Paper Link: https://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf

Overview Template:

1. Intro + related work - Tiffany
    a. Defn, what it is, FB inbox search ex?
    b. Data model + API
2. System Arch
    a. Partitioning - William
    b. Replication - Mario
    c. Gossip initialization (failure detection) Hinted Handoff- Ben
        i. Gossip overview syn/ack/ack2
        ii. Failure Detection definition, why it is important, Accrual Failure Detection is what cassandra uses, Accrual Failure Detection definition, phi_convict_threshold
        iii. ←----------> draw this graph hight value means less sensitive to failures, lower value has higher sensitivity for failures
        iv. Hinted Handoff overview, Consistency level, draw the graph, explain scenario
        v.
    d. Local Persistence - Dennis
        i. Local Persistence
            1. JBOB, mixed deployments
        ii. Write operation
        iii. Read operation
    e. Bootstrapping - Miguel

Intro - Tiffany
Apache Cassandra
Released in 2008
Free, open source distributed column wide store database management system

Popular because:

1. Designed to handle large amount of data across commodity servers, specifically known for its high write performance
2. Easy to handle failovers
3. High availability, no single point of failure
4. Asynchronous masterless replication for low latency operations
5. Simple to install + operate (CQL cassandra query language is similar to SQL)

Good Candidates of using cassandra:

1. Write exceeds read by a good margin
2. Data rarely needs to be updated (i.e., transaction logging, tracking, IOT statuses + event history)
3. In summary: for those with large incoming volume of data, very high writes per second, high reliability for data storage, storing metadata of static files
4. I.e. transaction logging, tracking, IOT statuses + event history
5. Currently used by Ebay, Instagram, Hulu, Netflix (entire streaming data persists in cassandra. Over 60+ clusters and 1200 nodes which is over 100 TB of data)

Cons of Cassandra: (Cassandra is designed for specific use cases.. Be careful not to use Cassandra in a situation where another database would be more suitable)

1. Joins and subqueries are not supported in Cassandra. If need to use those operations a lot, should probably consider a different database system
2. No ACID transactions. Logical units of work may need to be split and ordered differently than when using RDBMS.
3. Sort criteria needs to be defined ahead of time
4. Time series data or analytics over data not good.

https://www.ebayinc.com/stories/blogs/tech/cassandra-data-modeling-best-practices-part-1/
Data Model
The Relational Database Model that most everyone knows, and the Cassandra Data Model are quite different. Cassandra is more like a nested, sorted, map data structure than it is relational.
(draw table 1)
Cassandra's column family is a map of a map: the outer map is keyed by a row key, and an inner map keyed by a column key. Both maps are sorted.
(draw table 2, 3)

Cassandra Data Partitioning - William

Cassandra has a two ways of partitioning data. One is Random Partitioning and Ordered partitioning. For ordered partitioning it uses the raw byte array value of the row key is used to decide which nodes store the row. Depending on the distribution of the row keys, you may need to actively manage the tokens assigned to each node to maintain balance. On Random partitioning Cassandra used to use a system of consistent hashing. In the beginning they had a system of load balancing based on the nodes being evenly distributed along the hash function ring. We represent it in a ring because the hash functions revolves around the starting point. In order to mitigate the uneven distribution of data along the nodes. Cassandra used to use a system to analyze the load and reposition the node in the ring to allow for more evenly distributed data since there is no order-preserving hash function that exist. However, they were able to move to another system using virtual nodes.

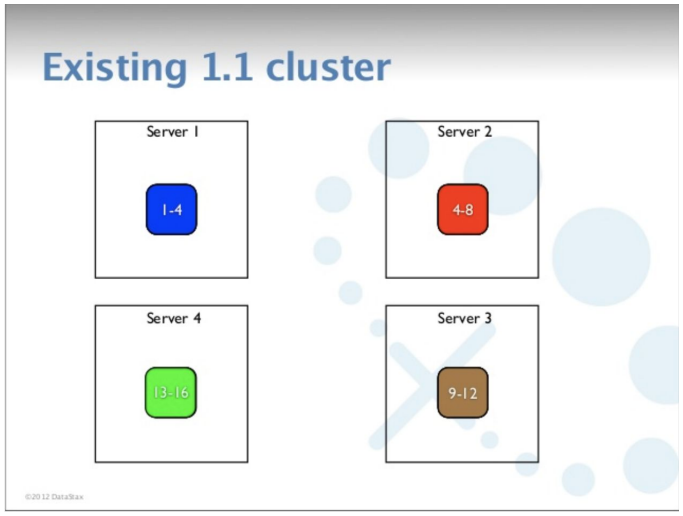
Virtual nodes means every server has many nodes. Each nodes should be small.

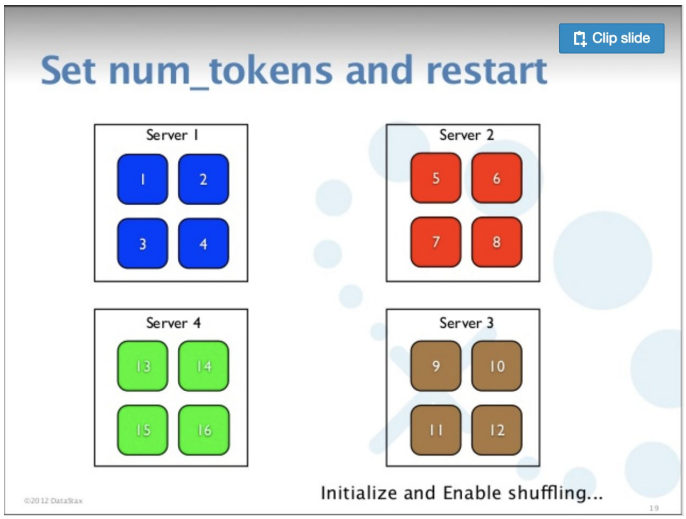
The default configuration are 256 nodes per server. This minimizes the chances of accidental hotspots, and removes the need for reshuffling upon scale.

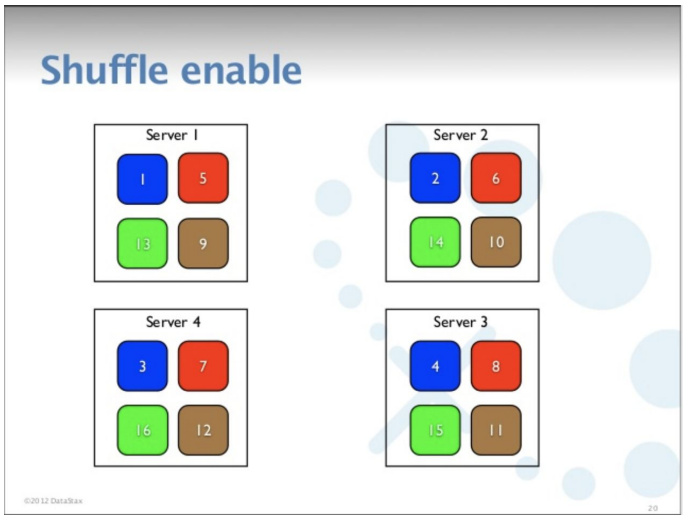
Every time you bring a server into the ring. No more token assignments. It has to do that itself and it evenly distributes them. It also uses a new partitioner called murmur3. Not a cryptographic algorithm. Just a way to create hashes. Faster than md5.

Using this new system it helps with creating a larger number of node space using less number of servers by virtualizing them. token ranges are distributed, so machines bootstrap faster, impact of virtual node failure is spread across entire cluster,

token range assignment automated


Existing 1.1 cluster
Server 1
1-4
Server 2
4-8
Server 4
13-16
Server 3
9-12
©2012 DataStax


Clip slide
Set num_tokens and restart
Server 1
1
2
3
4
Server 2
5
6
7
8
Server 4
13
14
15
16
Server 3
9
10
11
12
Initialize and Enable shuffling...
©2012 DataStax
19


Shuffle enable
Server 1
1
5
13
9
Server 2
2
6
14
10
Server 4
3
7
16
12
Server 3
4
8
15
11
©2012 DataStax
20

Casssandra basic functioning

Row key: identifying information inside a column family. It can be up to 64k size

Can be sorted, either (byte ordered partitioner) or random partitioner MD5 hashing gives a space of 2^128, cassandra uses 2^127 as they use a bit for something. This is similar to the key space of ipv6

Packing more data into a single serverToken assignment is a pain

Replication - Mario
Alright guys, so props to William for that wonderful summary on Partitioning.

Hardware problems can happen at any time and in order to ensure a more reliable and fault tolerant system Cassandra places replicas of data on different nodes. This can be configured by two parameters, the **Replication Factor** and **Replication Strategy**

The **Replication Factor** determines the total number of replicas of the data in the Cassandra ring. So if we had a **Replication Factor of 1** there would only be one copy of of each row, a **Replication Factor of 2** would dictate that there would be two copies of each row in the cluster and so on and so forth.

The **Replication Strategy** is what dictates where the replicas are stored. For this there are two strategies that one can use, the **SimpleStrategy** and **NetworkTopologyStrategy.**

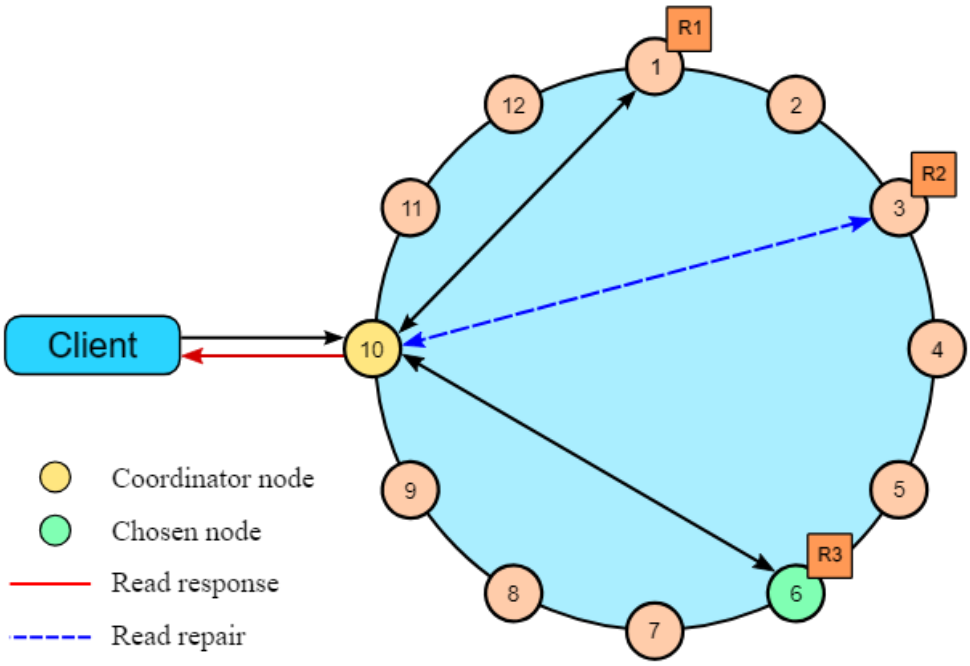
The **SimpleStrategy** method is used for cases where there is say a single rack and a single datacenter. The partitioner will determine the first node where the data will be placed. The replicas will then be placed on the next nodes clockwise in the ring. SimpleStrategy might be good for say development but for a production environment it is advised to go for the **NetworkTopologyStrategy**.

The **NetworkTopologyStrategy** is meant for cases where you have multiple datacenters. Using this strategy walks through the ring until it finds a node that is on a different rack as opposed to the next node that is clockwise. If it fails to find a node in another rack it will place it somewhere on the same rack. The goal is to place replicas on different racks because nodes on the same tend to fail at the same time if say there was a power or network issue, or because of cooling issues. By placing the replicas on different nodes if Rack 1 fails, Rack2 could still be up and running. In the **NetworkTopologyStrategy** each datacenter will have a complete copy of the dataset, i.e if we have a datacenter in the West Coast and a datacenter in the East Coast both West and East Coast will have the complete dataset.

The cool thing about Cassandra is that you can specify a replication factor for each datacenter. So if you wanted to say you had a larger datacenter on the East Coast you could specify a higher **Replication Factor**.

Now you might be wondering how reads and writes happen with all these replicas. This is where the **Consistency Level**. The Consistency Level is the minimum number of nodes that must acknowledge a read or write operation before the operation is considered successful.

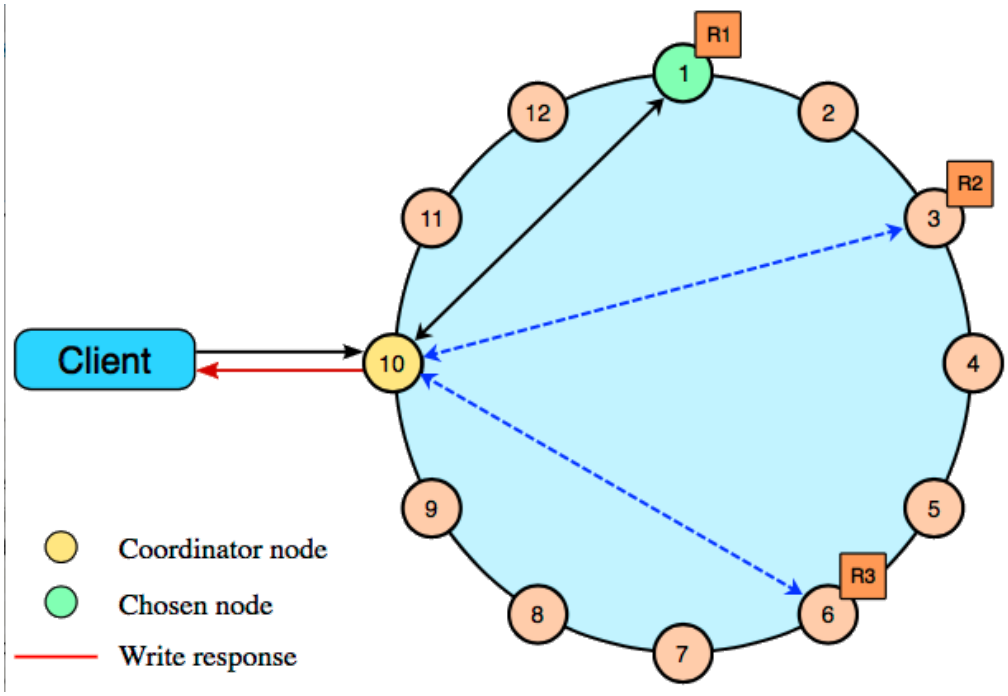
Say we had the following setup



We have a replication factor of 3 which means there are 3 total copies of the data on the nodes, and a consistency level of 2.

The client will send a read request to the closest node, in this case node 10. Node 10 will act as the coordinator node (the coordinator node MAY or MAY NOT have the replica on it). The coordinator node will then send the read request to the nodes that are in the cluster and once 2 of the 3 acknowledge the request the replica with the most recent version will return the requested data. In the background the third replica will be checked for the consistency with the 2 nodes that acknowledged the read request and if needed a read repair will be initiated for the out of date replicas.

For a write request behaves pretty much the same way.



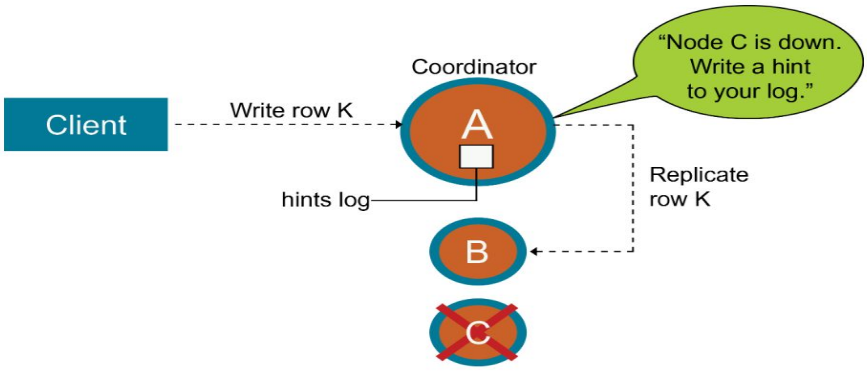
A client will send the write request to the closest node, in this case Node 10, and that will be the coordinator node. Say we have a consistency level of 1. The coordinator node will send the write request to the nodes with the replicas and will dub it a success once one of the nodes acknowledges the write operation. In this scenario node 1 will acknowledge the write and in the background a repair process might be started for the other nodes. Ben

Gossip Intro, failure detection, and Hinted Handoff - Ben

Cassandra users gossip protocol for nodes to internally communicate with other nodes. When a node starts communication with another node it uses a SYN/ACK/ACK2 way of communication, similar to the TCP three way handshake. The SYN is sent from the initiator to the peer which has all nodes (in the form of IP addresses) it knows of as well as other information about them. Then the Peer receives the SYN, does a diff of its local info compared to the SYN's information, and will send an ACK which consists of new data the initiator needs as well as requested data the peer needs from the initiator. The initiator receives this performs a diff and applies changes it needed from the peer, and also sends an ACK2 to the peer node which has information the peer had requested in its ACK.

Failure Detection is a way to determine if a node is up or down in the cluster. This is important in order to avoid communication with down nodes. Cassandra uses Accrual Failure Detection, which represents the likeliness a node is down. How this works is cassandra has a failure detection module which sends heartbeats to nodes, and a missed heartbeat may increase a node's Accrual Failure Detection value. This value depends on other factors such as the current state of the network. Once a node has an Accrual Failure Detection value past a certain threshold, the failure detection module will mark it as down. The user can set this value which is the phi convict threshold, lower values mean there will be more sensitivity to failures and higher values means less sensitivity to failures (may be suitable for a busy environment like AWS).

Hinted Handoff addresses attempts to write to a node that is marked as offline. A node can be offline due to network, hardware, or software memory issues. If a node goes offline, the coordinator node stores a hint for that write for the node. A hint consists of the downed node id, the timestamp, cassandra version, and the data itself. Once a node is detected to be back up, the hints are replayed to it. A property can be configured to stop writing hints once a node has been down for a certain amount of time.


Client
Write row K
Coordinator
A
hints log
"Node C is down.
Write a hint
to your log."
Replicate
row K
B
C

Bootstrapping by Miguel

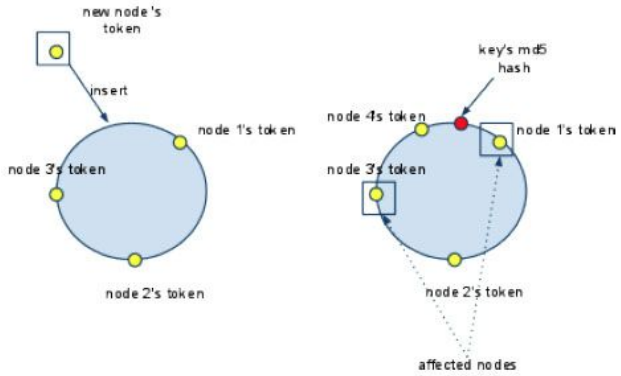
So what happens when a node starts for the first time?
Well this process is called bootstrapping!
Why do we want to add new nodes to the ring? To alleviate heavily loaded nodes.
The bootstrap feature in Apache Cassandra controls the ability for the data in cluster to be automatically redistributed when a new node is inserted. The new node joining the cluster is defined as an empty node without system tables or data.

- If we talk about virtual nodes “Adding new nodes is called “bootstrapping”. The num_tokens parameter will define the amount of virtual nodes (tokens) the joining node will be assigned during bootstrap. The tokens define the sections of the ring (token ranges) the node will become responsible for.”

One of the ways to add a new node is by assigning a random token to this node which gives its position in the ring. It gossips its location to the rest of the ring where the information is exchanged about one another. Another way to add a new node is by reading a config file to contact its initial contact points. The contact points are known as Seeds, which is basically used by newly added node to know about all nodes in the cluster. Seeds can also come from zookeeper which is a centralized service for maintaining configurations.

When the new node is added into the cluster it will calculate the tokens of the different data replicas that is to be responsible for. This process where tokens are calculated and acquired by the new node is often referred to as a range movement/streaming.
This new node will take the range which other node was responsible before. New writes will now be executed in the new node. What about reads? Reads are going to the old node.

A clean up script has to be manually executed to remove data from the old node(s).
How are new nodes actually added? New Nodes are added manually by administrator via CLI or Web interface provided by Cassandra.

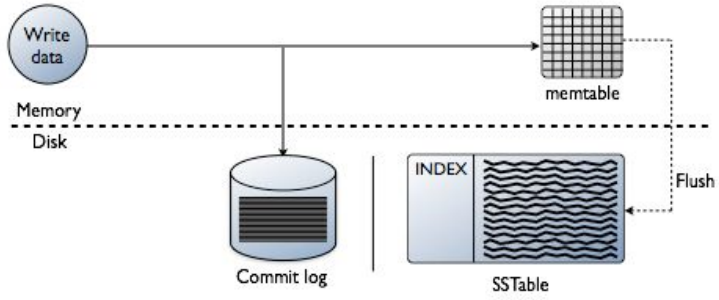


References:
http://cassandra.apache.org/doc/4.0/operating/topo_changes.html
https://docs.datastax.com/en/cassandra/3.0/cassandra/operations/opsAddNodeToCluster.html
http://thelastpickle.com/blog/2017/05/23/auto-bootstrapping-part1.html

Dennis - Local Persistence/Read & Write protocol

For my part i'm going to talk about local persistence in Cassandra. This is aided in a few ways like JBOD support and mixed SSD/HDD deployments. But for this i'm going to focus on how it's done by how data is represented on the disk and relies on the local file system to ensure persistence. To do this i'm going to have to explain how write and read operations work in Cassandra

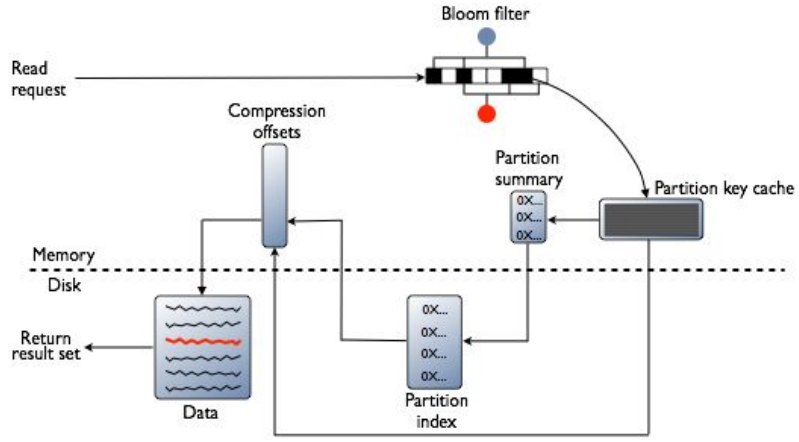
First off the write operation, it looks like so:



And there are a few steps for it which lead to data persistence and durability.

1. First off, when a write operation is induced, the write data is written to the commit log. The commit log is stored on the disk so if the node has any failures the data is durable and can be restored on restart. After writing to the commit log(unless configured otherwise) is confirmed, the write to memtable can begin.
2. The memtable is a writeback cache of data rows that can be looked up by key. It is stored on the Memory but if the data is lost it can be recovered by replaying the commit log to restore the memtables contents. The memtable has a configurable limit, and once the limit is reached from writing to it Cassandra automatically initiates a Flush of the memtable.
3. The flush is done by writing the data to the disk in memtable sorted order and is written out sequentially which leads to efficient and fast writes. It is stored in the disk in an SSTable. After a memtable is flushed the corresponding data in the commit log is also purged. This means that if the commit log is ever not empty upon startup there is a mutation and the data previously into the memtable wasnt flushed prior to node failure, so the commit log is written to memtable again .After being written to an SSTable cannot be written to again and is immutable, so there is one SSTable per memtable, which means data partitions are often stored across different SSTables.
4. Over time many SSTable files can exist that must be consulted for read operations and unused space can accrue, Cassandra has built in compaction methods in order to merge multiple old SSTables into a new singular one to remedy these issues.

Next the read operation, consult the read request flow diagram:



1. First off, the memtable is checked to see if corresponding read request data is still in the memtable(not flushed) and if it is then it is read.
2. If Row caching is enabled then this is checked next. Row-cache stores the most frequently accessed rows(configurable amount). It stores subsets of partition data inside memory for quick access. If not found here its sent to the bloom filter.
3. Next the read request information is sent to a bloom filter, there is a bloom filter for each SSTable and a bloom filter can check to see if the read request data is not present in the respective SSTable or if it is most likely present(it can't check if it is 100% there, so sometimes false positives). The bloom filter prevents from needing to check each and every SSTable for the read data. If it is deemed to probably be in the respective SSTable it is sent to the partition key cache.
4. The partition key cache stores a cache of a portion keys for an SSTable in off-heap memory. If the key is found here then it goes directly to the compression offset map to find the compressed block data on disk that has the data which can return the result set.
5. If it is not found on the partition key cache its sent to the SSTables partition summary, which stores a sample of each tables indexes(every X key). It stores the ranges for every X keys in partition index of an SSTable and shortens the amount of data that must be searched to find the read request data by giving a search range of requested partition key to the partition index.
6. The partition index stores all of the SSTables partition keys to find the location of a desired portion. After the data partition is found here, it is sent to the compression offset map to find compressed block on the disk that has the data and from there the result data set is returned.

References:

https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlAboutReads.html
https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlHowDataWritten.html
https://docs.datastax.com/en/articles/cassandra/cassandrathenandnow.html