Foundations of CASSANDRA NoSQL Database

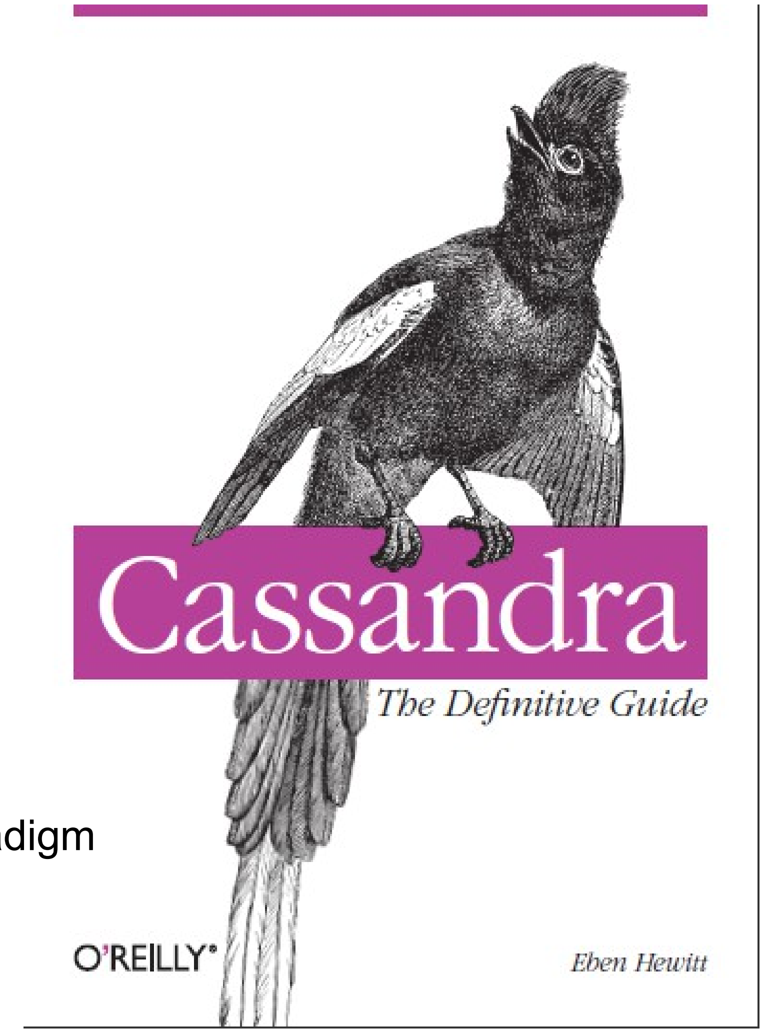
Janardhanan PS

08 Aug 2017

Agenda

Introduction

- NoSQL Databases
- Cassandra
    - Massively scalable
    - Partitioned row store
    - Master-less architecture
    - Linear scale performance
    - No single points of failure
    - Replication
    - Peer-to-peer
    - Written in Java
    - Supports MapReduce Programming paradigm
    - Tunable consistency
    - Clients - CQL and/or Thrift,



Scalability

- Vertical Scaling
  - More powerful CPU
  - More memory
  - Free performance lunch
- Horizontal Scaling
  - Cluster of identical machines (nodes) connected over High speed network
  - Multiple machines share the data so that no machine is bearing the complete load
- Elastic Scaling
  - Cluster will be able to seamlessly scale up and scale back down
  - Scale UP
    - By adding nodes and they can start serving clients
    - No cluster restart / No query change / No balancing
    - JUST add an another machine (node) to the cluster
  - Scale Down
    - Just unplug the system
    - Since Cassandra keeps multiple copies of the same data in more than one node there wont be any loss of data.

Quick Wins: Install and Start Cassandra

- Download (Datastax or Open source apache version)
  - Latest Apache Cassandra from http://cassandra.apache.org/download/
  - You will get apache-cassandra-3.11.0-bin.tar.gz
- Untar the binary .gz file
  - tar -zxvf apache-cassandra-3.11.0-bin.tar.gz
- Configuration
  - /conf/cassandra.yaml
- Set JAVA_HOME and PATH
  - export JAVA_HOME=/usr/local/java/jdk1.8.0_111
  - export PATH=$JAVA_HOME/bin:$PATH
  - export JRE_HOME=/usr/local/java/jdk1.8.0_111
  - export PATH=$JRE_HOME/bin:$PATH
- Set CASSANDRA home and PATH
  - export CASSANDRA_HOME=/home/cuda/apache-cassandra-3.11.0
  - export PATH=$CASSANDRA_HOME/bin:$PATH
- Start cassandra
  - cassandra -f

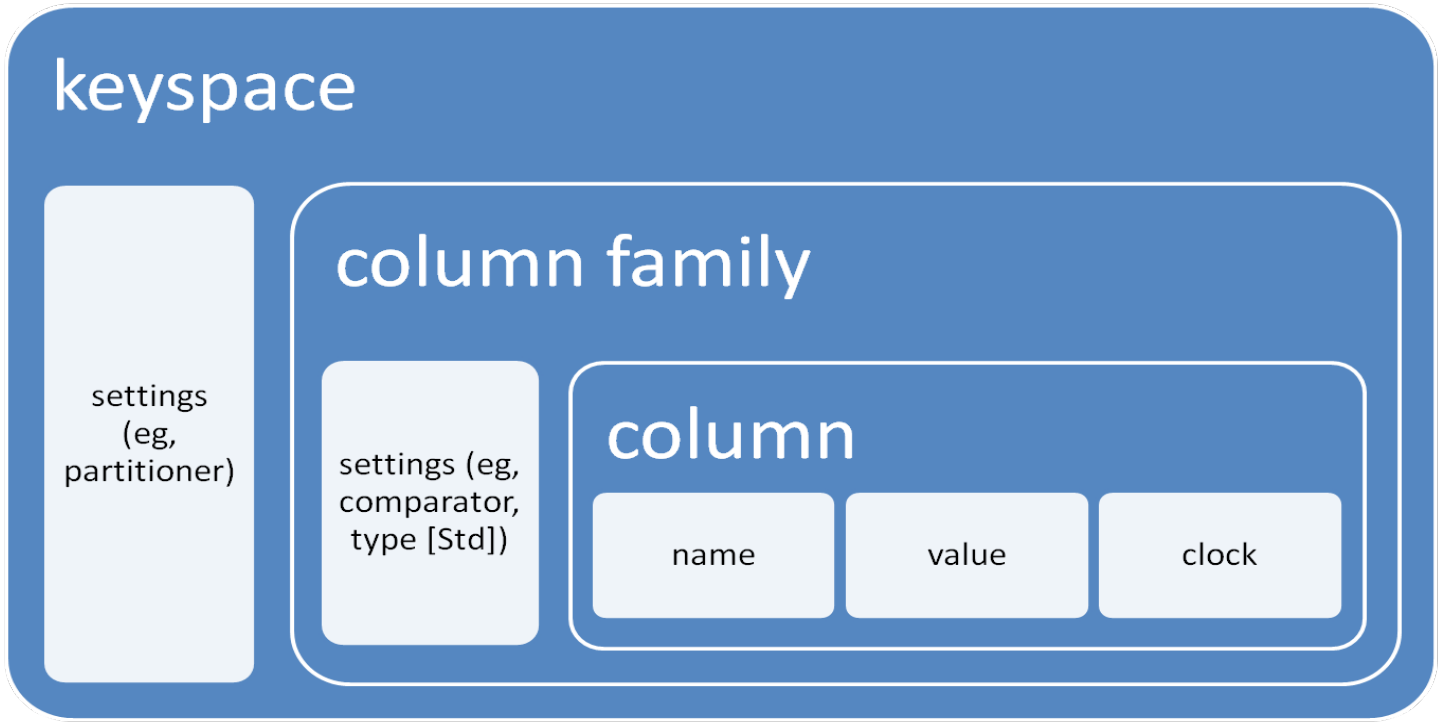
    (Node localhost/127.0.0.1 state jump to NORMAL)

Quick Wins: CQL

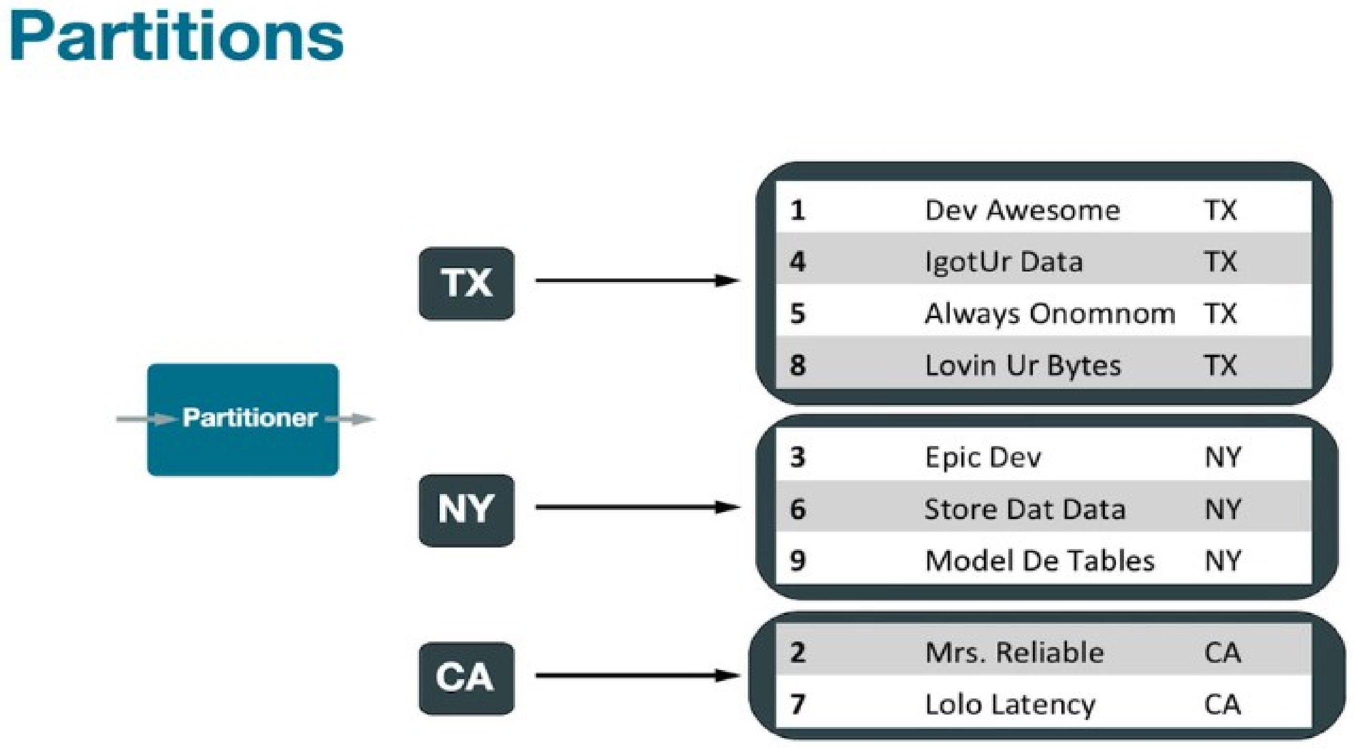
- CQL - Cassandra Query Language
  - DDL and Query language for Cassandra
  - Very similar to SQL, but not SQL
- Create Keyspaces
  - Key spaces are
  - Top-level Namespace / Containers
  - Similar to Relational Database schema
  - Stores the replication information
- Use command can be used to switch between different keyspaces
- Within the Keyspaces you have Table / Columfamily
- SELECT * FROM users;
- SELECT does not allow JOINS (involving more than 1 table)
- basic data types
  - text
  - int;
  - UUID - Universally Unique Identifier , generate via uuid()
  - TIMEUUID , generate via now();
  - sortable
- Copy - imports/exports data from .csv files

Cassandra Data Model

- Keyspace ~= database , typically one per application
- Some settings are configurable only per keyspace
- Row oriented - Each row is uniquely identified by a key


keyspace
settings (eg, partitioner)
column family
settings (eg, comparator, type [Std])
column
name
value
clock

Cassandra Data Model: Partitions



1	Dev Awesome	TX
4	IgotUr Data	TX
5	Always Onomnom	TX
8	Lovin Ur Bytes	TX
3	Epic Dev	NY
6	Store Dat Data	NY
9	Model De Tables	NY
2	Mrs. Reliable	CA
7	Lolo Latency	CA

Partitioners

Murmur3Partitioner (default): Uniformly distributes data across the cluster based on MurmurHash hash values.

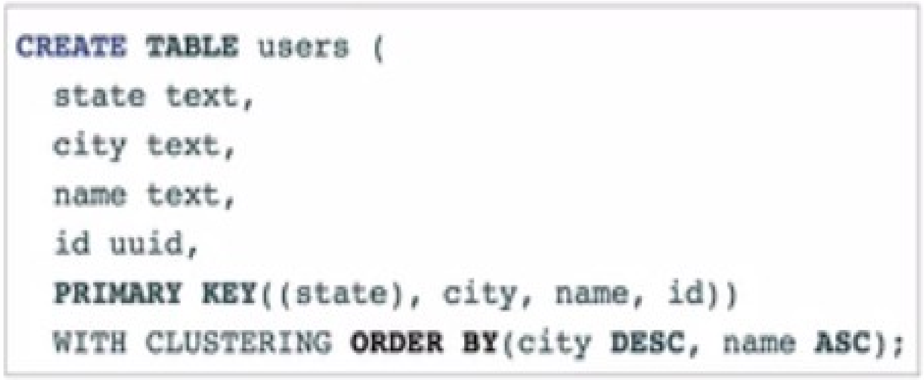
RandomPartitioner: Uniformly distributes data across the cluster based on MD5 hash values.

ByteOrderedPartitioner: Keeps an ordered distribution of data lexically by key bytes

- Partitions are groupings of data that will be co-located on storage
- First value in the primary key is the partition key
- If only one primary key, it becomes the partition key
- Partitioner finds out a numeric Token value from the key

Cassandra Data Model: Clustering Columns

- Clustering allows us to change default ordering
- PrimaryKey((State),City)
- Here State is the Partition key and City is the clustering column
- Clustering columns follow ascending order by default
- Change ordering direction via WITH CLUSTERING ORDER BY
- Ordering is done when data is stored



```
CREATE TABLE users (
  state text,
  city text,
  name text,
  id uuid,
  PRIMARY KEY((state), city, name, id))
  WITH CLUSTERING ORDER BY(city DESC, name ASC);
```

Allow Filtering

- Relaxes the querying on partition key constraint
- Causes Cassandra to scan all partitions
- Do not use it
    - Unless you really have to
    - Best on small data sets

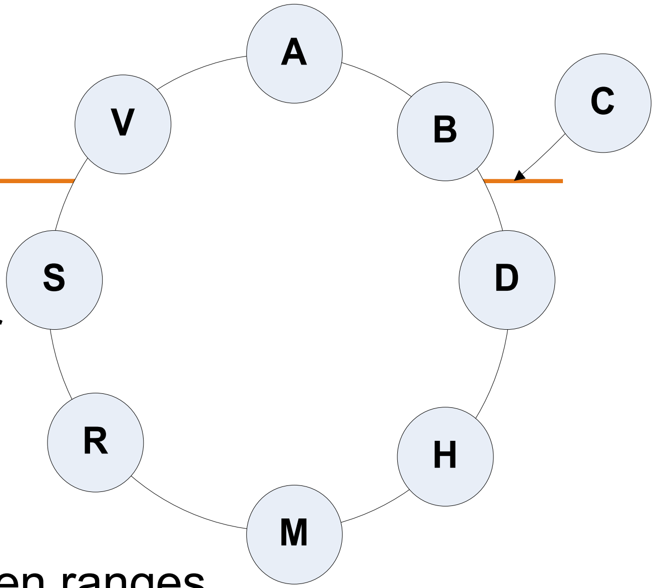
Application Connectivity: Drivers

- Driver connects to Cassandra database from your application
  - Datastax driver
  - Thrift client driver
  - Similar to JDBC for RDBMS
- Driver listens and manages changes to the cluster (Node addition/deletion etc.)
- Driver specifies multiple connection points to the cluster
- Setup
  - Create a cluster object
  - Use the cluster to obtain a session
  - Session manages connection to the cluster
  - Insert - session.execute(“ CQL command for data insertion”)
  - Select – result = session.execute(“SELECT * from mytable …..”)
  - Print result.Firstname, result.Lastname

Distributed Architecture: Node

- Cassandra is a Java Process running on a JVM hosted on a Node
- Nodes participate in a cluster
- Nodes can be Servers, Virtual Machines, Docker containers
- Nodes should have minimum of 8GB memory
- Nodes stores data on local disks - 1 to 3 Terrabytes disk
- SAN or anything that depends on Network for storage is not recommended
- The TPS rate depends on the number of CPU cores on the node
- nodetool info - See information on running node
- nodetool info -T - See all 256 tokens used by a node
- nodetool status - See status of all nodes in the cluster

Distributed Architecture: Ring



- Cassandra Ring - Clustering for scaling
- Spread the load by adding more nodes to the cluster
- Clients connect to any node of the cluster
- The connected node becomes the coordinator
- Each node is assigned different ranges of data - Token ranges

  (1) nodetool describecluster (2) nodetool ring (3) *nodetool describering table*

- The coordinator routes the data to the proper node based on the token
- The partitioner does the job of creating the tokens
- Token aware drivers can route data directly to the destination node
- Nodes can be added to the cluster online (redistributes token ranges)
- Nodes can be removed from a live cluster - decommissioning
- States of a node - joining, leaving, up, and down

Distributed Architecture: Peer-to-peer

- Client-Server Model for RDBMS
  - A leader node is responsible for Inserts,Updates,Reads
  - If the leader goes down, data becomes inaccessible
  - Failover clusters , no service during failover (High availability)
  - Split-brain situations - only one side is seen by clients
- In P2P
  - There is no permanent leader in Cassandra
  - Replicas of data inside the cluster
  - The replicas are independent , peer to peer
  - If the cluster splits, each side can be seen by the clients

Distributed Architecture: Vnodes

- Token assignment range fixes the range of data assigned to the nodes
- Hot spots can occur when there is disparity in token ranges
- If a new node is added to the cluster, we need to bifurcate the ranges perfectly
- Bifurcation of token ranges for equal distribution is challenging
- We can get this done using Virtual nodes (Vnodes)
- Vnodes allow us to create individual smaller ranges per node
- The default assignment is 256 Vnodes per node (Pre 3.0)
- In Cassandra 3.0 and beyond, the token ranges are always automatic
- Adding/removing nodes with Vnodes does not make the cluster unbalanced
- Vnodes automate token range assignment
- Configuration in cassandra.yaml
    - num_tokens - value greater than one turns on Vnodes

Distributed Architecture: Gossip

- Gossip keeps the Cassandra cluster working well
- One node tells its information to another node
- Then that node's job is to tell other nodes
  - Each node initiates a gossip round every few seconds
  - Picks 1 to 3 nodes (any node) to gossip with
  - Nodes do not track which nodes they gossiped with prior
- Reliably and efficiently spreads node metadata through the cluster
- Information sharing by gossip takes time (unpredictable)
- What information in gossip packet ?
  - Cluster metadata
  - State of the cluster, health of the node
  - Location information, Load on the node
- nodetool gossipinfo

Distributed Architecture: Snitch

- Snitch controls the node placement in data centers
- Determines/declares each node's rack and data center
- Defines the topology of the cluster
- Several different types of snitches (Eg: SimpleSnitch)
  - Ec2Snitch - for cloud
  - Property File Snitch (for big clusters)
  - Gossiping Property File Snitch
    - In cassandra-rackdc.properties file you put the data center this node belongs to and the rack that it belongs to.
- Configured in cassandra.yaml
- nodetool describecluster

Distributed Architecture: Snitch

- Snitch controls the node placement in data centers
- Determines/declares each node's rack and data center
- Defines the topology of the cluster
- Several different types of snitches (Eg: SimpleSnitch)
  - Ec2Snitch - for cloud
  - Property File Snitch (for big clusters)
  - Gossiping Property File Snitch
    - In cassandra-rackdc.properties file you put the data center this node belongs to and the rack that it belongs to.
- Configured in cassandra.yaml
- nodetool describecluster

Replication/Consistency: Replication

- Keyspace holds replication information
- Replication factor – 1, 2, or 3

1. Each node stores its own data, only one place where you store the data (Sharding)
2. Each node stores data for itself and its neighbor
3. Each node stores its own data, and its neighbors data, and its neighbor's neighbor's data.

- Replication factor 3 is recommended for production, gives strong resilience to failures

Data Placement Strategies

Replication Strategy: two options:

1. *SimpleStrategy*
2. *NetworkTopologyStrategy*

<u>SimpleStrategy</u>: uses the Partitioner, of which there are two kinds

- *RandomPartitioner*: Chord-like hash partitioning
- *ByteOrderedPartitioner*: Assigns ranges of keys to servers.
- Easier for *<u>range queries</u>* (e.g., Get me all twitter users starting with [a-b])

<u>NetworkTopologyStrategy</u>: for multi-DC deployments

- Two replicas per DC
- Three replicas per DC
- Per DC
- First replica placed according to Partitioner
- Then go clockwise around ring until you hit a different rack

Replication/Consistency: Consistency

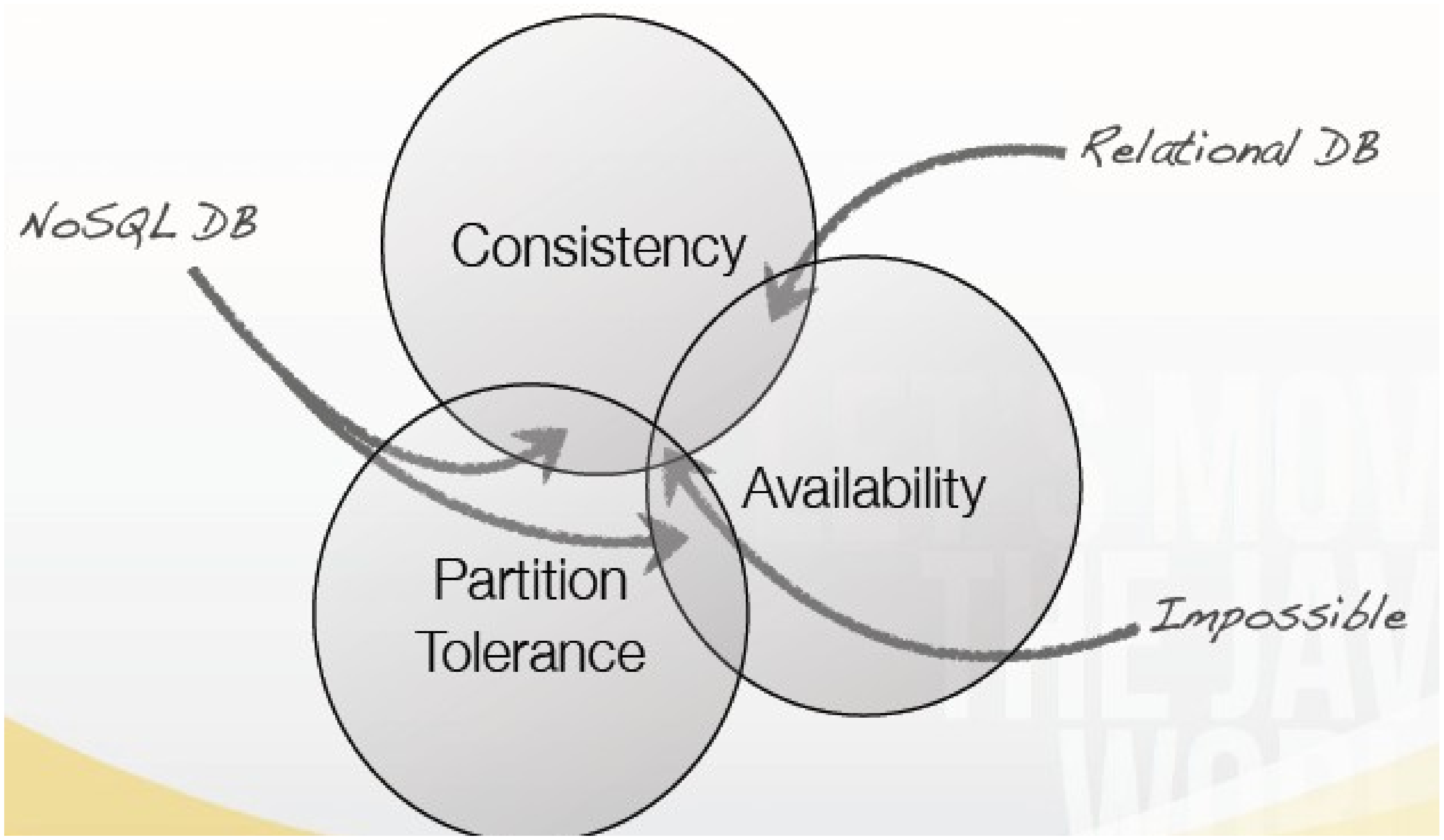
- ACID is strong consistency available in RDBMS
  - ➢ Atomicity: Either all parts of a transaction must be completed or none.
  - ➢ Consistency: The integrity of the database is preserved by all transactions. The database is not left in an invalid state after a transaction.
  - ➢ Isolation: A transaction must be run isolated in order to guarantee that any inconsistency in the data involved does not affect other transactions.
  - ➢ Durability: The changes made by a completed transaction must be preserved or in other words be durable.
- NoSQL provides weak consistency

  Consistency - All nodes have same data at any time

  Availability - System allows operations all the time

  Partition tolerance - System continues to work in spite of network partitions
- Cassandra provides Availability and Partition Tolerance
- What about consistency
  - Tunable consistency
  - Eventual consistency

Replication/Consistency: CAP



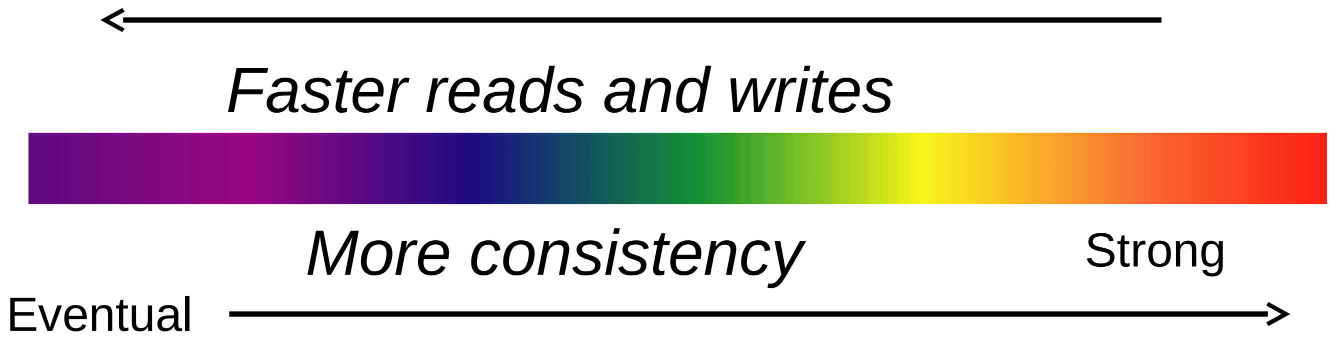
Programmable Consistency

- Read Consistency levels:
  - One: Return from the first node that responds
  - Quorom: Query from all nodes and respond with the one that has latest timestamp once a majority of nodes responded
  - All: Query from all nodes and respond with the one that has latest timestamp once all nodes responded. An unresponsive node will fail the node
- Write Consistency levels:
  - Zero: Ensure nothing. Asynchronous write done in background
  - Any: Ensure that the write is written to at least 1 node
  - One: Ensure that the write is written to at least 1 node's commit log and memory table before receipt to client
  - Quorum: Ensure that the write goes to node/2 + 1
  - All: Ensure that writes go to all nodes. An unresponsive node would fail the write

Consistency Spectrum

The higher the consistency, the less chance you may get stale data

- Pay for this with latency
- Depends on your situational needs



Replication/Consistency: Hinted Handoff

- Node is offline, what happens to the data we go to write there
- Cassandra is Always writable: Hinted Handoff mechanism
- If any replica is down, the coordinator writes to all other replicas, and keeps the write locally until down replica comes back up.
- When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours ~ 3 hours).
- When the node comes back it can write the hints to that node
- If we want to reboot a node, we are free to do that

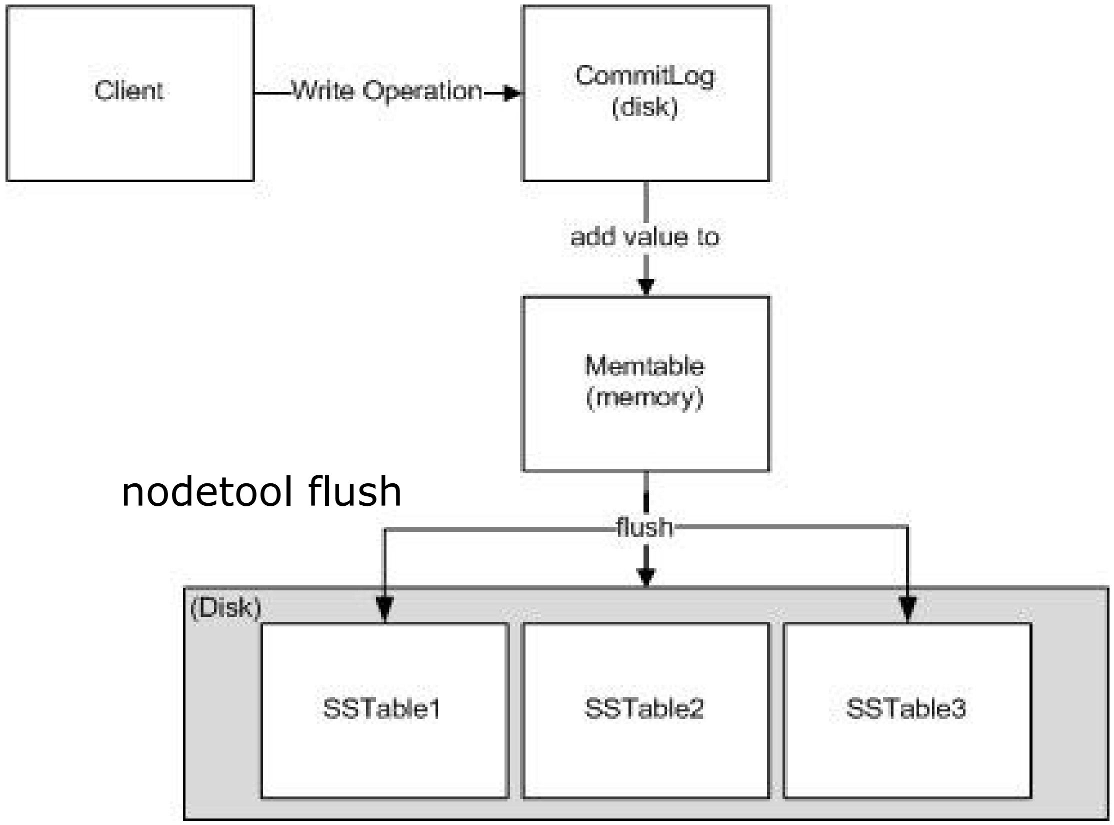
"*I have the write information that is intended for node B. I'm going to hang onto this write, and I'll notice when node B comes back online; when it does, I'll send it the write request*"

Replication/Consistency: Read-repair

- Network partitions can cause nodes to get out of sync
- Hinted-handoff works only for 3 hours
- Read repair is the process of making sure your data is consistent across all replicas
- It finds an answer to the question "*Oh, is this data consistent with the rest of the replicas?*" at the time of read
- Read repair is done by comparing the timestamps
- Choose data with latest timestamp, return that data to the client and update it on all other nodes
- Repair manually at least once in 10 days using *nodetool repair* command - (repairs one or more tables)

Internal Architecture: Write-Path

- A client issues a write request to a random node in the Cassandra cluster.
- The "Partitioner" determines the nodes responsible for the data
- On receiving a Write request, log it in disk commit log
- Make changes to appropriate memtables – In-memory representation of multiple key-value pairs



- Later, when memtable is full or old, flush to disk , remove commitLog
- Data File: An SSTable (Sorted String Table) – list of key value pairs, sorted by key
- Index file: An SSTable – (key, position in data sstable) pairs

Internal Architecture: Read-Path

- Read: Similar to writes, except
  - Need to touch log and multiple SSTables
  - Coordinator can contact closest replica (e.g., in same rack)
  - Coordinator also fetches from multiple replicas
    - check consistency in the background, initiating a read-repair if any two values are different
    - Makes read slower than writes (but still fast)
    - Read repair: uses gossip (remember this?)
  - May be slower than writes
- Delete: don’t delete item right away
  - add a tombstone to the log
  - Compaction will remove tombstone and delete item

Internal Architecture: Compaction

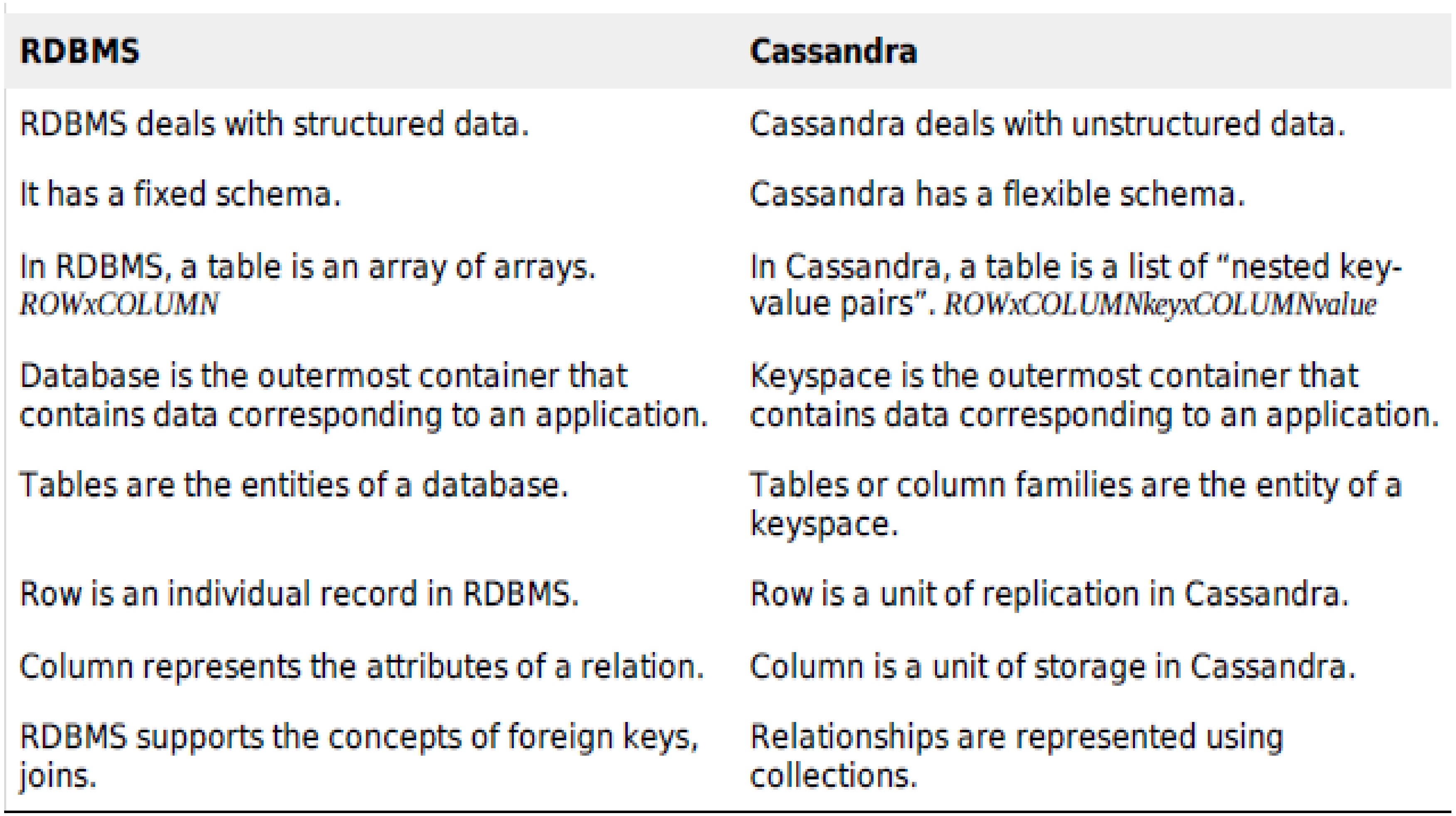
- Compaction is delayed IO on your file, Cleaning up the disk space
- Compaction operations occur periodically to re-write and combine SSTables.
- Required because SSTables are immutable (no modifications once written)
- Compactions prune deleted data and merge disparate row data into new SSTables in order to reclaim disk space and keep read operations optimized
- Multiple Compaction Strategies are included with Cassandra, and each is optimized for a different use case:

1. SizeTiered Compaction Strategy (STCS) (Default) - Good for insert-heavy and general workloads
2. Leveled Compaction Strategy (LCS) - Best for read-heavy workloads
3. DateTiered Compaction Strategy (DTCS) - Designed for use with time-series data. stores data written within a the same time period in the same SSTable.

- nodetool compact nodetool compactionhistory

- From cqlsh , compaction strategy can be changed
- ALTER TABLE fresher.world2 WITH compaction = { 'class' : 'LeveledCompactionStrategy'};
- ALTER TABLE fresher.world2 WITH compaction = { 'class' : 'SizeTieredCompactionStrategy'};

Cassandra Data Model: Comparison with RDBMS



RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. *ROWxCOLUMN*	In Cassandra, a table is a list of "nested key-value pairs". *ROWxCOLUMNkeyxCOLUMNvalue*
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

Hands On Sessions

nodetool command

1. nodetool info
2. nodetool status

(Selected subset of CQL commands)

1. CREATE KEYSPACE - Define a new key space and its replica placement strategy.
2. CREATE TABLE - Define a new table.
3. CREATE INDEX - Define a new index on a single column of a table.
4. ALTER KEYSPACE - Change property values of a keyspace.
5. ALTER TABLE - Modify the column metadata of a table.
6. INSERT - Add or update columns.
7. SELECT - Retrieve data from a Cassandra table.
8. TRUNCATE - Remove all data from a table.
9. UPDATE - Update columns in a row.
10. COPY – Data from text file into / from table
11. DELETE - Removes entire rows or one or more columns from one or more rows.
12. DROP INDEX - Drop the named index.
13. DROP TABLE - Remove the named table.
14. DROP KEYSPACE - Remove the key Space.

Course Conclusion

- You have been introduced to the Cassandra NoSQL
  http://nosql-database.org/
- To make meaningful contributions, you need to learn it completely
- Cassandra versions keep on changing
- Always start with the latest version – Current version 3.11.0
  Keep watching http://cassandra.apache.org/
- **Resources:**

1. Learn Cassandra
   - https://www.gitbook.com/book/teddyma/learncassandra/details
   - https://teddyma.gitbooks.io/learncassandra/content/index.html
2. Datastax Academy
   - https://academy.datastax.com/
   - https://academy.datastax.com/resources/ds201-foundations-apache-cassandra