# Cassandra Lab Report

By Yihui Feng, Scott Sun, Shannon Jin

## Introduction

Cassandra is an open source, distributed, wide-column store database that provides scalability and high availability without compromising performance. Cassandra supports replication across multiple data centers which allows lower latency operations for users and surviving regional outages.

## Installation

### Prerequisites

- Have Java8 installed:

```
$ sudo apt update
$ sudo apt install openjdk-8-jdk
```

- The installation can be verified by

```
$ java -version
```

- You should see output that looks like this

```
openjdk version "1.8.0_222"
OpenJDK Runtime Environment (build 1.8.0_222-8u222-b10-1ubuntu1~16.04.1-b10)
OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)
```

### To install the Debian package

- Add the Apache Cassandra repository

```
$ echo "deb http://downloads.apache.org/cassandra/debian 40x main" | sudo tee -a
/etc/apt/sources.list.d/cassandra.sources.list
deb http://downloads.apache.org/cassandra/debian 40x main
```

- Add the Apache Cassandra repository keys to the list of trusted keys

```
$ curl https://downloads.apache.org/cassandra/KEYS | sudo apt-key add -
```

- Install Cassandra with apt

```
$ sudo apt update
$ sudo apt install cassandra
```

# Basic Commands to Run Cassandra

## Start Cassandra

- To start Cassandra

```
$ sudo service cassandra start
```

- To check status of Cassandra

```
$ sudo service cassandra status
```

- To stop cassandra

```
$ sudo service cassandra stopa
```

## Connect to Cassandra using the CQL Shell

- run the executable `cqlsh` with the `listen_address`, which is the localhost by default
- Cqlsh stands for Cassandra Query Language Shell, it's a shell that uses a SQL like language to perform operations just like in a normal DBMS.

```
$ cqlsh listen_address
Connected to Test Cluster at localhost:9042.
[cqlsh 5.0.1 | Cassandra 3.8 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
cqlsh> SELECT cluster_name, listen_address FROM system.local;

 cluster_name | listen_address
--------------+----------------
 Test Cluster |      127.0.0.1

(1 rows)
cqlsh>
```

# Scalability

Cassandra, linearly scalable, provides the strategies to increase the scalability both horizontally and vertically. It allows us to scale horizontally by adding more datacenters and scale vertically by adding more nodes within datacenters. Datacenter refers to a group of nodes related to within a cluster for replication purposes. The level of consistency is controlled by the replication factor (number of copies in the cluster). Sharding is also supported. Sharding in Cassandra's context is referred to as partitioning, which packs the rows with the same partition key and stores them in one single instance. Rows with other partition keys may be sent to other nodes.

# Cassandra Cluster (Ring Structure) Setup

- First, stop Cassandra on each node and delete the default database

```
$ sudo service cassandra stop
$ sudo rm -rf /var/lib/cassandra/data/system/*
```

- Next, change the configuration file cassandra.yaml for each node

```
$ sudo vi /etc/cassandra/cassandra.yaml
```

- After changing the configuration, restart each node and check the detailed status

```
$ nodetool status
```

Properties to set includes:

- `cluster_name`: (the name of your cluster)

- `-seeds`: list of **internal** IP addresses of each seed node in the cluster, which are separated by comma; the seed nodes serve to: 1. allow the new nodes to find the cluster and catch up, and 2. Assist in gossip convergence

- `listen_address`: the **internal** IP address of the **local** node; this is the IP address that other nodes in the cluster will connect to, and by default, it is set to localhost.

- `rpc_address`: the **external** IP address of the local **node**; this is the IP address for remote procedure calls, and by default, it is set to localhost.

- `endpoint_snitch`: Name of the snitch that tells Cassandra about what its network looks like. Options are `SimpleSnitch`, `GossipingPropertyFileSnitch`, `PropertyFileSnitch,` etc. In our configuration we use `SimpleSnitch` since our data application implements single-datacenter deployments.

```
cluster_name: 'TutorialCluster'
...
seed_provide:
 - class_name: org.apache.cassandra.locator.SimpleSeedProvider
 parameters:
   - seeds: "your_server_ip,...,your_server_ip_n"
...
listen_address: your_server_ip (internal)
...
rpc_address: your_server_ip (external)
...
endpoint_snitch: GossipingPropertyFileSnitch
...
```

# Cassandra Keyspace Operations

## Create a new Keyspace

- A keyspace in Cassandra is a concept similar to a database in mongoDB, a namespace in Redis. This is the level where you can specify how you want data to be partitioned and how you want to manage data replication.
- You can create a keyspace named “sample” by the following command

```
cqlsh> create keyspace sample with replication =
{'class':'SimpleStrategy','replication_factor':num_replicas};
```

- `SimpleStrategy` only uses a single data center.
  You can specify a `NetworkTopologyStrategy` to allow for multiple data centers.
- `Replication_factor` is used to indicate the number of nodes needed to respond to a read/write request for the operation to succeed. For example, you have a cluster of 10 nodes with a `replication_factor` of 4, 4 responses from the 10 nodes are required to acknowledge for the operation to succeed.

## Modify/Delete a keyspace

- To alter a keyspace you can run the following

```
cqlsh> alter keyspace sample with replication =
{'class':'SimpleStrategy','replication_factor':1};
```

- To delete a keyspace

```
cqlsh> drop keyspace sample
```

- After deleting, you can check by running the following to see all remaining keyspaces

```
cqlsh> describe keyspaces
```

- Similar to moving into a database in MongoDB, you can move into a keyspace in Cassandra

```
cqlsh> use sample
```

- After moving into the keyspace,

```
cqlsh:sample> use sample
```

# Cassandra Table Operations

## Create a table

- To create a table in Cassandra, you should follow the following syntax

```
cqlsh:sample> create table sample_table (
      <column1_name> datatype PRIMARY KEY,
      <column2_name> datatype,
      <column3_name> datatype
);
```

- Define column/columns with name and datatype
- Declare a inline primary key of the table

```
cqlsh:sample> create table sample_table (
      <column2_name> datatype,
      <column3_name> datatype,
      <column3_name> datatype
      PRIMARY KEY(<column1_name>, <column2_name>))
      with clustering order by (<column2_name>)
);
```

- Or declare a composite primary key. Take the table above as an example, `Column1_name` is defined as the **partition key**, which is responsible for data distribution across the nodes. `Column2_name` is defined as the **distribution key**, which is responsible for data sorting within the partition.
- The last clause with `clustering order by` is used to make use of the on-disk sorting by the column. You can also add `ASC` (default order) or `DESC` to specify the direction of the ordering.

## Alter/Drop/Truncate a table

- Alter the table by adding/dropping a column after the table is created.

```
cqlsh:sample> alter table <table_name> drop <column_name> ;
cqlsh:sample> alter table <table_name> add <new_column_name> datatype ;
```

- It is straightforward to drop a table.

```
cqlsh:sample> drop table <table_name>
```

- It is also straightforward to remove all the data in a table by truncating it

```
cqlsh:sample> truncate table <table_name> ;
```

## Cassandra CRUD Operations

The CRUD operations in Cassandra are similar to those in the SQL relational database.

### Create:

```
cqlsh:sample> insert into <table_name>
      (<column1_name>, <column1_name>...)
      values(<value1>,<value2>...);
```

- You can also add the clause `if not exists` so that the row is inserted only if it does not exist.

### Update:

```
cqlsh:sample> update <table_name>
      set <column_name> = <new value>,...
      where <condition>;
```

- You can also add `allow filtering` (which is discussed in the section Creating Index in Cassandra)

### Read:

```
cqlsh:sample> select <column_name>,... from <table_name>;
```

- The clause `order by` only works for the primary key column
- The clause `limit n` works in the same manner as those in other DBs
- You can also add `allow filtering`

### Delete:

```
cqlsh:sample> delete from <table_name> where <condition>;
```

## Creating Index in Cassandra

- Create an index on a column.

```
cqlsh:sample> create index <index_name> on <tablename>(<column_name>)
```

- Drop an index on a column

```
cqlsh:sample> drop index <index_name>
```

In Cassandra, an index provides means to access rows with specified conditions of fields other than the primary key column(s). Without indexes, we have to add the clause `allow filtering` when we read or update the data using conditions. This undermines the performance of the operations when there is a large number of rows where the conditions are true for only a few of them.

# Mini Project - Parking Lot Management System

In this mini project, we will build a simple company parking lot management system and demonstrate how Cassandra can actualize this using Python. In order to have Cassandra interact with Python, we have to use the Cassandra driver package.

```
$pip install cassandra-driver
```

Here are some commands that you can use to connect the Python to a Cassandra server. Configuring the `row_factory` to `dict_factory` enables us to return each row in a `dict` structure.

```
>>> from cassandra.cluster import Cluster
>>> from cassandra.query import dict_factory
>>> cluster = Cluster(ip_addr)
>>> session = cluster.connect('sample')
>>> session.row_factory = dict_factory
```

The drive interacts with the server using the following command.

```
>>> session.execute("some_cql")
```

We can use the sample keyspace and create the table for our parking lot management system

```
cqlsh:sample> CREATE TABLE parking_lot (
    employee_uname text,
    cid int,
    car_make text,
    car_model text,
    position text,
    PRIMARY KEY (employee_uname, cid)
) WITH CLUSTERING ORDER BY (cid ASC);
```

In this table, the primary key is a composite of employee name and the car id, where the employee name is the partition key and the car id is the clustering key. Each of these composite keys represent a unique row. Even though Cassandra does not support constraints, the uniqueness of records are ensured. Large number of records may not apply to this sample case. Assume there is a large number of records for each partition, Cassandra sorts the data within each partition on the disk, and the retrieval of data is fast.

Now, we can insert some data into the table using the Python driver.

```
cqlsh:sample> INSERT INTO parking_lot
    (employee_uname, cid, car_make, car_model, position)
    VALUES ('John', rand_int,'BMW', 'M4', 'Manager')
    if not exists;
cqlsh:sample> INSERT INTO parking_lot
```

```
    (employee_uname, cid, car_make, car_model, position)
    VALUES ('Bob', rand_int,'Audi', 'A4','Assistant')
    if not exists;
cqlsh:sample> INSERT INTO parking_lot
    (employee_uname, cid, car_make, car_model, position)
    VALUES ('Bill', rand_int,'Lexus', 'LS500', 'CTO')
    if not exists;
```

We can also make a string for these CQL commands in Python, and use the driver to execute them. The Python code should follow the pattern as follows.

```
>>> cql = f"""
      INSERT INTO parking_lot (employee_uname, cid, car_make, car_model, position)
      VALUES ('{name}', {car_id}, '{car_make}', '{car_model}', '{position}')
      if not exists
"""
>>> session.execute(cql)
```

In order to use the `WHERE` clause with conditions on columns other than the primary key columns, we need to create indexes for columns including `car_make car_model` and `position`.

```
cqlsh:sample> CREATE INDEX car_make_index ON parking_lot (car_make);
cqlsh:sample> CREATE INDEX car_model_index ON parking_lot (car_model);
cqlsh:sample> CREATE INDEX car_position_index ON parking_lot (position);
```

After creating indexes on all the non-primary-key columns, we can make conditions on them after the `WHERE` clause when we read the data. Otherwise, we have to use `allow filtering` which is an inefficient way to retrieve data in Cassandra.

```
cqlsh:sample> SELECT * FROM parking_lot WHERE position = 'Assistant';
```

In update and delete operations, `WHERE` clause can be **only** used to identify the primary-key columns. If we set any condition for the non-primary-key column, error will be generated.

```
cqlsh:sample> UPDATE parking_lot SET position = 'CEO', car_make = '{car_make}',
      car_model = '{car_model}'
      WHERE employee_uname = 'John' and cid = 1;
cqlsh:sample> DELETE FROM parking_lot WHERE employee_uname = 'John' and cid = 1;
```

We can also make f-strings for these commands in Python and utilize the driver to execute the CQL. By combining all these features, we can develop a simple registration application with basic CRUD functionalities for the company's parking lot management system.

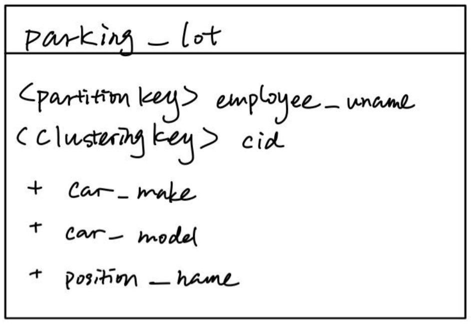
# Follow-on Project - Parking Lot Management System

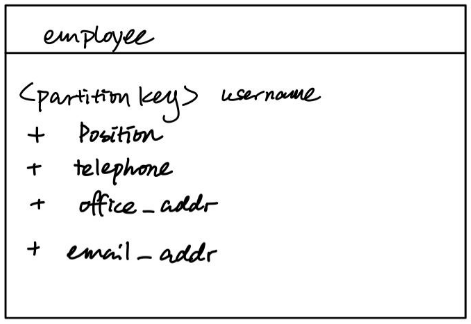
Following on the previous mini project's basic CRUD features, we can add more features on the company's parking system. Think about what data we need and the design of tables in the Cassandra setting.

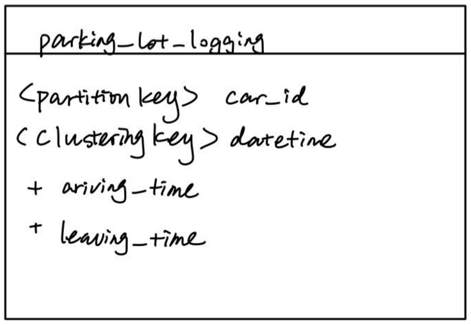
Here are the list tasks that the system can do:

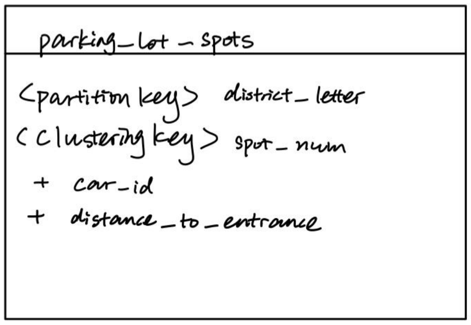
1. Basic CRUD operations for cars and their owners' information in the registration system
2. Find basic information (i.e. email address, office location, etc) about the car owner
3. Track each employee's arriving time and leaving time on daily basis and describe the pattern of these times
4. Track each car's position in the parking lot
5. Find and recommend available spots in the park lot

## Example Design for Tables in Project









## Advantages and disadvantages in using Cassandra

### Advantage:

1. In the cluster structure, all nodes are equally treated. We do not need worry about the master node and slave nodes reconfiguring when the master is down.
2. Cassandra has linear scalability. It provides both scaling strategies, horizontal strategy and vertical strategy, in increasing scalability. We can scale horizontally by adding more datacenters and scale vertically by adding more nodes within datacenters.
3. The level of consistency can be tuned flexibly even after the keyspaces have already been created. We can change the number of replication factors to vary the level of consistency.
4. Rows/Records within each partition are automatically sorted according to the clustering keys on disk.
5. Cassandra is fast in terms of writes.
6. The CQL syntax is similar to the SQL syntax in relational databases.

### Disadvantages:

1. There are no ACID transactions in Cassandra, as a cost of eventual consistency.
2. Cassandra has a strict restriction on the ordering of columns. Thus, the same information might be stored for multiples in order to actualize different features.
3. Cassandra does not support join and subquery, which can make the implementation difficult in some use cases.
4. Cassandra is slow in terms of reads.
5. Cassandra is Java-based, so sometimes the Garbage Collection can eat up some of the efficiency.
6. Cassandra is key-value pairs, performing aggregation actions such as SUM, AVG, MAX can be very resource intensive.

## Relevant Documentations

- Official Cassandra website:
  https://cassandra.apache.org/
- Datastax Cassandra driver documentation:
  https://docs.datastax.com/en/developer/python-driver/3.25/
- Datastax CQL documentation:
  https://docs.datastax.com/en/dse/6.8/cql/
- An article that introduces Cassandra:
  https://blog.knoldus.com/is-apache-cassandra-really-the-database-you-need/