.0.*

default_time_to_live

See CQL properties in *CQL for Cassandra 2.0.*

default_validation_class

(Default: N/A) Defines the data type used to validate column values. There are several built-in column validators available.

dclocal_read_repair_chance

See CQLCompression Subproperties in *CQL for Cassandra 2.0.*

gc_grace

See CQL properties in *CQL for Cassandra 2.0.*

index_interval

See CQL properties in *CQL for Cassandra 2.0.*

key_validation_class

(Default: N/A) Defines the data type used to validate row key values. There are several built-in key validators available, however `CounterColumnType` (distributed counters) cannot be used as a row key validator.

max_compaction_threshold

See max_threshold in CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

min_compaction_threshold

See min_threshold in CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

memtable_flush_after_mins

Deprecated as of Cassandra 1.0, but can still be declared for backward compatibility. Use commitlog_total_space_in_mb.

memtable_flush_period_in_ms

See CQL properties in *CQL for Cassandra 2.0.*

memtable_operations_in_millions

Deprecated as of Cassandra 1.0, but can still be declared for backward compatibility. Use commitlog_total_space_in_mb.

memtable_throughput_in_mb

Deprecated as of Cassandra 1.0, but can still be declared for backward compatibility. Use commitlog_total_space_in_mb.

min_sstable_size

See CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

name

(Default: N/A) Required. The user-defined name of the table.

populate_io_cache_on_flush

See CQLCompression Subproperties in *CQL for Cassandra 2.0.*

read_repair_chance

See CQL properties in *CQL for Cassandra 2.0.*

replicate_on_write

See CQLCompression Subproperties in *CQL for Cassandra 2.0.*

speculative_retry

See CQL properties in *CQL for Cassandra 2.0.*

sstable_size_in_mb

See CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

sstable_compression

See compression in CQL properties in *CQL for Cassandra 2.0.*

tombstone_compaction_interval

See CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

tombstone_threshold

See CQL Compaction Subproperties in *CQL for Cassandra 2.0.*

Moving data to or from other databases

Cassandra offers several solutions for migrating from other databases:

- The COPY command, which mirrors what the PostgreSQL RDBMS uses for file/export import.
- The Cassandra bulk loader provides the ability to bulk load external data into a cluster.

About the COPY command

You can use COPY in Cassandra's CQL shell to load flat file data into Cassandra (nearly all RDBMS's have unload utilities that allow table data to be written to OS files) as well as data to be written out to OS files.

ETL Tools

If you need more sophistication applied to a data movement situation (more than just extract-load), then you can use any number of extract-transform-load (ETL) solutions that now support Cassandra. These tools provide excellent transformation routines that allow you to manipulate source data in literally any way you need and then load it into a Cassandra target. They also supply many other features such as visual, point-and-click interfaces, scheduling engines, and more.

Many ETL vendors who support Cassandra supply community editions of their products that are free and able to solve many different use cases. Enterprise editions are also available that supply many other compelling features that serious enterprise data users need.

You can freely download and try ETL tools from Jaspersoft, Pentaho, and Talend that all work with community Cassandra.

Troubleshooting

Peculiar Linux kernel performance problem on NUMA systems

Problems due to zone_reclaim_mode. The Linux kernel can be inconsistent in enabling/disabling zone_reclaim_mode. This can result in odd performance problems:

- Random huge CPU spikes resulting in large increases in latency and throughput.
- Programs hanging indefinitely apparently doing nothing.
- Symptoms appearing and disappearing suddenly.
- After a reboot, the symptoms generally do not show again for some time.

To ensure that zone_reclaim_mode is disabled:

```
echo 0 > /proc/sys/vm/zone_reclaim_mode
```

Reads are getting slower while writes are still fast

The cluster's IO capacity is not enough to handle the write load it is receiving.

Check the SSTable counts in cfstats. If the count is continually growing, the cluster's IO capacity is not enough to handle the write load it is receiving. Reads have slowed down because the data is fragmented across many SSTables and compaction is continually running trying to reduce them. Adding more IO capacity, either via more machines in the cluster, or faster drives such as SSDs, will be necessary to solve this.

If the SSTable count is relatively low (32 or less) then the amount of file cache available per machine compared to the amount of data per machine needs to be considered, as well as the application's read pattern. The amount of file cache can be formulated as ($TotalMemory - JVMHeapSize$) and if the amount of data is greater and the read pattern is approximately random, an equal ratio of reads to the cache:data ratio will need to seek the disk. With spinning media, this is a slow operation. You may be able to mitigate many of the seeks by using a key cache of 100%, and a small amount of row cache (10000-20000) if you have some hot rows and they are not extremely large.

Nodes seem to freeze after some period of time

Some portion of the JVM is being swapped out by the operating system (OS).

Check your system.log for messages from the `GCInspector`. If the `GCInspector` is indicating that either the `ParNew` or `ConcurrentMarkSweep` collectors took longer than 15 seconds, there is a high probability that some portion of the JVM is being swapped out by the OS.

One way this might happen is if the `mmap DiskAccessMode` is used without JNA support. The address space will be exhausted by `mmap`, and the OS will decide to swap out some portion of the JVM that isn't in use, but eventually the JVM will try to GC this space. Adding the JNA libraries will solve this (they cannot be shipped with Cassandra due to carrying a GPL license, but are freely available) or the `DiskAccessMode` can be switched to `mmap_index_only`, which as the name implies will only `mmap` the indices, using much less address space.

DataStax strongly recommends that you disable swap entirely (`sudo swapoff --all`). Because Cassandra has multiple replicas and transparent failover, it is preferable for a replica to be killed

immediately when memory is low rather than go into swap. This allows traffic to be immediately redirected to a functioning replica instead of continuing to hit the replica that has high latency due to swapping. If your system has a lot of DRAM, swapping still lowers performance significantly because the OS swaps out executable code so that more DRAM is available for caching disks. To make this change permanent, remove all swap file entries from `/etc/fstab`.

If you insist on using swap, you can set `vm.swappiness=1`. This allows the kernel swap out the absolute least used parts.

If the GCInspector isn't reporting very long GC times, but is reporting moderate times frequently (`ConcurrentMarkSweep` taking a few seconds very often) then it is likely that the JVM is experiencing extreme GC pressure and will eventually OOM. See the section below on OOM errors.

You must disable swap entirely. Failure to do so can severely lower performance. Because Cassandra has multiple replicas and transparent failover, it is preferable for a replica to be killed immediately when memory is low rather than go into swap. This allows traffic to be immediately redirected to a functioning replica instead of continuing to hit the replica that has high latency due to swapping. If your system has a lot of DRAM, swapping still lowers performance significantly because the OS swaps out executable code so that more DRAM is available for caching disks.

Nodes are dying with OOM errors

Nodes are dying with OutOfMemory exceptions. Check for these typical causes:

Row cache is too large, or is caching large rows

Row cache is generally a high-end optimization. Try disabling it and see if the OOM problems continue.

The memtable sizes are too large for the amount of heap allocated to the JVM

You can expect N + 2 memtables resident in memory, where N is the number of tables. Adding another 1GB on top of that for Cassandra itself is a good estimate of total heap usage.

If none of these seem to apply to your situation, try loading the heap dump in MAT and see which class is consuming the bulk of the heap for clues.

Nodetool or JMX connections failing on remote nodes

Nodetool commands can be run locally but not on other nodes in the cluster.

If you can run nodetool commands locally but not on other nodes in the ring, you may have a common JMX connection problem that is resolved by adding an entry like the following in *install_location*/conf/`cassandra-env.sh` on each node:

```
JVM_OPTS = "$JVM_OPTS -Djava.rmi.server.hostname=<public name>"
```

If you still cannot run nodetool commands remotely after making this configuration change, do a full evaluation of your firewall and network security. The nodetool utility communicates through JMX on port 7199.

View of ring differs between some nodes

Indicates that the ring is in a bad state.

This situation can happen when not using virtual nodes (vnodes) and there are token conflicts (for instance, when bootstrapping two nodes simultaneously with automatic token selection.) Unfortunately, the only way to resolve this is to do a full cluster restart. A rolling restart is insufficient since gossip from nodes with the bad state will repopulate it on newly booted nodes.

Java reports an error saying there are too many open files

Java may not have open enough file descriptors.

Cassandra generally needs more than the default (1024) amount of file descriptors. To increase the number of file descriptors, change the security limits on your Cassandra nodes as described in the Recommended Settings section of Insufficient user resource limits errors .

Another, much less likely possibility, is a file descriptor leak in Cassandra. Run `lsof -n | grep java` to check that the number of file descriptors opened by Java is reasonable and reports the error if the number is greater than a few thousand.

Insufficient user resource limits errors

Insufficient resource limits may result in a number of errors in Cassandra and OpsCenter.

Cassandra errors

Insufficient as (address space) or memlock setting

```
ERROR        [SSTableBatchOpen:1        ]        2012-07-25        15:46:02,913
  AbstractCassandraDaemon.java  (line 139)
Fatal exception in thread Thread [SSTableBatchOpen:1,5,main ]
java.io.IOError: java.io.IOException: Map failed  at ...
```

Insufficient memlock settings

```
WARN [main ] 2011-06-15 09:58:56,861 CLibrary.java  (line 118) Unable to lock
  JVM memory  (ENOMEM).
This can result in part of the JVM being swapped out, especially with mmapped
  I/O enabled.
Increase RLIMIT_MEMLOCK or run Cassandra as root.
```

Insufficient nofiles setting

```
WARN 05:13:43,644 Transport error occurred during acceptance of message.
org.apache.thrift.transport.TTransportException: java.net.SocketException:
Too many open files ...
```

Insufficient nofiles setting

```
ERROR [MutationStage:11 ] 2012-04-30 09:46:08,102 AbstractCassandraDaemon.java
   (line 139)
Fatal exception in thread Thread [MutationStage:11,5,main ]
java.lang.OutOfMemoryError: unable to create new native thread
```

Recommended settings

You can view the current limits using the `ulimit -a` command. Although limits can also be temporarily set using this command, DataStax recommends making the changes permanent:

Packaged installs: Ensure that the following settings are included in the `/etc/security/limits.d/cassandra.conf` file:

```
cassandra - memlock unlimited
cassandra - nofile 100000
cassandra - nproc 32768
cassandra - as unlimited
```

Tarball installs: Ensure that the following settings are included in the `/etc/security/limits.conf` file:

```
* - memlock unlimited
```

```
* - nofile 100000
* - nproc 32768
* - as unlimited
```

If you run Cassandra as root, some Linux distributions such as Ubuntu, require setting the limits for root explicitly instead of using *:

```
root - memlock unlimited
root - nofile 100000
root - nproc 32768
root - as unlimited
```

For CentOS, RHEL, OEL systems, also set the nproc limits in `/etc/security/limits.d/90-nproc.conf`:

```
* - nproc 32768
```

For all installations, add the following line to `/etc/sysctl.conf`:

```
vm.max_map_count = 131072
```

To make the changes take effect, reboot the server or run the following command:

```
$ sudo sysctl -p
```

To confirm the limits are applied to the Cassandra process, run the following command where *pid* is the process ID of the currently running Cassandra process:

```
$ cat /proc/<pid>/limits
```

OpsCenter errors

See the OpsCenter Troubleshooting documentation.

Cannot initialize class org.xerial.snappy.Snappy

An error may occur when Snappy compression/decompression is enabled although its library is available from the classpath.

```
java.util.concurrent.ExecutionException: java.lang.NoClassDefFoundError:
    Could not initialize class org.xerial.snappy.Snappy
...
Caused by: java.lang.NoClassDefFoundError: Could not initialize class
 org.xerial.snappy.Snappy
   at
 org.apache.cassandra.io.compress.SnappyCompressor.initialCompressedBufferLength
       (SnappyCompressor.java:39)
```

The native library `snappy-1.0.4.1-libsnappyjava.so` for Snappy compression is included in the `snappy-java-1.0.4.1.jar` file. When the JVM initializes the JAR, the library is added to the default temp directory. If the default temp directory is mounted with a noexec option, it results in the above exception.

One solution is to specify a different temp directory that has already been mounted without the noexec option, as follows:

- If you use the DSE/Cassandra command `$_BIN/dse cassandra` or `$_BIN/cassandra`, simply append the command line:
  - DSE: `bin/dse cassandra -t -Dorg.xerial.snappy.tempdir=/path/to/newtmp`
  - Cassandra: `bin/cassandra -Dorg.xerial.snappy.tempdir=/path/to/newtmp`
- If starting from a package using service dse start or service cassandra start, add a system environment variable *JVM_OPTS* with the value:
  ```
  JVM_OPTS=-Dorg.xerial.snappy.tempdir=/path/to/newtmp
  ```

The default `cassandra-env.sh` looks for the variable and appends to it when starting the JVM.

Firewall idle connection timeout causing nodes to lose communication during low traffic times

About this task

During low traffic intervals, a firewall configured with an idle connection timeout can close connections to local nodes and nodes in other data centers. The default idle connection timeout is usually 60 minutes and configurable by the network administrator.

Procedure

To prevent connections between nodes from timing out, set the TCP keep alive variables:

1. Get a list of available kernel variables:

```
$ sysctl -A | grep net.ipv4
```

The following variables should exist:

- *net.ipv4.tcp_keepalive_time*

  Time of connection inactivity after which the first keep alive request is sent.
- *net.ipv4.tcp_keepalive_probes*

  Number of keep alive requests retransmitted before the connection is considered broken.
- *net.ipv4.tcp_keepalive_intvl*

  Time interval between keep alive probes.

2. To change these settings:

```
$ sudo sysctl -w net.ipv4.tcp_keepalive_time=60
 net.ipv4.tcp_keepalive_probes=3 net.ipv4.tcp_keepalive_intvl=10
```

This sample command changes TCP keepalive timeout to 60 seconds with 3 probes, 10 seconds gap between each. This setting detects dead TCP connections after 90 seconds (60 + 10 + 10 + 10). There is no need to be concerned about the additional traffic as it's negligible and permanently leaving these settings shouldn't be an issue.

DataStax Community release notes

Apache Cassandra 2.0.10 CHANGES.txt lists the changes in this release.

You can view version changes by branch or tag in the **branch** drop-down list:

Tips for using DataStax documentation

Navigating the documents

To navigate, use the table of contents or search in the left navigation bar. Additional controls are:

⇥	Hide or display the left navigation.
« »	Go back or forward through the topics as listed in the table of contents.
✎	Toggle highlighting of search terms.
🖨	Print page.
🐦	See doc tweets and provide feedback.
⦂	Grab to adjust the size of the navigation pane.
¶	Appears on headings for bookmarking. Right-click the ¶ to get the link.
⊕	Toggles the legend for CQL statements and nodetool options.

Other resources

You can find more information and help at:

- Documentation home page
- Datasheets
- Webinars
- Whitepapers
- Developer blogs
- Support