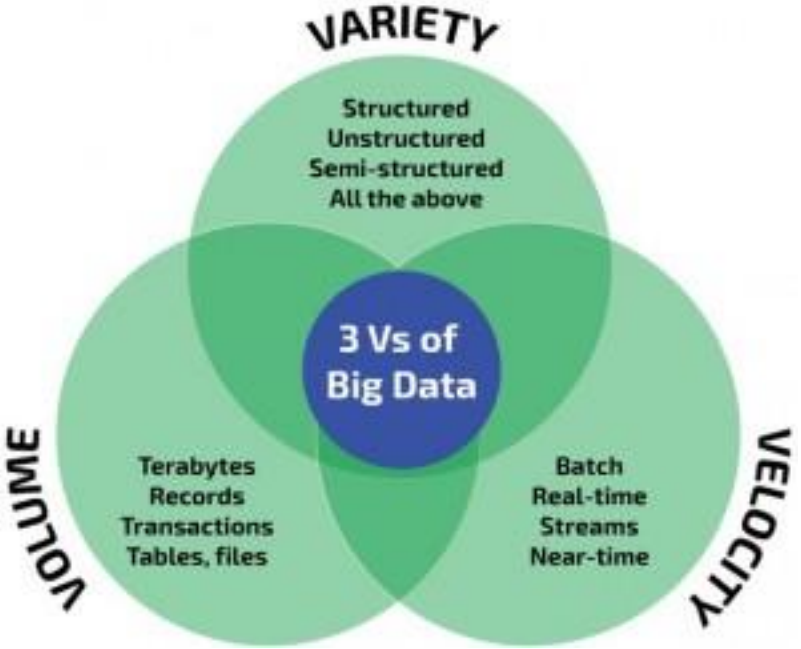
Cassandra Overview

29/11/2017

SQL v/s No SQL Databases

SQL	NOSQL
Relational	Non Relational
Structured data type	Non structured data, such as Photos, Videos, Articles, etc
Supports powerful query language. Fixed schema. Supports transactions.	Supports simple query language. No fixed schema.Does not support transactions.
MySQL, Oracle, IMB DB2, Sybase, My SQL server, Microsoft Azure, Maria DB, Postgre SQL	Mongo DB, Apache's Couch DB, Hbase, Oracle nosql,Apache's Cassandra DBS

Volume , Variety , Velocity



Types of No sql databases

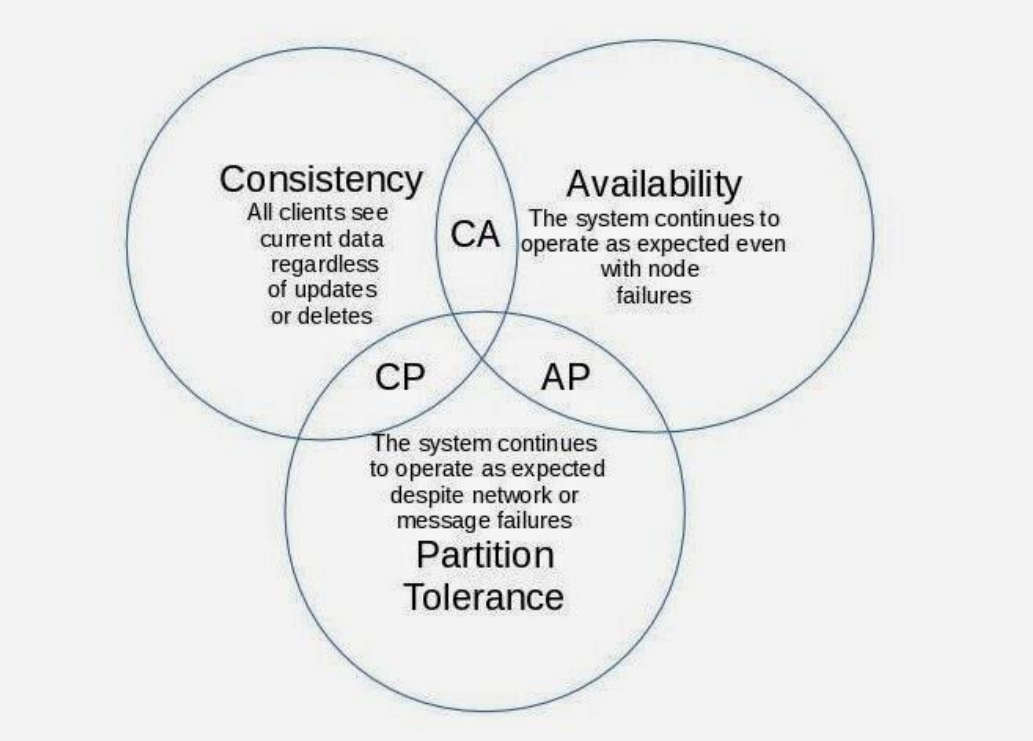
1. Key value Stores : Cassandra, DynamoDB etc
2. Column Stores : HBase, BigTable and Hypertable
3. Document Stores: MongoDB, Couchbase
4. Graph database : Polyglot

History of Cassandra

Features of Cassandra

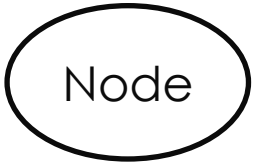
1. **Distributed:** Capable of running on multiple machines while appearing to users as a unified whole.
2. **Decentralized :** Cassandra has no single point of failure. All of the nodes in a Cassandra cluster function exactly the same.
3. **Elastic scalability :**the ability to add nodes without going down produces predictable increases in performance.
4. **High availability and fault tolerance:** It is easy to remove a node from the cluster without affecting the cluster.
5. **Flexible data storage :**Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured.
6. **Tuneable Consistency**
7. **Open source**

Brewer's CAP theorem



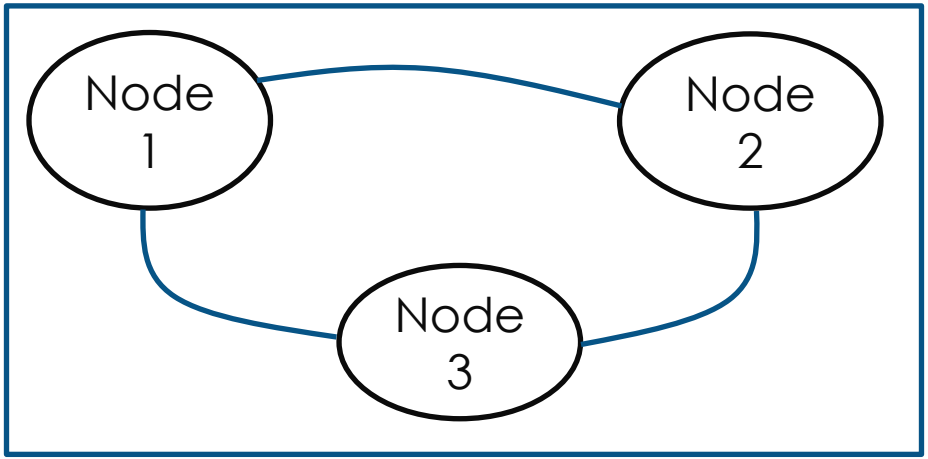
Data model of Cassandra

- Node : It is the place where data is stored.

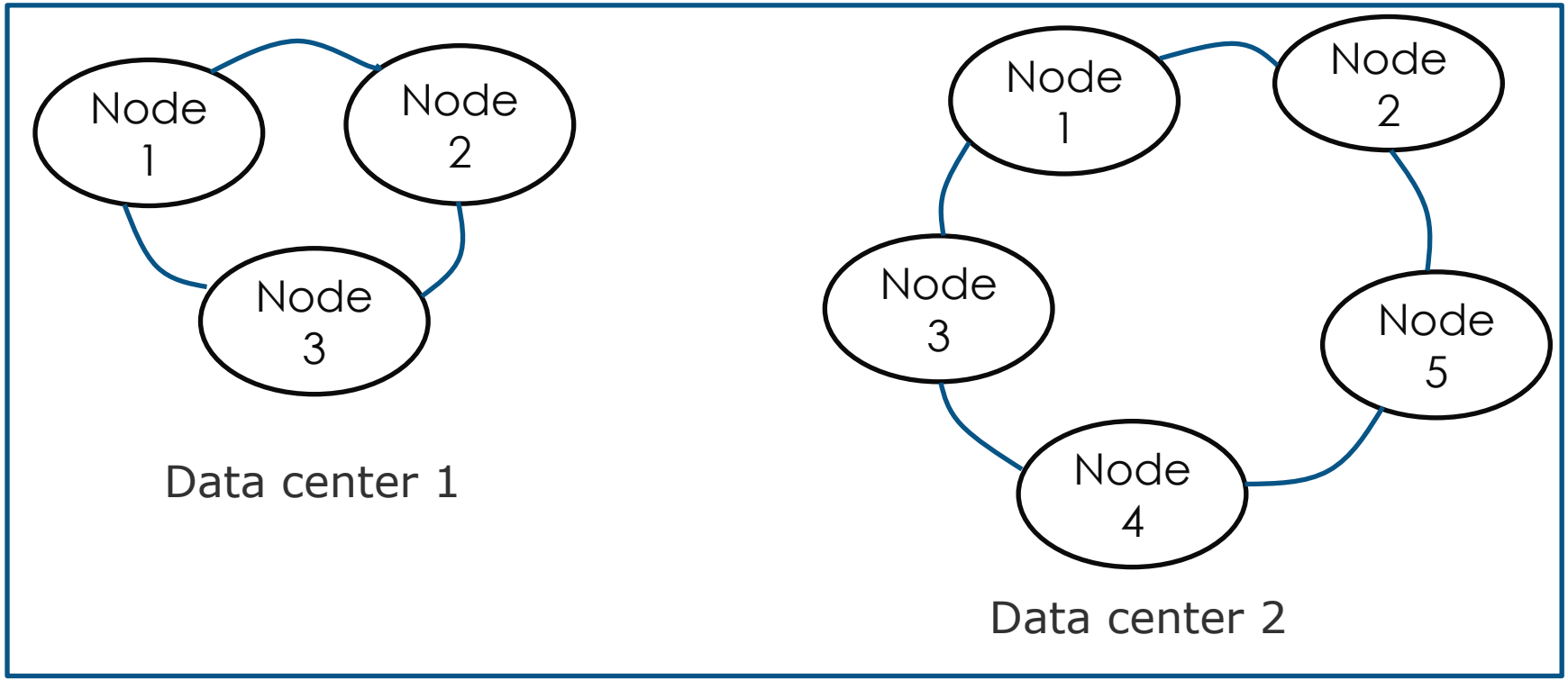


3000-5000 transactions per core

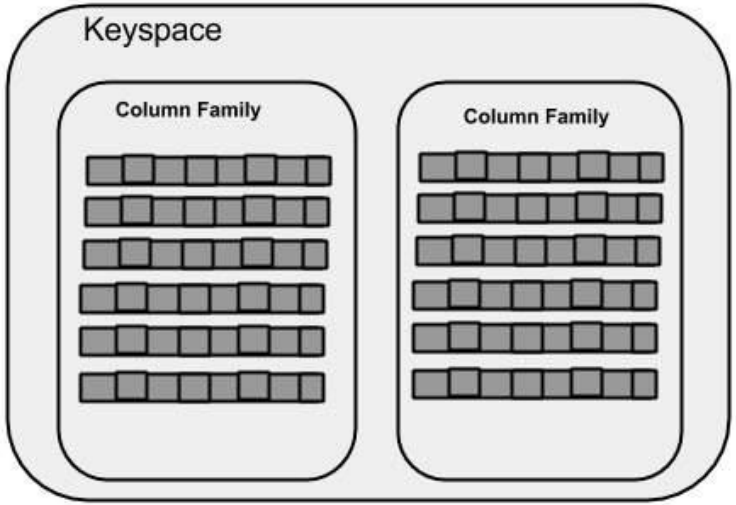
- Data center: A collection of related nodes.



- **Clusters** : Collection of one or more data centers.



- **Keyspaces** : It is the outermost container for data in Cassandra.



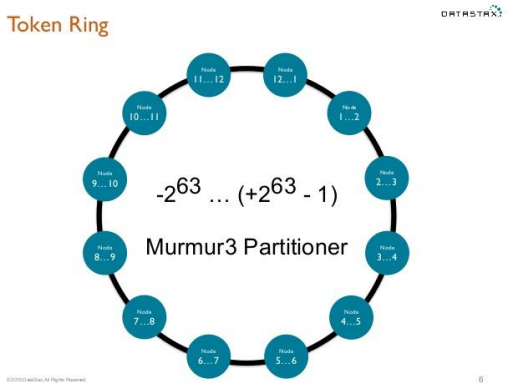
- **Column family :** A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns.

Cassandra Architecture

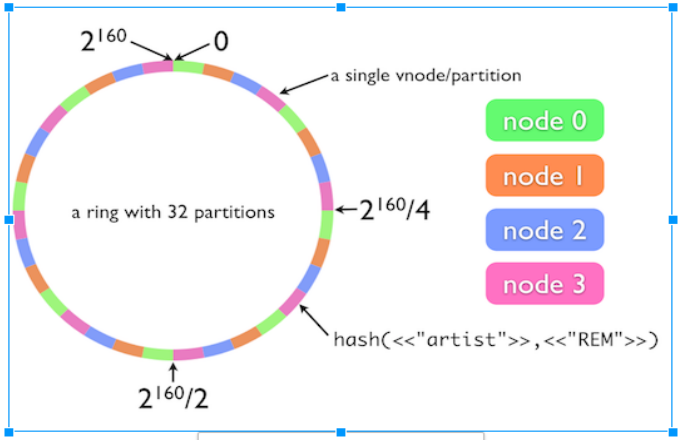
1. **Partitions :** Grouping of data.

   Partitioners: Determines how data is didtributed across the nodes in the cluster.

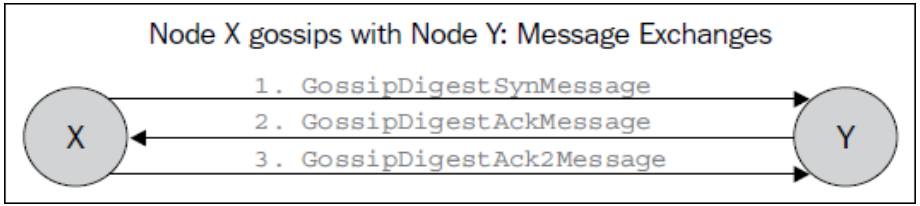
2. **Rings and tokens :**



3. Virtual Nodes : These nodes change the paradigm of one token or range per node to many tokens or range per node.



4. **Gossip protocol :** Cassandra uses gossip protocol that allows each node to keep track of state information about the other nodes.



- 5. **Snitches** : A snitch determines which datacenters and racks nodes belong to.

➢ Regular snitch :

- SimpleSnitch
  The SimpleSnitch is used only for single-datacenter deployments.
- PropertyFileSnitch
  Determines the location of nodes by rack and datacenter.
- GossipingPropertyFileSnitch
  Automatically updates all nodes using gossip when adding new nodes and is recommended for production.

6. **Replication** : Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance. A replication strategy determines the nodes where replicas are placed.

- Two replication strategies are available:
- SimpleStrategy : Used for single data center and one rack.
- NetworkTopologyStrategy : Used for multiple data centers.

7. **Consistency levels**: Cassandra provides tuneable consistency.

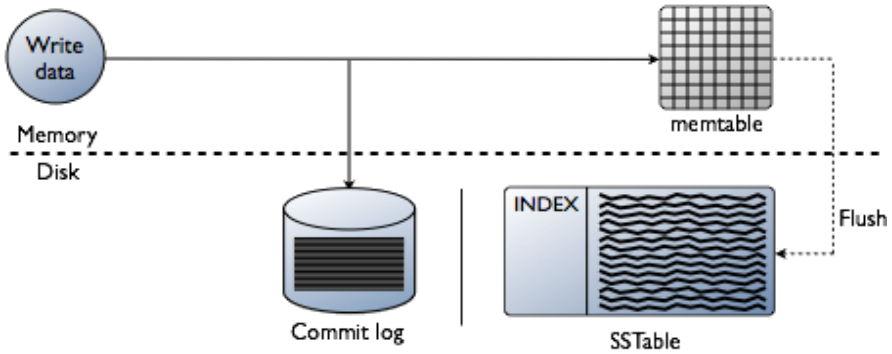
- A higher consistency means more nodes needs to respond.
- A read consistency specifies how many nodes should respond to a read before returning the data.
- A write consistency level specifies how many nodes should respond for a write to be successful.

- ALL
- ANY
- ONE,TWO,THREE
- QUORUM
- EACH_QUORUM
- LOCAL_QUORUM
- LOCAL_ONE

Commit logs : Commitlogs are an append only log of all mutations local to a Cassandra node. Any data written to Cassandra will first be written to a commit log before being written to a memtable. This provides durability in the case of unexpected shutdown.

Memtables: Memtables are in-memory structures where Cassandra buffers writes. In general, there is one active memtable per table. Eventually, memtables are flushed onto disk and become immutable SSTables.

SSTables: SSTables are the immutable data files that Cassandra uses for persisting data on disk.



Write Path

1. Client Writes to any Cassandra Node
2. Coordinator Node replicates to nodes and Zones
3. Nodes return ack to coordinator
4. Coordinator returns ack to client
5. Data written to internal commit log disk

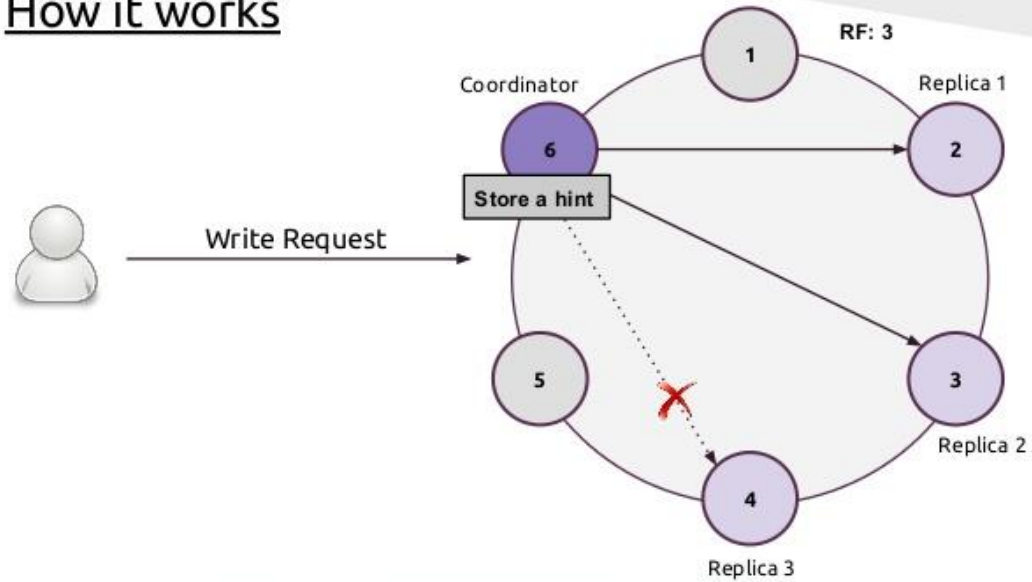
If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

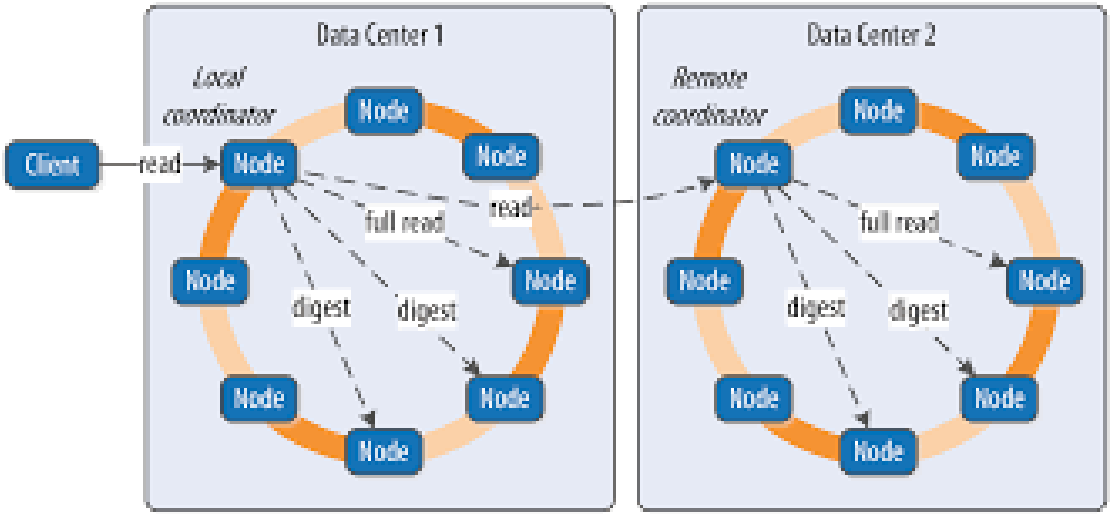
SSTable disk writes and compactions occur asynchronously

Hinted Handoff

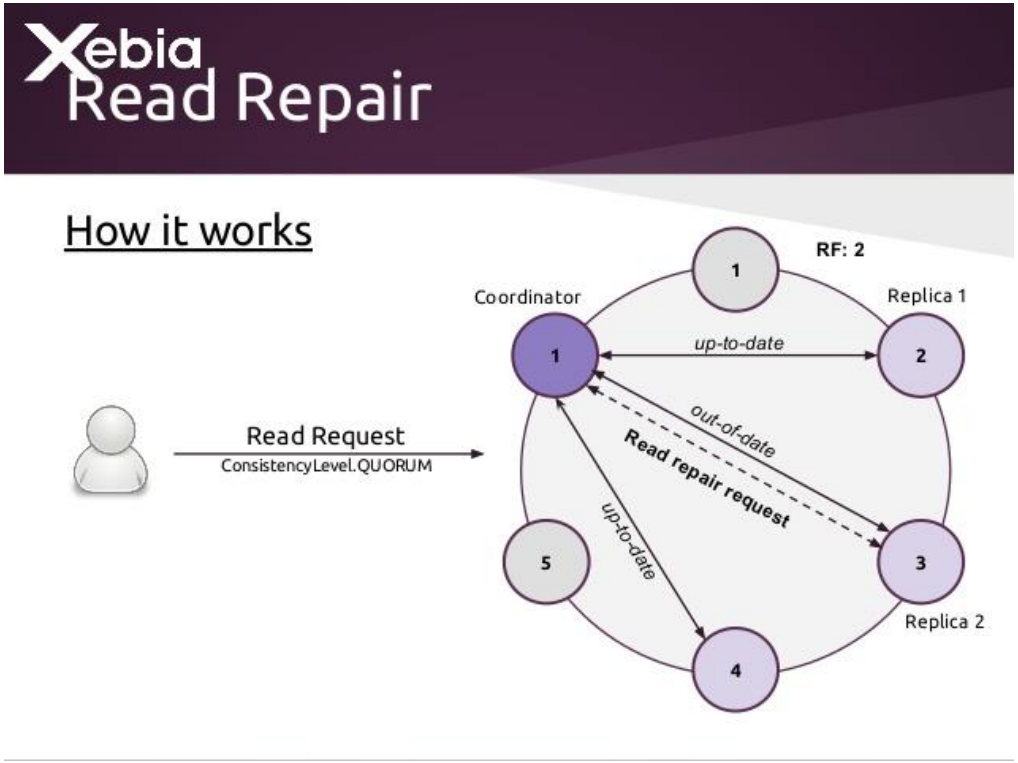
How it works



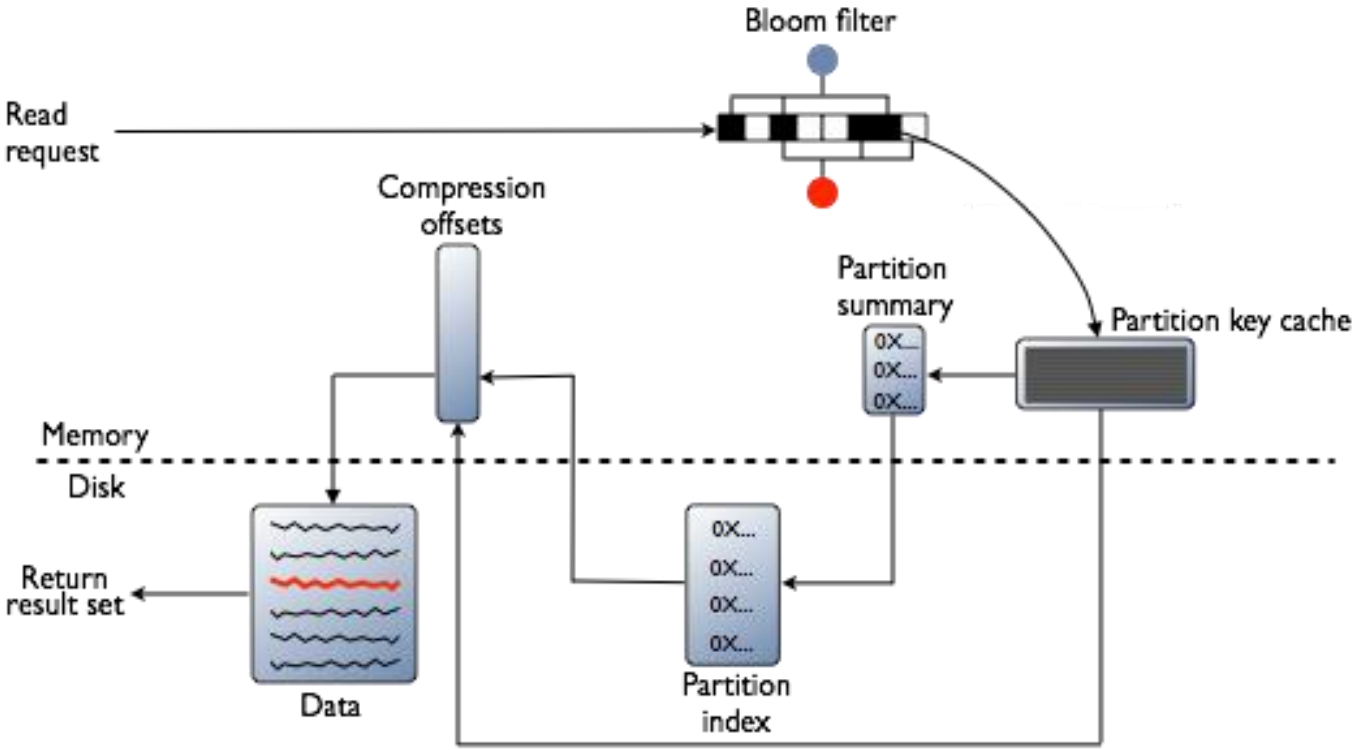
Read Path



Read Repair

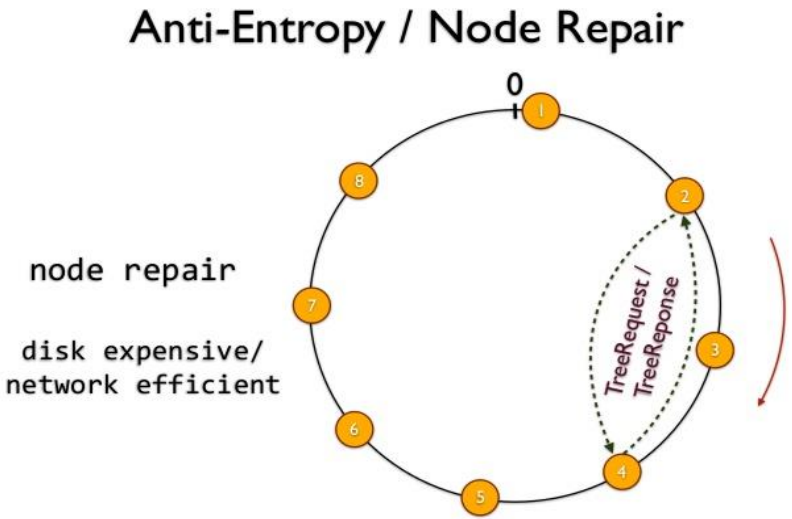


Read path in SSTable



Repairs

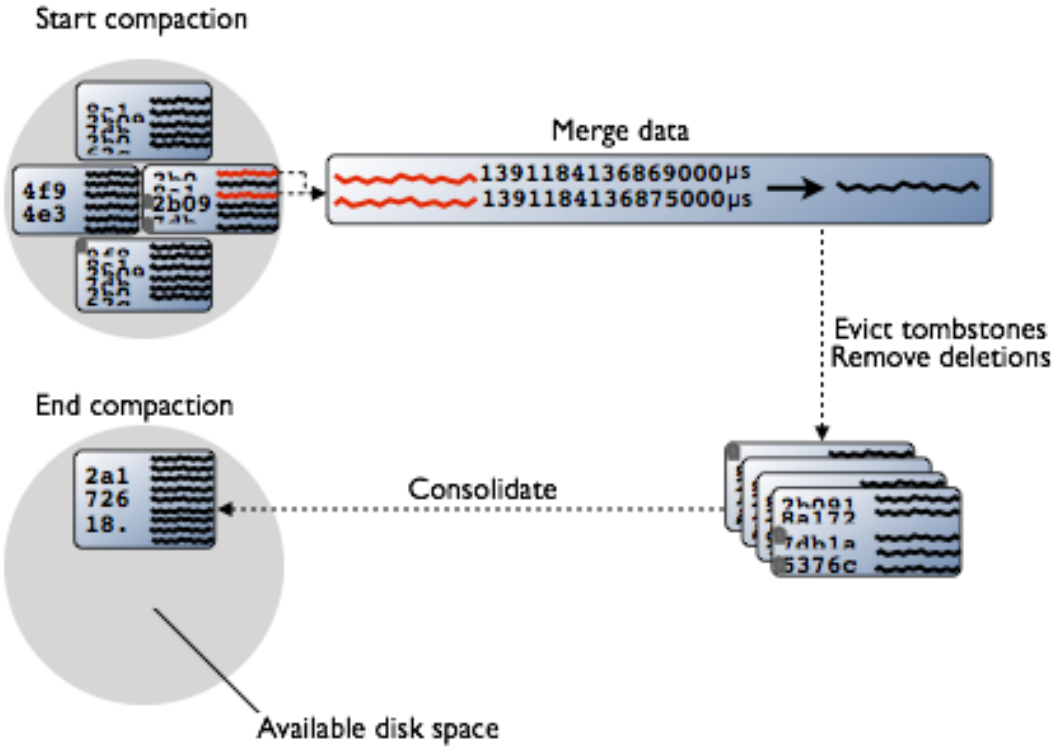
- Why repairs?
- Anti-Entropy Repairs- Manual repairs



Type of repairs

- Full repair
- Incremental repair
- Sequential and parallel repairs
- Primary range repairs
- Subrange repairs

Compaction



Types of compactions

- Minor compactions:
- Major compactions :

Compaction Stratergies

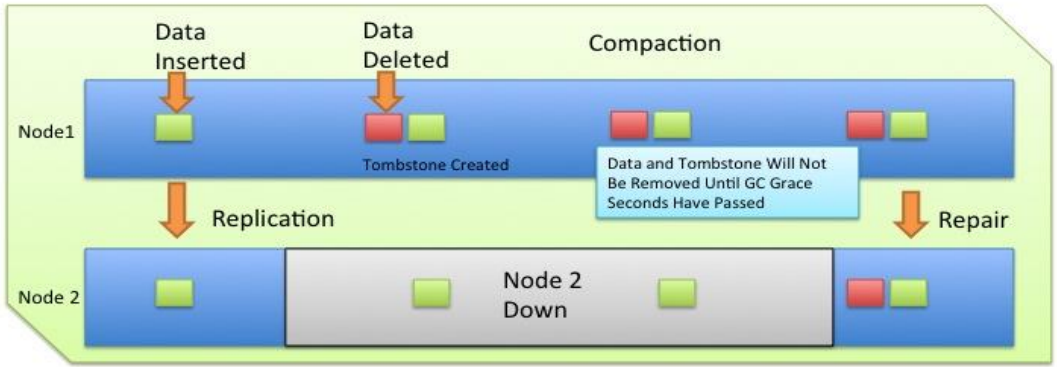
- SizeTieredComapctionStratergy(STCS):
- LevelCompactionStratergy(LCS):
- DateTiered Compaction Strategy (DTCS):

UDC (User Defined Compactions)

- Creates a compaction task for one or more tables explicitly. This task is then handed off to the Cassandra runtime to be executed like any other compaction.

Tombstone

With GC Grace Seconds a Temporary Outage
Will Not Effect Deletes



Data deleted will not be removed until after GC Grace Seconds. This prevents an outage of Node 2 of less than GC Grace seconds from modifying the database.

Monitoring

KPI

- Down nodes
- Throughput, especially read and write requests.
- Latency, especially read and write latency.
- Disk usage, especially disk space on each node.
- Garbage collection frequency and duration.
- Errors and overruns, especially unavailable exceptions which indicate failed requests due to unavailability of nodes in the cluster.

Subrange Repair Script WRT BA & BC

BC :

CSC1 :

META : mtlmet{01..09} tormet{01..09} - uses “imail2” user. One META instance per server.
BLOB : mtlcsh{01..20} torcsh{01..20} - Uses "imail1", and "imail3" users.

CSC2 :

META : mtlcsh{23..45} torcsh{23..45} - Uses "imail5" user. One META instance per server.

BLOB : mtlcsh{23..45} torcsh{23..45} - Uses "imail1", "imail2", "imail3", and "imail4" users. Four BLOB instances per server.

- On BC, the subrangerepair script is on all the nodes. It just runs for Meta so imail2 user on CSC1 and imail5 user on CSC2 (only for MONTREAL nodes).
- They usually complete in less than 12 hours so we don't need to worry about overlapping as long as we have them run on different days.

Running only for Meta

- CSC1 - for imail2 user
- mtlmet{01..09}

  mtlmet01 - 0 23 * * 0 cd $HOME/cron && $HOME/bin/run_subrangerepair

  mtlmet02 - 0 23 * * 1 cd $HOME/cron && $HOME/bin/run_subrangerepair

  mtlmet 09 - 0 23 * * 0 cd $HOME/cron && $HOME/bin/run_subrangerepair

- tormet{01..09}

  tormet01 - 0 23 * * 3 cd $HOME/cron && $HOME/bin/run_subrangerepair

  tormet02 - 0 23 * * 4 cd $HOME/cron && $HOME/bin/run_subrangerepair

- CSC2 -
- mtlcsh{23..45}
- mtlcsh23 - 0 21 * * 1 cd $HOME/cron && $HOME/scripts/run_subrangerepair
- mtlcsh24 - 0 21 * * 2 cd $HOME/cron && $HOME/scripts/run_subrangerepair
- mtlcsh25 - 0 21 * * 3 cd $HOME/cron && $HOME/scripts/run_subrangerepair

- torcsh{23..45}
- Not Running -

BA :

- META :
- TR1 : tr1cass{01..08} - Uses "imail" user.
- TR2 : tr2cass{01..08} - Uses "imail" user.

- BLOB :
- TR1 : tr1cass{01..08} - Uses "imail1" and "imail2" users. Two BLOB instances per server.
- TR2 : tr2cass{01..08} - Uses "imail1" and "imail2" users. Two BLOB instances per server.r.

- for BA, Each nodetool repair takes about 48-56 hours so we need them to run after another and dont want any repair to overlap.
- Script run via Root user for imail1 and imail2.

Frequent issues :

- High Load
- Latency, especially read and write latency.
- Down nodes
- Disk usage, especially disk space on each node.