Introduction to Database Systems: CS312 Cassandra Theory

Oliver Bonham-Carter

8 April 2019

A NoSQL Database Management System

- Apache Cassandra is a massively scalable open source non-relational database
- Offers continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones.
- Founded at Facebook 2008, developed at Apache in 2010

- `http://cassandra.apache.org/`
- `https://academy.datastax.com/planet-cassandra/cassandra`

Cassandra is different from relational DBs

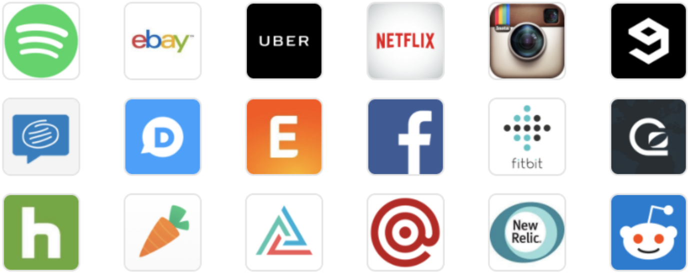
Relational (Sqlite3)	**Cassandra**
Handles moderate incoming data velocity	Handles high incoming data velocity
Data arriving from one/few locations	Data arriving from many locations
Manages primarily structured data	Manages all types of data
Supports complex/nested transactions	Supports simple transactions
Single points of failure with failover	No single points of failure; constant uptime
Supports moderate data volumes	Supports very high data volumes

Cassandra is different from relational DBs

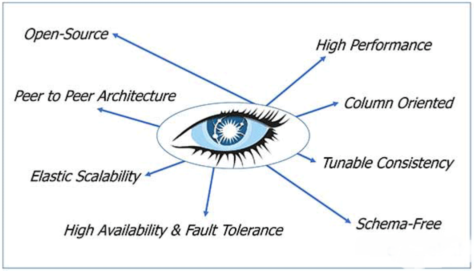
Relational (Sqlite3)	**Cassandra**
Centralized deployments	Decentralized deployments
Data written in mostly one location	Data written in many locations
Supports read scalability (with consistency sacrifices)	Supports read and write scalability
Deployed in vertical scale up fashion	Deployed in horizontal scale out fashion

Some of its users ...

... and others, too!



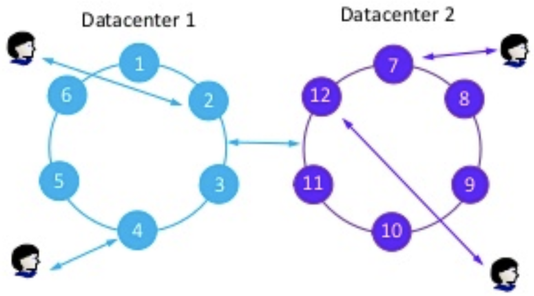
Key Features



- Built-for-scale architecture: Cassandra is capable of handling petabytes of information and thousands of concurrent users/operations per second (across multiple data centers) as easily as it can manage much smaller amounts of data and user traffic.
- Unlike other master-slave or sharded systems, Cassandra has no single point of failure and therefore is capable of offering true continuous availability.

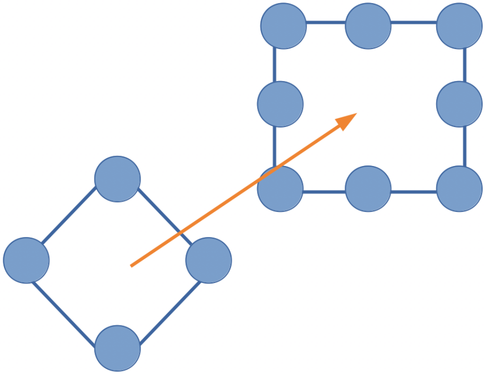
Key Features

Distributed and Decentralized



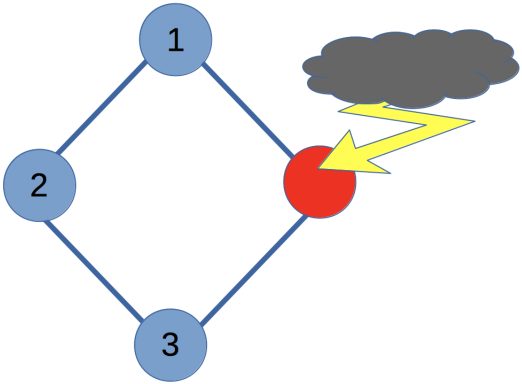
- Distributed: able to be run across several machines at diverse locations
- Active everywhere design: all nodes may be written to and read from.
- No master-slave configurations: all nodes *gossip* meaning they share information using peer-to-peer architecture

Key Features
Elastic Scalability



- Horizontal scaling: adding more machines to handle loads (a linear increase in performance)
- Flexible and dynamic data model: supports modern data types with fast writes and reads.

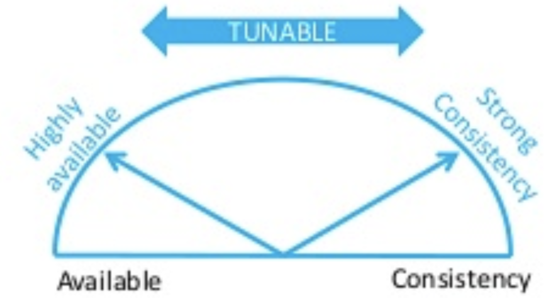
Key Features
High Availability and Fault Tolerance



- No single point of failure: no “master” node
- Continuous availability: offers redundancy of both data and node function, which eliminate single points of failure and provide constant uptime.

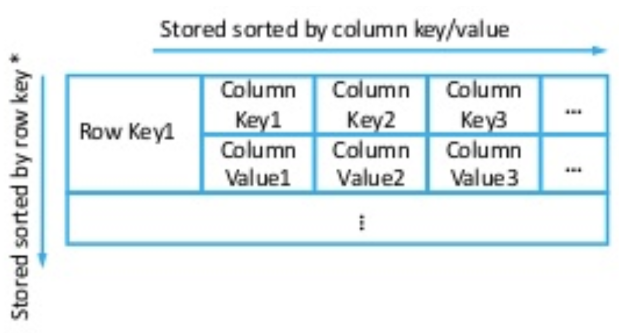
Key Features

Tune up / down the consistency factor



- Tunable Consistency: chose between *strong* and *eventual* consistency
- Adjustable read and write operations (separately)
- Conflicts are solved during reading while the focus is on the write performance.

Tables and Columns



- Data is stored in sparse multidimensional hash tables
- Rows may have multiple columns, not necessary to have same number of columns for each row
- No relations as in Sqlite3
- Each row has unique key that also serves partitioning

Database Language Guide

SQL systems versus CQL (i.e., Cassandra's Query Client)

SQL	**CQL**	**Elaboration**
Database	Keyspace	Contain tables. A *keyspace* defines the replication factor (i.e., the number of replica nodes for ensuring reliability and fault tolerance) and replication strategy for all tables that it contains.
Materialized view	Table + Partition	A CQL table defines a schema much like an SQL table. However, CQL tables contain partitions and each partition contains rows. The combination of a CQL table plus a partition is similar to a materialized view in SQL. A macro-like system of running queries.

Database Language Guide

SQL systems versus CQL (i.e., Cassandra's Query Client)

SQL	CQL	Elaboration
Primary key		An SQL primary key is a unique identifier per row. There is no direct equivalent in CQL, although the term "primary key" is used in CQL.
	Primary key	A CQL primary key is a composite key that may define the partition key and optionally clustering columns.
Column	Column	The concept of a column is very similar in Cassandra vs. an RDMBS. Although how a column is physically stored is very different in Cassandra vs. an RDBMS.
Value	Value	The concept of a value is very similar in Cassandra vs. an RDMBS.

Database Language Guide

SQL systems versus CQL (i.e., Cassandra's Query Client)

SQL	**CQL**	**Elaboration**
ORDER BY	Clustering columns	Cassandra stores data in sorted order. Therefore, you achieve the equivalent of an SQL ORDER BY through the selection of clustering columns.
JOIN	Achieved via materialized view	As mentioned above, a CQL table plus partition is conceptually closer to a materialized view than a relational table. In a materialized view in an RDBMS you would achieve the equivalent of a JOIN by denormalizing data. The same concept applies to Cassandra where you denormalize data.

Ref: `http://exponential.io/blog/2015/01/08/cassandra-terminology/`

Getting started with Cassandra

- Copy your Cassandre setup tar file (apache-cassandra-3.11.2-bin.tar.gz) to a desktop directory. Do not copy this file or open it in your submission directory!!!
- Click on this file to unpack its contents
- Locate the `bin` directory and then locate file: `cassandra` and file: `cqlsh`

Start the `cassandra` server
Note: Control-C to exit.

```
./cassandra -f
```

With new another terminal, start the `cqlsh` client

```
./cqlsh
```

Keyspaces

Similar to a schema

Find keyspaces (i.e., resembling the schema concept of Sqlite3 systems or tables

```
describe keyspaces;
describe tables;
```

Start a new keyspaces

```
create keyspace mydb with replication =
{ 'class':'SimpleStrategy',
 'replication_factor':1 };
```

Use a keyspace

```
use mydb;
```

Remove a keyspace

```
/*Drop the "my=db" keyspace*/
DROP KEYSPACE mydb;
```

Inserting
Very Similar to SQL

Build a Table

```
create table emp(
  empid int primary key,
  emp_first varchar,
  emp_last varchar,
  emp_dept varchar);
```

Insert Data Using CQL INSERT command

```
insert into emp
  ( empid, emp_first,emp_last,emp_dept )
   values ( 1, 'Fred','Smith','English' );
insert into emp
  ( empid,emp_first,emp_last,emp_dept )
   values ( 2, 'Bob','Alison','English' );
insert into emp
  ( empid,emp_first,emp_last,emp_dept )
   values ( 3, 'Judy','Miller','French' );
insert into emp
  ( empid,emp_first,emp_last,emp_dept )
   values ( 4, 'Jasmin','Jones','Computer Science' );
```

ALLEGHENY COLLEGE

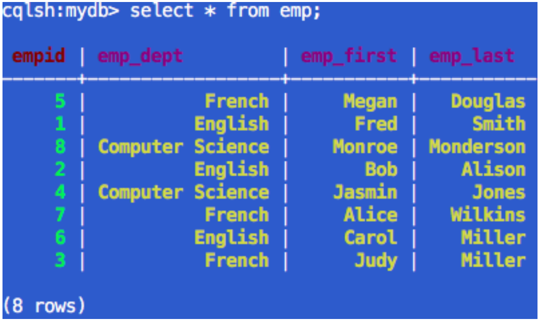
General Query

Study the schema and indexes (*query*-able columns)

```
describe schema
```

Simple Query

```
select * from emp;
```

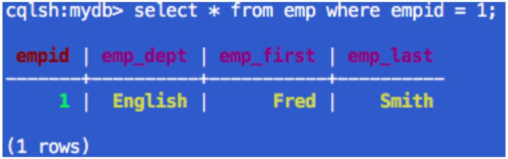


```
cqlsh:mydb> select * from emp;

 empid | emp_dept         | emp_first | emp_last
-------+------------------+-----------+-----------
     5 |           French |     Megan |   Douglas
     1 |          English |      Fred |     Smith
     8 | Computer Science |    Monroe | Monderson
     2 |          English |       Bob |    Alison
     4 | Computer Science |    Jasmin |     Jones
     7 |           French |     Alice |   Wilkins
     6 |          English |     Carol |    Miller
     3 |           French |      Judy |    Miller

(8 rows)
```

General Query

Simple Query

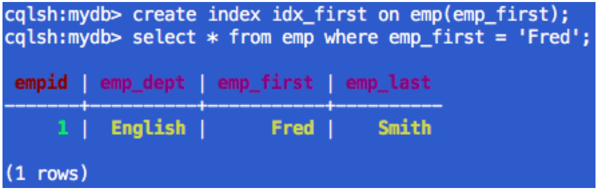
```
select * from emp where empid = 1;
```



```
cqlsh:mydb> select * from emp where empid = 1;

 empid | emp_dept | emp_first | emp_last
-------+----------+-----------+----------
     1 |  English |      Fred |    Smith

(1 rows)
```

Simple Query

```
create index idx_first on emp(emp_first); /* run this once, ever in DB */
select * from emp where emp_first = 'Fred';
```



```
cqlsh:mydb> create index idx_first on emp(emp_first);
cqlsh:mydb> select * from emp where emp_first = 'Fred';

 empid | emp_dept | emp_first | emp_last
-------+----------+-----------+----------
     1 |  English |      Fred |    Smith

(1 rows)
```

Creating Indexes

- In Cassandra, if you want to query columns other than the primary key, you need to create a secondary index on them

```
select * from emp where emp_dept = 'English';
```

Try creating an index and run again...

```
create index idx_dept on emp(emp_dept);
select * from emp where emp_dept = 'English';
```

```
select * from emp where emp_first = 'Fred';

create index idx_first on emp(emp_first);
select * from emp where emp_first = 'Fred';
```

```
select * from emp where emp_last = 'Miller';

create index idx_last on emp(emp_last);
select * from emp where emp_last = 'Miller';
```

More complex queries

Insert more data

```
insert into emp
 (empid, emp_first, emp_last, emp_dept)
 values (5,'Megan','Douglas','French');
insert into emp
 (empid, emp_first, emp_last, emp_dept)
 values (6,'Carol','Miller','English');
insert into emp
 (empid, emp_first, emp_last, emp_dept)
 values (7,'Alice','Wilkins','French');
insert into emp
 (empid, emp_first, emp_last, emp_dept)
 values (8,'Monroe','Monderson','Computer Science');
```

To do a more complicated query, we first have to index the column.

```
/* Create an index to find a last name */
create index idx_last on emp(emp_last);
describe index idx_first /*a working index?*/

select * from emp where emp_last = 'Alison';
```

How to shut down a session

Remove a keyspace

```
/*Drop the "mydb" keyspace*/
DROP KEYSPACE mydb;
```

Remove a table

```
/*Drop the "emp" table*/
DROP TABLE emp;
```

Closing down

- `exit` in the client terminal
- `Control-C` in the server terminal

Consider this...

- Can you create and populate a new Cassandra database?