# 快速安装

1. 准备4个节点：192.168.5.126，192.168.5.127，192.168.5.118，192.168.5.134；
2. 安装JDK8，配置JAVA\_HOEM；
3. 关闭防火墙

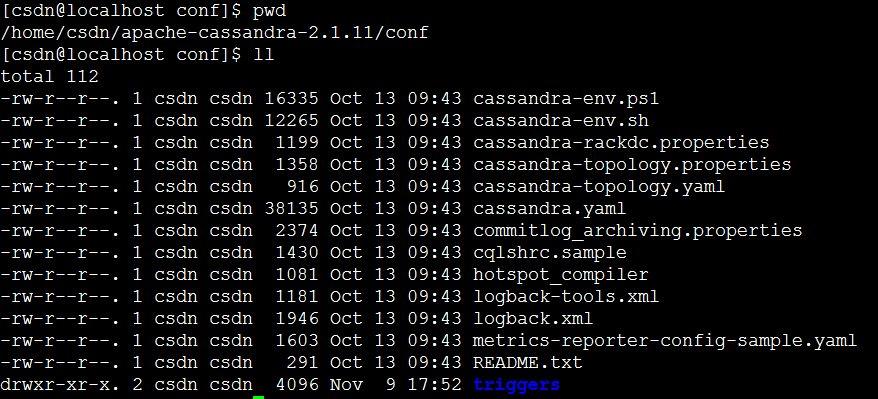
Service iptables stop #关闭防火墙

Chkconfig iptables off #开机不启动防火墙

1. Apache 官网下载稳定发布版本apache-cassandra-2.1.11-bin.tar.gz；
2. 解压apache-cassandra-2.1.11-bin.tar.gz；

tar -xzvf apache-cassandra-2.1.11-bin.tar.gz

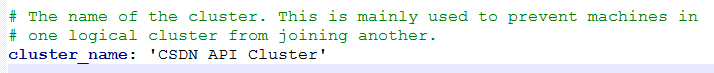
1. 进入解压的目录的conf目录下 ,编辑cassandra.yaml文件；



主要修改的参数：

**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**

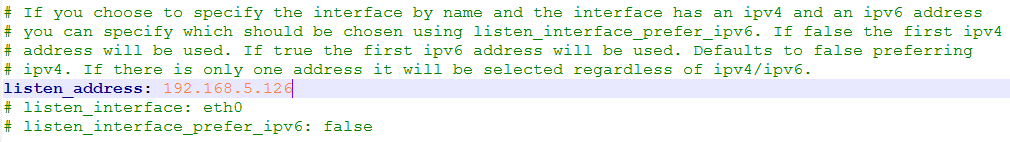
(Default: localhost) The IP address or hostname that Cassandra binds to for connecting to other Cassandra nodes. Set this parameter or listen\_interface, not both. You must change the default setting for multiple nodes to communicate:

• 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.

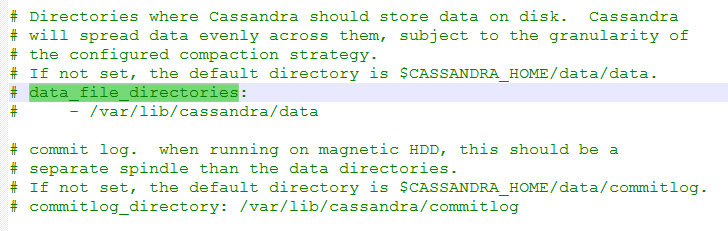


**listen\_interface**

(Default: eth0)note The interface that Cassandra binds to for connecting to other Cassandra nodes. Interfaces must correspond to a single address, IP aliasing is not supported. See listen\_address.

*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**

The directory where the commit log is stored. Default locations:

• Package installations: /var/lib/cassandra/commitlog

• Tarball installations: *install\_location*/data/commitlog

For optimal write performance, place the commit log be on a separate disk partition, or (ideally) a separate

physical device from the data file directories. Because the commit log is append only, an HDD for is acceptable for this purpose.

**data\_file\_directories**

The directory location where table data (SSTables) is stored. Cassandra distributes data evenly across the location, subject to the granularity of the configured compaction strategy. Default locations:

• Package installations: /var/lib/cassandra/data

• Tarball installations: *install\_location*/data/data

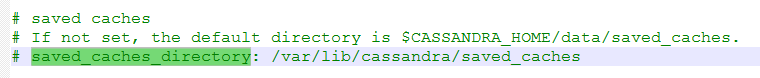
As a production best practice, use RAID 0 and SSDs.

**saved\_caches\_directory**

The directory location where table key and row caches are stored. Default location:

• Package installations: /var/lib/cassandra/saved\_caches

• Tarball installations: *install\_location*/data/saved\_caches



**rpc\_address**

(Default: localhost) The listen address for client connections (Thrift RPC service and native transport).Validvalues are:

• 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.

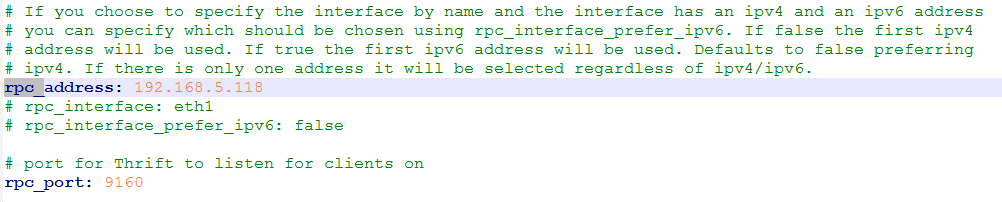
• 0.0.0.0:

Listens on all configured interfaces, but you must set the broadcast\_rpc\_address to a value other than

0.0.0.0.

• IP address

• hostname



**seed\_provider**

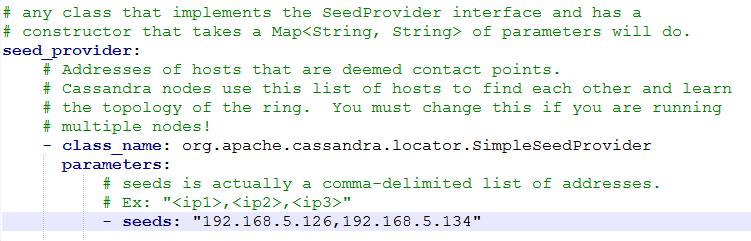
The addresses of hosts deemed contact points. Cassandra nodes use the -seeds list to find each other and learn the topology of the ring.

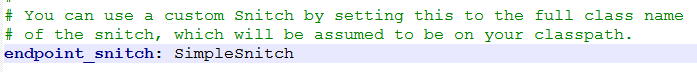
• class\_name (Default: org.apache.cassandra.locator.SimpleSeedProvider)

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 used by gossip for bootstrapping new nodes joining a cluster.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).



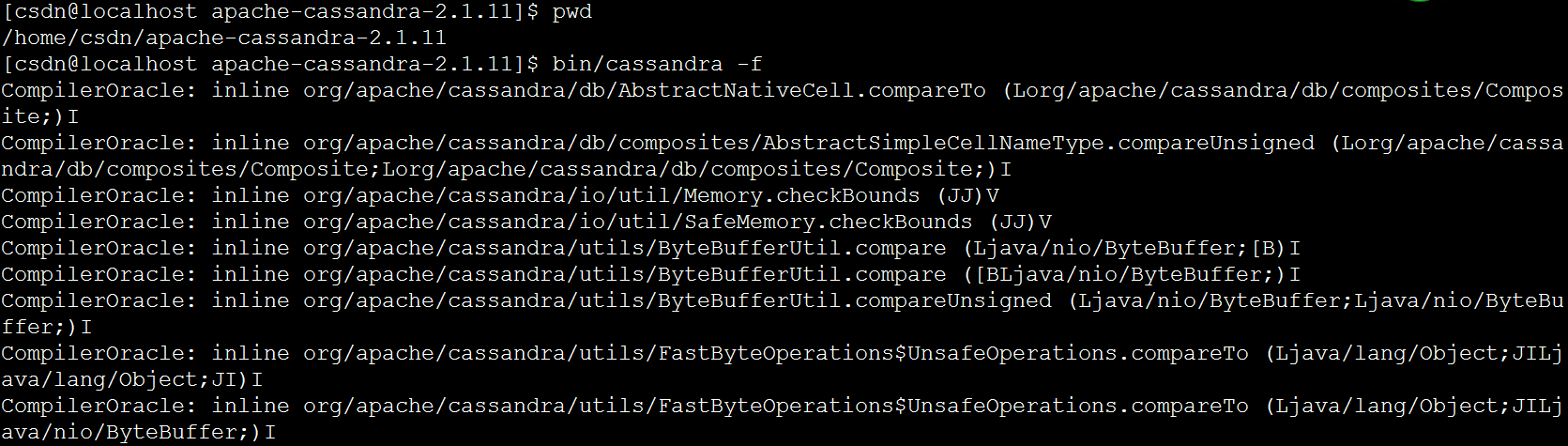


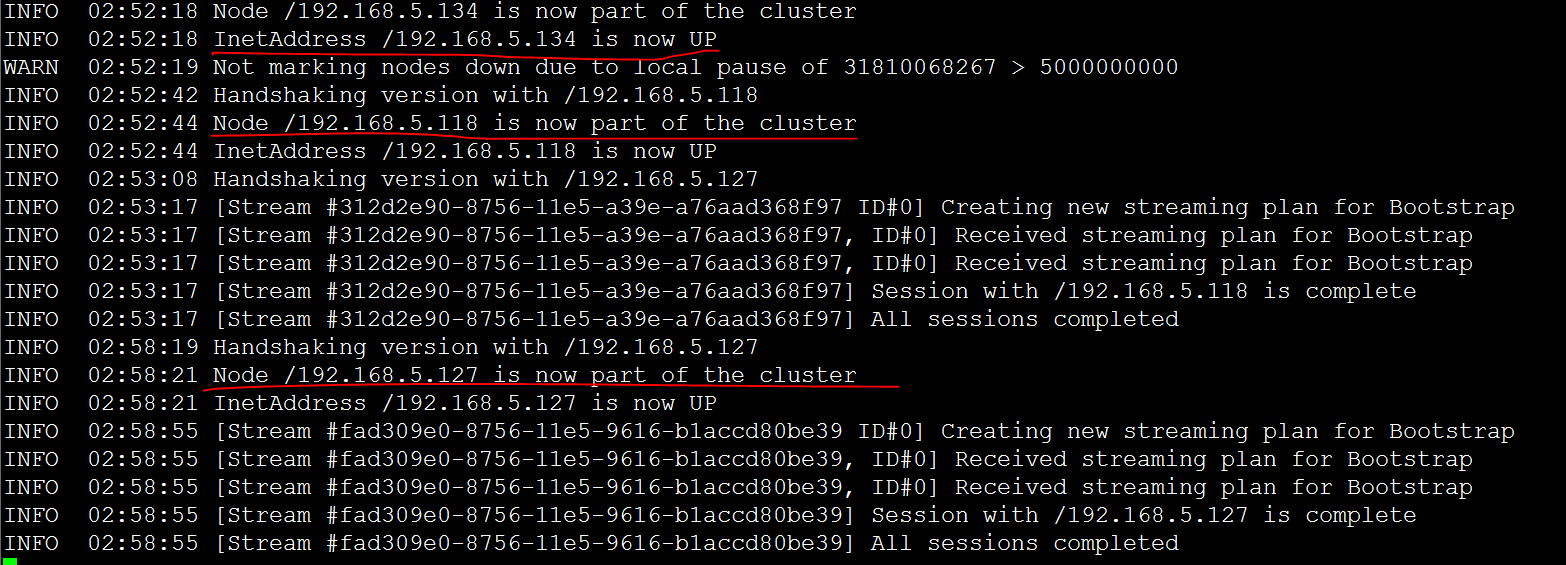
7、将cassandra.yaml文件分发到集群各个节点，然后将各节点上的**listen\_address参数修改为本机的ip地址；**

8、修改cassandra-rackdc.properties文件，可以设置DC（集群） 和RACK（机架）的名字，本此不作修改，采用默认。

9、先启动seed节点，然后再启动集群中的其他节点，启动顺序没什么所谓，只要保证种子节点启动就可以了：$ bin/cassandra -f

参数 -f 的作用是让 Cassandra 以前端程序方式运行，这样有利于调试和观察日志信息，而在实际生产环境中这个参数是不需要的（即 Cassandra 会以 daemon 方式运行）。

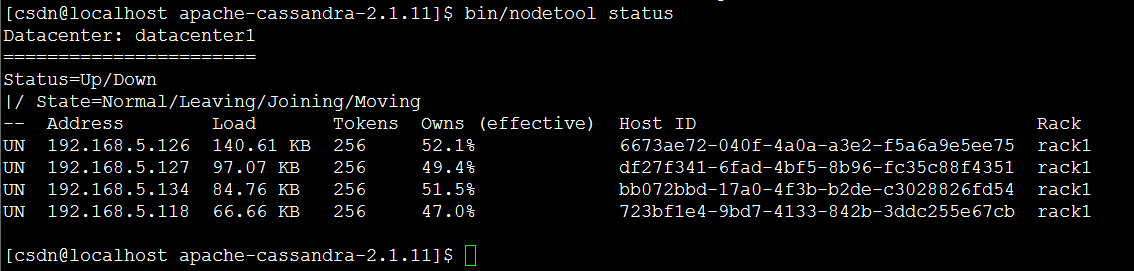




10、集群测试；

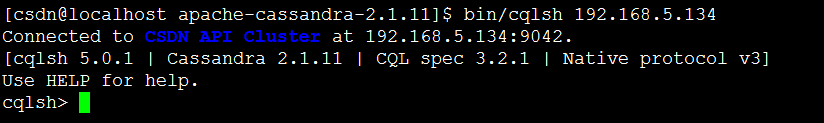
启动完成后，可以使用nodetool工具查看集群各节点状态：

bin/nodetool status



启动cql，不能像单机时直接使用./cqlsh 启动。后面需要加上IP地址，如：

bin/cqlsh 192.168.5.134



首先，创建keyspace；

CREATE KEYSPACE API

WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };

使用keyspace

USE api;

创建用户表

CREATE TABLE users (

user\_id int PRIMARY KEY,

fname text,

lname text

);

插入数据

INSERT INTO users (user\_id, fname, lname) VALUES (1745, 'john', 'smith');

INSERT INTO users (user\_id, fname, lname) VALUES (1744, 'john', 'doe');

INSERT INTO users (user\_id, fname, lname) VALUES (1746, 'john', 'smith');

查看所有数据

SELECT \* FROM users;

结果

user\_id | fname | lname

---------+-------+-------

1745 | john | smith

1744 | john | doe

1746 | john | smith

添加where条件，需要在where列上先创建index

CREATE INDEX ON users (lname);

SELECT \* FROM users WHERE lname = 'smith';

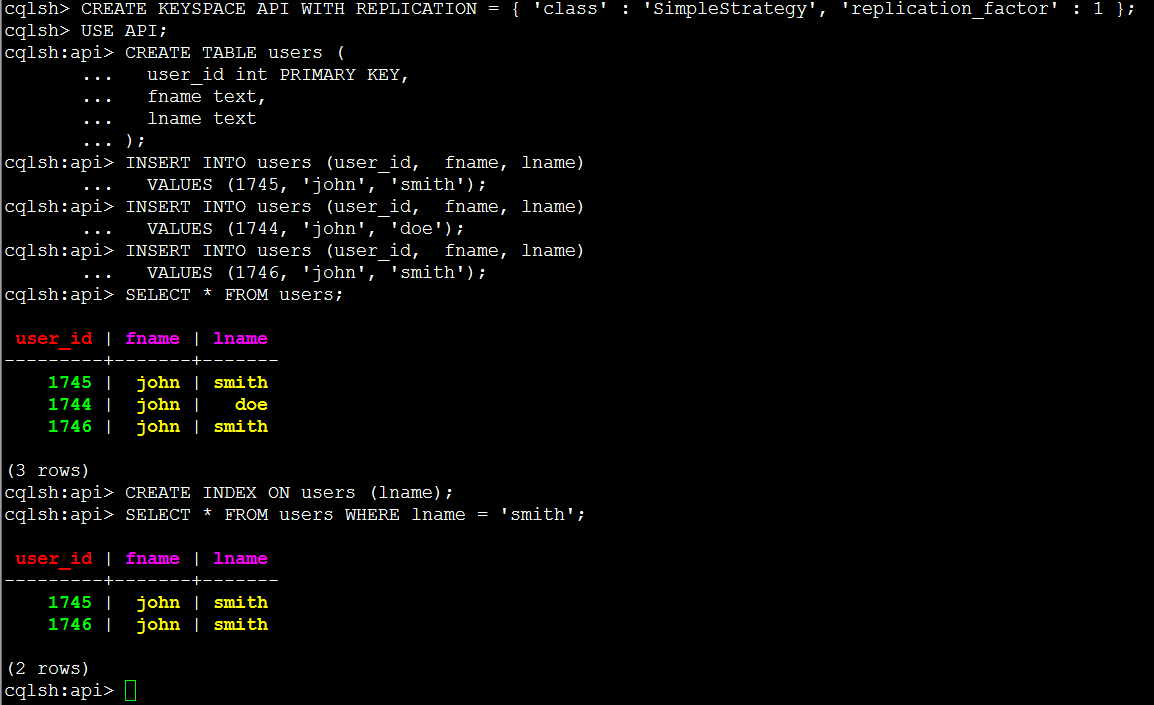
结果

user\_id | fname | lname

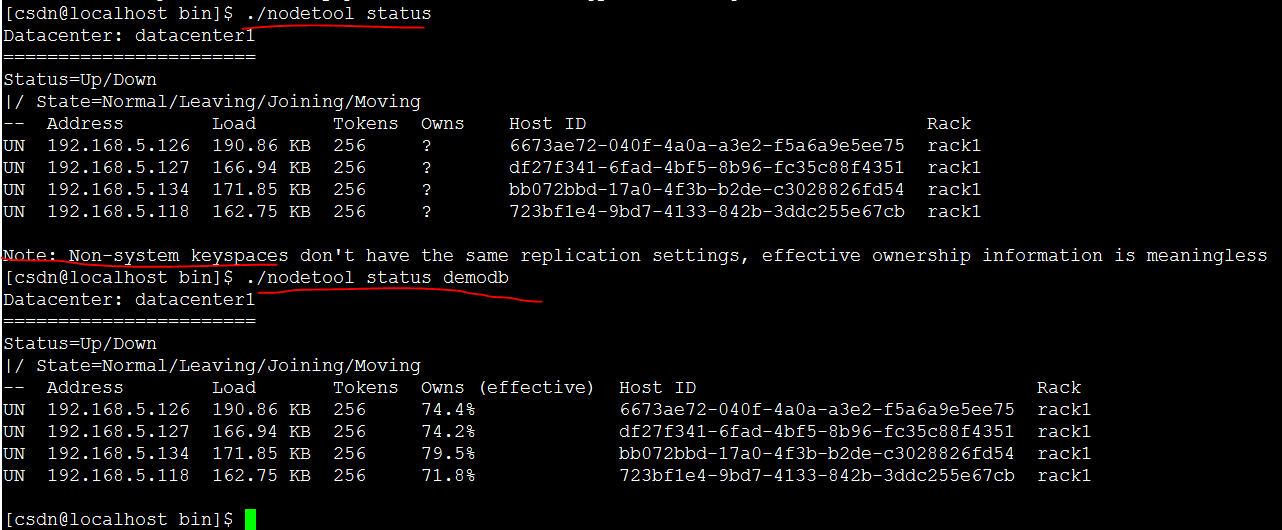
---------+-------+-------

1745 | john | smith

1746 | john | smith



bin/nodetool status api;



# CQL

Cassandra Query Language (CQL) is the default and primary interface into the Cassandra DBMS.

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 columns and rows. The main difference from SQL is that Cassandra does not support joins or subqueries. Instead, Cassandra emphasizes denormalization（前端反规范化） 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.

详细信息 <http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html>

或 cql31.pdf 文件

# Cassandra 系统架构

Cassandra is designed to handle big data workloads across multiple nodes with no single point of failure.( Cassandra是设计用于处理大数据跨多个节点的工作负载,没有单点故障)

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. （Cassandra 采用点对点的分布式系统处理故障问题，同构节点数据分布在集群中的所有节点。每秒钟跨集群每个节点交换信息。）

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.（按顺序写在每个节点上commit log捕捉写活动确保数据持久性。然后索引数据并写入一个内存结构,称为memtable,它像一个回写缓存。一旦内存写满,数据被写入磁盘SSTable数据文件。所有的写操作都被自动分区和复制整个集群。）

Using a process called compaction Cassandra periodically consolidates SSTables, discarding obsolete data and tombstones (an indicator that data was deleted).（使用这一过程被称为压实Cassandra 定期巩固SSTables,放弃过时数据和墓碑(一个指标,数据被删除)。）

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.（Cassandra是一个row-oriented数据库。Cassandra允许任何授权用户连接到数据中心的任何节点使用CQL语言访问数据。使用的方便性,CQL使用类似SQL的语法。从CQL角度数据库包含表。通常,集群中每个应用程序会有一个keyspace。开发人员可以通过cqlsh执行CQL也可以通过驱动程序以及应用程序的语言执行CQL。）

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. （客户端读或写请求可以发送到集群中的任何节点。当客户端请求连接到一个节点,该节点作为特定的客户端操作的coordinator。Coordinator 作为客户端应用程序和节点之间的数据请求的代理。环中的coordinator基于集群配置确定哪些节点应该相应客户端请求。）

## 关键架构

### • 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. （使用独立的数据中心可以防止Cassandra的事务被影响其他工作负载和保持请求不跨数据中心，保证更低的延迟。）

Depending on the replication factor, data can be written to multiple data centers. However, data centers should never span physical locations.（数据可以跨数据中心，数据中心不能跨地理位置）

### • 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（持久化）. 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.（table是row的集合，row是有序columns的集合。）

### • SSTable

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.

## Cassandra 主要配置项

### 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.

**• 去获得集群中其他节点的位置和状态信息**

**• 点对点的通信协议，在这个协议中，节点之间定期交换状态信息**

**• Gossip协议每隔一秒运行一次，节点和不超过的三个节点交换信息**

**• 在信息的交换中，旧的信息会被新的状态信息覆盖**

**• 在同一个cluster集群中，应该使用相同的seed节点**

### 分区器(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.

分区器决定了数据如何在集群内被分发。简单来说，一个分区器就是一个用来计算partition key哈希值的一个哈希函数。每一行数据，由partition key的值唯一标识，并且根据partition key的哈希值决定如何在集群内被分发。包路径org.apache.cassandra.dht

• Murmur3Partitioner：基于MurmurHash哈希算法

• RandomPartitioner：基于MD5哈希算法

• ByteOrderedPartitioner：根据partition key的bytes进行有序分区

### 副本因子(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.

数据复制：在集群的副本总数被称为复制因子，它决定的是一行数据被存储到多少

台机器上面。复制因子为1意味着每一行在一个节点只有一个副本。为2意味着每一

行有两个备份，其中每个副本是在不同的节点。所有的副本都同等重要，没有主副

之分。

### 副本放置策略(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 ecommended 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.

• SimpleStrategy：单数据中心使用，SimpleStrategy根据主键的哈希值，决定

第一个副本应该复制到哪个节点，然后不考虑节点之间的拓扑网络结构，其他的

副本放置在环中顺时针方向的下一个节点。

• NetworkTopologyStrategy：当你的集群使用或者计划使用多数据中心时，请

使用NetworkTopologyStrategy。这种策略指定了每个数据中心有多少副本。

### 告密者(Snitch)

一个snitch（告密者）决定应当从哪个数据数据中心和机架写入和读取数据。

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.

### 配置文件cassandra.yaml

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, andsecurity.

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

• Package installations: /var/lib/cassandra

• Tarball installations: *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.

## 节点间通信 (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. （在第一次启动这是最关键的一个节点。默认情况下,一个节点记得其他节点之间的gossip,后来重启。）

The seed node designation has no purpose other than（没有以外的目的） bootstrapping the gossip process for new nodes joining the cluster.

Seed nodes are *not* a single point of failure, nor（也不是） 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(优化) is not critical（关键）, but it is recommended to use a small seed list

(approximately three nodes per data center).

### 失败检测与恢复

Failure detection is a method for locally determining from gossip state and history if another node in the system is down or has come back up.

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, third-hand, 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 causing node failure. Use the default value for most situations, but increase it to 10 or 12 for Amazon EC2 (due to frequently encountered network congestion). 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.

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.

# 内部构件（Database internals）

Topics about the Cassandra database.

## Storage engine

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(连续) IO to update data is essential（本质的） 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 directories for each table. From the installation directory, data files are stored using this directory and file naming format on default tarball installations:

/data/data/ks1/cf1-5be396077b811e3a3ab9dc4b9ac088d/ks1-cf1-hc-1-Data.db

On packaged installations, the data files are stored in the same format, but in /var/lib/cassandra/data by default. In this example, ks1 represents the keyspace name to distinguish the keyspace for streaming or bulk loading data. A hexadecimal string, 5be396077b811e3a3ab9dc4b9ac088d in this example, is appended to table names to represent unique table IDs.

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

Understanding how Casssandra stores data.

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（一致性） 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（压紧）:

• 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

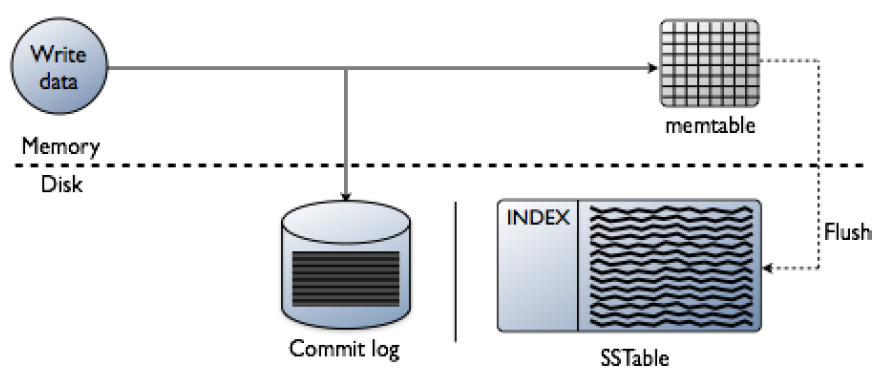
When a write occurs, Cassandra stores the data in a structure in memory, the memtable, and also appends writes to the commit log on disk, providing configurable durability. The commit log receives every write made to a Cassandra node, and these durable writes survive permanently even after power failure. 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（阀值）, 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\_heap\_space\_in\_mb or memtable\_heap\_space\_in\_mb setting 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(不可变的), 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

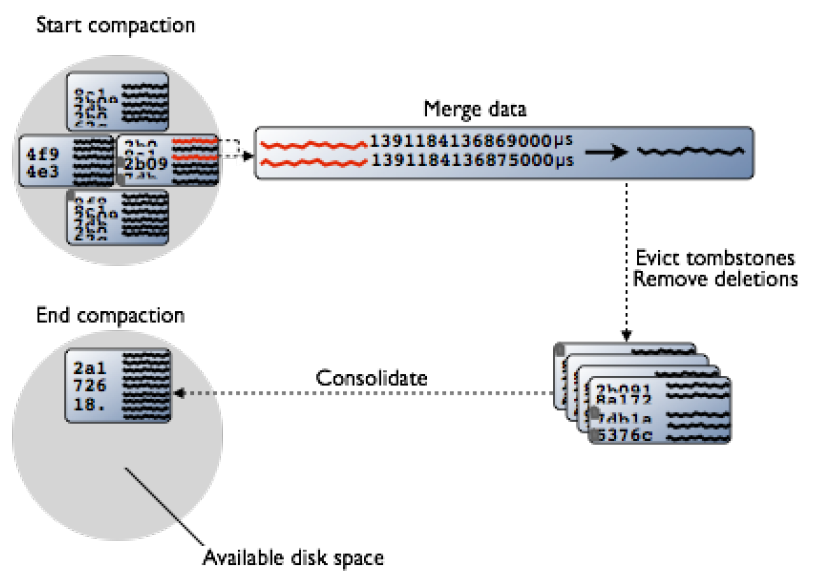
#### Compaction

Periodic(周期的) compaction is essential(本质) 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(积聚) 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(墓碑). 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 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.

Cassandra 2.1 improves read performance after compaction by performing an incremental replacement of compacted SSTables. Instead of waiting for the entire compaction to finish and then throwing away the old SSTable (and cache), Cassandra can read data directly from the new SSTable even before it finishes writing.

As data is written to the new SSTable and reads are directed to it, the corresponding data in the old SSTables is no longer accessed and is evicted from the page cache. Thus begins an incremental process of caching the new SSTable, while directing reads away from the old one. The dramatic cache miss is gone. Cassandra provides predictable high performance even under heavy load.

#### Starting compaction

You can configure these types of compaction to run periodically:

• SizeTieredCompactionStrategy

For write-intensive workloads

• LeveledCompactionStrategy

For read-intensive workloads

• DateTieredCompactionStrategy

For time series data and expiring (TTL) data

You can manually start compaction using the nodetool compact command.

For more information about compaction strategies, see When to Use Leveled Compaction and Leveled

Compaction in Apache Cassandra.

### How Cassandra stores and distributes 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(复合的) partition 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

A brief description 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(净化) 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

How Cassandra deletes data and why deleted data can reappear.

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. A tombstone exists for 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 size-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.

• Deleted data can reappear if you do not run node repair routinely.

**Why deleted data can reappear（**再出现**）**

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 inconsistenciesRepairing nodes on page 140 across all of the replicas when bringing a

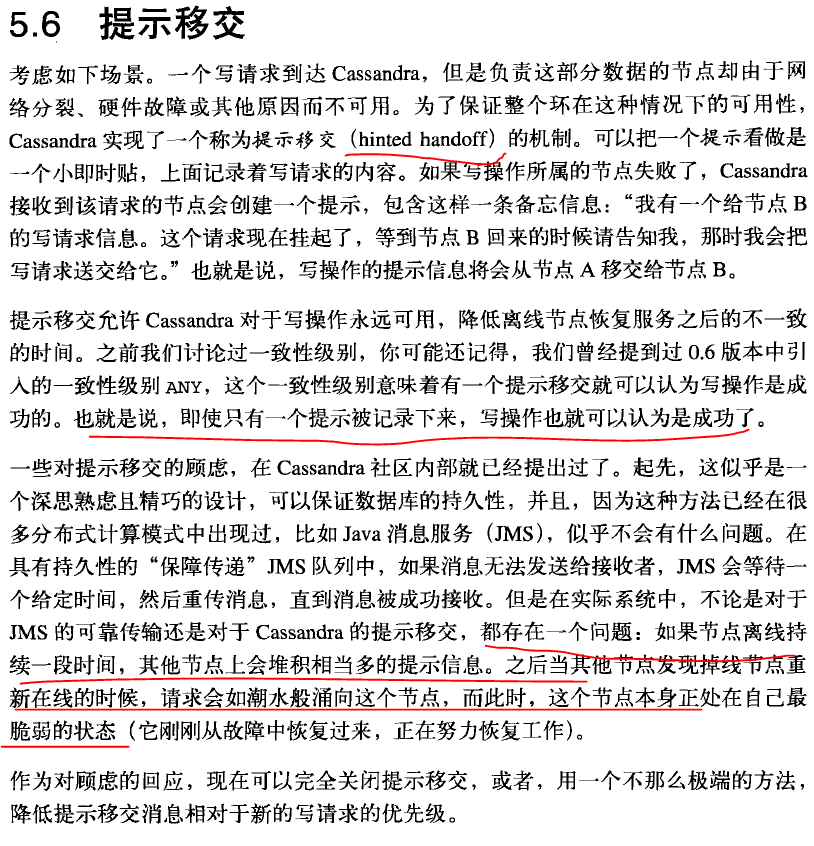
node back into the cluster. If the node doesn't come back within gc\_grace,\_seconds, remove the node, wipe it, and bootstrap it again.

### About hinted handoff writes

How hinted handoff works and how it optimizes the cluster.

Hinted handoff is a Cassandra feature that optimizes（优化） the cluster consistency process and anti-entropy（反熵） 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.



### Reads

#### About reads

How Cassandra combines To satisfy a read, Cassandra must combine results from the active memtable and potentially（可能地） multiple SSTables. 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（排除） the SSTable,（先通过Bloom filter在SSTable中查找） Cassandra checks the partition key cache and takes one of these courses of action:（找不到在到partition key cache中找）

• 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.

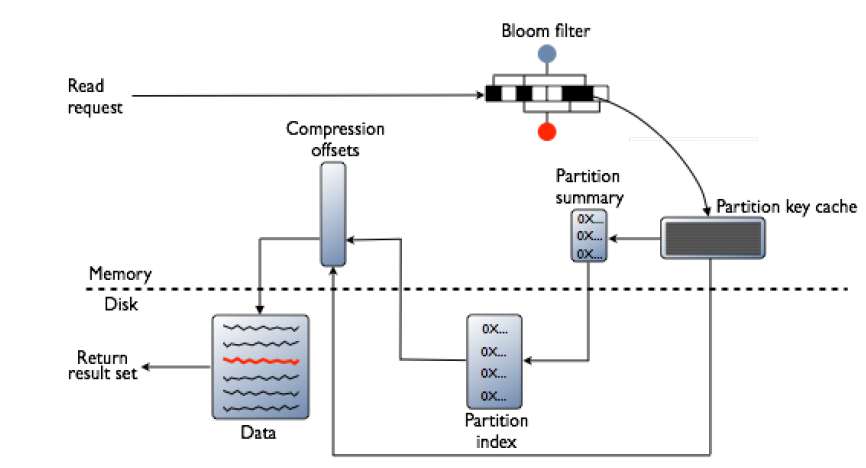
• If an index entry is not found in the cache:

• 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

Bloom Filter是一种空间效率很高的随机数据结构，它利用位数组很简洁地表示一个集合，并能判断一个元素是否属于这个集合。Bloom Filter的这种高效是有一定代价的：在判断一个元素是否属于某个集合时，有可能会把不属于这个集合的元素误认为属于这个集合（false positive）。因此，Bloom Filter不适合那些“零错误”的应用场合。而在能容忍低错误率的应用场合下，Bloom Filter通过极少的错误换取了存储空间的极大节省。

The Bloom filter grows to approximately 1-2 GB per billion（十亿） 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(可调谐的) 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

A brief description about 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（查阅） the partition-level index.

#### How write patterns affect reads

A brief description about how write patterns affect reads.

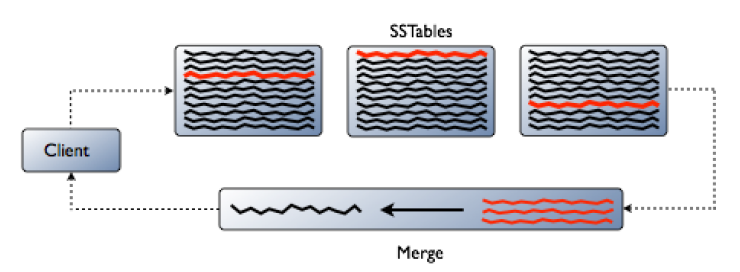
The type of compaction strategy Cassandra performs on your data is configurable and can significantly affect read performance. Using the SizeTieredCompactionStrategy or DateTieredCompactionStrategy

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.



### 事务和并发控制（About transactions and concurrency control）

A brief description 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 （最总/可调节 的一致性）that lets the user decide how strong or eventual they want each transaction’s consistency to be.

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

#### 原子性 Atomicity

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

In Cassandra, a write is atomic at the partition-level, meaning inserting or updating columns in a row is treated as one write operation. A delete operation is also performed atomically. By default, all operations in a batch are performed atomically. Cassandra uses a batch log to ensure all operations in a batch are applied atomically. There is a performance penalty for batch atomicity when a batch spans multiple partitions. If you do not want to incur this penalty, use the UNLOGGED option. Using UNLOGGED makes the batch operation atomic only within a single partition.

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. Depending on the version of the Native CQL Protocol, the timestamp is provided by either the client application or the server. 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 seen by readers.

**Important:** Native CQL Protocol V3 supports client-side timestamps. Be sure to check your client's documentation to ensure that it generates client-side timestamps and that this feature is activated.

#### 一致性 Consistency

A transaction cannot leave the database in an inconsistent state. Cassandra offers different types of consistency.

Cassandra offers two consistency types:

• Tunable（可调谐的） 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（线性化） consistency

In ACID terms, linearizable consistency is a serial (immediate) isolation level for lightweight

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.

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.

#### 隔离性 Isolation

Transactions cannot interfere with each other.

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. Delete operations are performed in isolation. All updates in a batch operation belonging to a given partition key are performed in isolation.

#### 持久性 Durability

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

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.

#### Lightweight transactions

A description about lightweight transactions and when to use them.

Lightweight transactions with linearizable consistency ensure transaction isolation level similar to the serializable level offered by RDBMS’s. They are also known as compare and set transactions. You use lightweight transactions instead of durable transactions with eventual/tunable consistency for situations the require nodes in the distributed system to agree on changes to data.

For example, two users attempting to create a unique user account in the same cluster could overwrite each other’s work. Using a lightweight transaction, the nodes can agree to create only one account.

Cassandra implements lightweight transactions by extending the Paxos consensus protocol, which is based on a quorum-based algorithm. Using this protocol, a distributed system can agree on proposed data additions/modifications without the need for a master database or two-phase commit. You use extensions in CQL for lightweight transactions. You can use an IF clause in a number of CQL statements, such as INSERT, for lightweight transactions.

For example, to ensure that an insert into a new accounts table is unique for a new customer, use the IF

NOT EXISTS clause:

INSERT INTO customer\_account (customerID, customer\_email)

VALUES (‘LauraS’, ‘lauras@gmail.com’)

IF NOT EXISTS;

DML modifications you make using UPDATE can also make use of the IF clause by comparing one or

more columns to various values:

UPDATE customer\_account

SET customer\_email=’laurass@gmail.com’

IF customerID=’LauraS’;

Cassandra 2.1.1 and later support non-equal conditions for lightweight transactions. You can use <, <=, >, >=, != and IN operators in WHERE clauses to query lightweight tables. 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 performs a read repair as part of the commit.

### 数据一致性 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. Tunable consistency means for any given read or write operation, the client application decides how consistent the requested data must be.

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.

The CQL documentation contains a tutorial comparing consistency levels using cqlsh tracing.

Topics about