VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Cassandra Data Model

Lecturer* : *Dr. Pavle Mogin

SWEN 432
Advanced Database Design and Implementation

Plan for Cassandra Data Model

- Prologue
- CQL Data Model
  - The key space
  - Column families and tables
  - Columns
- A running example
- Data Types

  - ***Readings:*** Have a look *at Useful Links at the Course Home Page*

Prologue

- Class of CDBMS according to CAP theorem: **AP**
- Internode communication: **Gossip Protocol**
- Partitioning: **Consistent Hashing**
- A cluster can span more then one data centre
- Replication: Definable by data centre
- Versioning: **timestamps**
- Consistency: **Tuneable** ranging from **strict** to **eventual**
  - Default consistency level is eventual
- Typical applications:
  - Applications with intensive write and moderate read requirements, like:
    - Time series data collection
    - Stock exchange
    - Vehicle tracking

Cassandra's Column Family DM

- Originally, Cassandra was classified as a column family NoSQL CDBMS
- Cassandra column family data model was defined in terms of:
  - Key spaces,
  - Column families,
  - Super column families,
  - Columns, and
  - Super columns.
- This approach has been abandoned in 2011 with the announcement of Cassandra V 0.8 with the new Cassandra Query Language (CQL)
  - We consider CQL V 3.1 that has been released within Cassandra V 2.1 in 2014

Formal Definition of the Column Family

- Cassandra column family is a map of a map:
  - An outer map keyed by a row key, and
  - An inner sorted map keyed by a column key

map<row_key, sorted_map<column_name, column_value>>

- In Cassandra, we can use:
  - Row keys to do efficient lookups, and
  - Column keys (names) to do less efficient lookups and range scans
  - The number of column keys is unbounded allowing wide rows
- A super column family turns the two nested maps into three nested maps as follows:

map<row_key, sorted_map<super_column_key, sorted_map<column_name, column_value>>>

- Each *column_value* has a time stamp for Cassandra to use internally for conflict resolution

CQL Data Model

- CQL is a SQL like declarative data language, having DDL and DML parts
- It adds a new level of abstraction to the Cassandra Column Family Data Model and describes it as:
  - A partitioned row store data model with tunable consistency,
  - Rows are organized into tables,
  - A defined prefix of a table's primary key is the partition key,
  - Within a partition, rows are clustered by the remaining columns of the table's primary key
- The main objects of the CQL data model are:
  - Keyspaces,
  - Tables, logically representing column families, and
  - Columns
- Super columns and super column families are gone due to performance issues

Keyspace

- A ***keyspace*** is a named data container of an application
- A keyspace contains a set of table (column family) declarations
  - In relational terms, a keyspace is analogous to a database schema
- In Cassandra, keyspaces are used to group column families having the same replication requirements
- The keyspace declaration is discussed in more detail in lecture: Cassandra Architecture

Tables

- A ***table*** is a set of rows containing data
  - Each row has a unique row key and a set of columns
  - A table is a map (row key→set of columns)
  - A table is a logical unit of data and a unit of data access to a Cassandra database
- There is no relationship between tables maintained by Cassandra CDBMS
  - There are no formal foreign keys and referential integrity constraints
  - Hence, joins are not supported at query time

Column

- A ***column*** is a triplet (*column_name*, *value*, *timestamp*)
  - A column must have a name, and a column name is also called the column key,
  - Columns are nullable, but a row must have at least one not null column (the row key column),
  - The timestamp is provided by a client application and used by Cassandra to determine the most recent update to a column,
  - The triplet (*column_name*, *value*, *timestamp*) is also called a **cell**
- A column can have an optional expiration time called TTL (time to live)
  - TTL is defined by the client application inserting the column
  - After the requested amount of time has expired, the column is marked deleted by Cassandra

Cassandra Query Language CQL

- The CQL syntax is very close to SQL
  - DDL supports commands like:
    - CREATE, DROP, and ALTER KEYSPACE <keyspace_name>,
    - CREATE, DROP, and ALTER TABLE <table_name>,
    - CREATE and DROP INDEX
  - DML supports commands like:
    - INSERT row and DELETE column,
    - UPDATE column,
    - SELECT <column_name_list_1> FROM <table_name> WHERE <conditional_expression> ORDER BY <column_name_list_2> LIMIT <number>
      - JOIN is not supported
      - Nested queries are not supported
      - GROUP BY and HAVING with aggregate function are not supported

CQL Declaration of `blogs` KEYSPACE

```
cqlsh => CREATE KEYSPACE blogs
         WITH REPLICATION = { 'class' :
         'SimpleStrategy',
         'replication_factor' : 3 };
```

```
cqlsh => USE blogs;
```

```
cqlsh.blogs => CREATE TABLE users (
                user_name text PRIMARY KEY,
               name text,
               city text
               );
```

CQL Declaration of `blogs` KEYSPACE

```
cqlsh.blogs > CREATE TABLE blog_entries (
                user_name text,
                body text,
                no int,
                PRIMARY KEY (user_name, no)
                );
// user_name is the partition key, no is the
// clustering key

cqlsh.blogs > CREATE TABLE subscribed_to (
                user_name text PRIMARY KEY,
                subscribers set<text>
                );
```

Declaring the Consistency Level

- The consistency level can be declared in `cqlsh`

```
cqlsh:blogs> consistency;
Current consistency level is ONE.
```

```
cqlsh:blogs> consistency quorum;
Consistency level set to QUORUM.
```

```
// All subsequent statements will be
executed under the consistency level
quorum, until a new change of the
consistency level happens
```

Populating blogs KEYSPACE

```
cqlsh.blogs > INSERT INTO users (user_name, name,
              city) VALUES ('jbond', 'James',
              'London');
```

```
cqlsh.blogs > INSERT INTO blog_entries (user_name,
              body, no) VALUES ('jbond', 'Today I
              ...', 1);
```

```
cqlsh.blogs > INSERT INTO subscribes_to (user_name,
              subscribers) VALUES ('jbond',
              {'asmith'});
```

```
cqlsh.blogs > UPDATE subscribes_to SET subscribers
              = subscribers + {'canslow'} WHERE
              user_name = 'jbond';
```

blog KEYSPACE

users

row key		
user_name	name	city
jbond	James	London
asmith	Ann	Napier
canslow	Craig	

blog_entries

row key		
user_name	no	body
jbond	1	Well, ...
jbond	2	Today, ...
jbond	3	Lectures ...
asmith	1	I am...
canslow	1	This is...

subscribed_to

row key	
user_name	subscribers
jbond	{canslow, asmith}
asmith	{jbond}

CQL CREATE TABLE (1)

- The CQL basic table declaration is very close to SQL DDL table declaration, but, there is also a number of differences

```
CREATE TABLE IF NOT EXISTS
keyspace_name.table_name (
column_definition, column_definition,
..., primary key (column_name,…)) WITH
property AND property ...
```

- Columns can be of:
  - Basic cql_type,
  - Tuple type,
  - User_defined type,
  - Collection type, and
  - Counter type
  - A static column is used to store the same data in multiple clustered rows of a partition, and then retrieved with a single SELECT

CQL Primary Key (2)

- The compulsory primary key can be defined either in-line, or as a table key
- The table key:

```
column_name |
(column_name1, column_name2,...) |
((column_name3, column_name4),
column_name5, column_name6,...)
```

  - Partition key columns are in normal text and **red**, clustering key columns are in italic and **blue**
- CQL tables have a number of properties like:
  - Clustering order, and
  - Compaction strategy
  - These properties influence table physical features and are discussed in more details in lecture: Cassandra Storage Engine

Partition and Clustering Key Example

- Consider the `blog_entries` table with the key `(user_name, no)`
  - The column `user_name` is the partition key, and
  - The column `no` is the clustering key
- Assume the partitioner hashes:
  - `jbond` and `asmith` to tokens of node1 and
  - `canslow` to a token of node3

node1		
jbond	1	*Well, ...*
jbond	2	*Today, ...*
jbond	3	*Lectures ...*
asmith	1	*I am...*
asmith	2	*This is...*

node3		
canslow	1	*Again, ...*
canslow	2	*Yesterday, ...*
canslow	3	*Annoyingly ...*
canslow	4	*Hmm …*

CQL Built-In Data Types

- CQL data types:
  - `blob`,
  - `ascii (char string)`,
  - `text`,
  - `varchar`,
  - `boolean`,
  - `float`,
  - `int`,
  - `decimal`,
  - `uuid()`,
  - `timestamp`,
  - `counter`,
  - `collection (set, list, map)`,
  - `inet (`IP address string in IPv4 or IPv6 format`)`, and
  - others

`uuid` and `timeuuid` Data Type

- The **universally unique identifier** (`uuid`) and `timeuuid` data types are used to avoid collisions in column names and for table primary key values
- A part of a `timeuuid` value contains information about time that can be converted into a `timestamp` value
- For human-readable display of a `timeuuid` value, a hexadecimal text with inserted hyphen characters is used:

  `de305d54-75b4-431b-adb2-eb6b9e546014`

- The word unique should be taken to mean "practically unique“

`timestamp` Type

- Values for the `timestamp` type are encoded as 64-bit signed integers representing date plus time
- A timestamp type can be entered as an integer for CQL input, or as a string literal in many formats
- We use the following format:

  `yyyy-mm-dd HH:mm:ssZ`

  where `Z` is the RFC-822 4-digit time zone, expressing the time zone's difference from UTC
  - For new Zealand, `Z = 1300` in March
- We also use just the date part of a `timestamp`

  `yyyy-mm-dd`
- Examples:

  `2016-03-26 19:31:20+1300`

  `2016-03-26`

Collection Data Type

- A collection column is declared using the collection type, followed by another `data_type`, such as `int` or `text`, in angle brackets
  - Collections should be small enough to be manageable
- There are three subtypes of the collection type:
  - `list<data_type>`,
  - `set<data_type>`, and
  - `map<data_type, data_type>`
- Each element of a set, list, or map is internally stored as one Cassandra column
- The collection columns are updatable
  - The individual elements can be added or removed

The List Data Type

- The list data type is used when:
  - The order of elements matters, or
  - Same value needs to be stored multiple times
  - List values are returned according to their index value in the list, whereas set values are returned in alphabetical order, assuming the values are text

```
CREATE TABLE best_students (
class text PRIMARY KEY, students
LIST<text>);
```

- The `UPDATE` command inserts values into the list

```
UPDATE best_students SET students =
['James', 'Craig' ] WHERE class =
'SWEN432';
```

More Operations on The List Data Type

- Prepend:

```
UPDATE best_students SET students =
['Ann'] + students WHERE class =
'SWEN432';
```

- Append:

```
UPDATE best_students SET students =
students + ['Ann'] WHERE class =
'SWEN432';
```

- Appending and prepending a new element to the list writes only the new element without any read-before-write

Even More Operations on List Data Type

- Add (or delete) an element at a particular position:

```
UPDATE best_students SET students[2] =
'George' WHERE class = 'SWEN432';
```

```
DELETE students[2] FROM best_students
WHERE class = 'SWEN432';
```

  - Here Cassandra reads the entire list, updates it and then writes it back
  - Consequently, adding an element at a particular position results in greater latency than appending or prefixing an element to a list
  - Also, it is transactionally not safe, can overwrite other people's job

```
UPDATE best_students SET students =
students - ['George'] WHERE class =
'SWEN432';
```

  - Deletes all `'George'` entries,
  - Doesn't read

The Map Data Type

- A map is a named pair of typed values
- Each element of the map is internally stored as one Cassandra column that can be modified, replaced, deleted, and queried
  - Also, each element can have an individual time-to-live and expires when the TTL ends

```
ALTER TABLE subscribed_to DROP
subscribers);
```

```
ALTER TABLE subscribed_to ADD
subscribers map<text: text>);
```

```
UPDATE subscribed_to SET subscribers = {
'asmith' : 'Napier', 'canslow' : null}
WHERE user_name = 'jbond';
```

More Operations on The Map Data Type

- To add a new element to a map:

```
UPDATE subscribed_to SET subscribers =
subscribers + {‘pmogin’: ‘Upper Hutt’}
WHERE user_name = ‘jbond';
```

- A specific map element can be updated, or deleted from the map

```
UPDATE subscribed_to SET
subscribers[‘pmogin'] = ‘Wellington'
WHERE user_name = ‘jbond';
```

```
DELETE subscribers[‘pmogin'] FROM
subscribed_to WHERE user_name = ‘jbond';
```

`counter` Data Type

- A counter column value is a 64-bit signed integer
  - The value of a counter can't be set, it can be incremented and decremented only, using UPDATE statements
- The use of counter data types has a number of restrictions:
  - A column of the counter type can't serve as the primary key or partition key
  - A table containing counter type columns can't contain anything other than counter types and the primary key
  - A counter column can't be used to create an index
  - Data in a counter column can't be set to expire using the Time-To-Live (TTL) property.

Expiring Data

- Data in a column, other than a counter column, can have an optional expiration period called TTL (time to live)
- The client request specifies a TTL value, defined in seconds, for the data
- TTL data is marked with a tombstone (deletion marker) after the requested amount of time has expired
- A tombstone exists for `gc_grace_seconds`
  - Specifies the time to wait before garbage collecting tombstones (10 days by default)
- After data is marked with a tombstone, the data is automatically removed during a normal compaction and repair processes

Expiring Data

- To change the TTL of expiring data, one has to re-insert the data with a new TTL
- TTL data has a precision of one second
  - A very small TTL probably does not make much sense
- Expiring data introduces an additional overhead of 8 bytes in memory and on disk

```
INSERT INTO users (user_name, name, city)
VALUES ( 'pmogin', 'Pavle', 'Upper Hutt')
USING TTL 86400;
```

```
SELECT TTL(city) from users
WHERE user_name = 'Pavle';

ttl(city)
-----------
85908
```

Static Column

- A table column may have the optional property `static`
- In a table that uses clustering columns, columns that do not belong to the primary key can be declared `static` in the table definition
- A static column is a special column whose value is shared by all rows of the same partition key value
  - The `DISTINCT` keyword can be used in a query to select static columns
    - In this case, Cassandra retrieves only the beginning (static column) of the partition

Summary

- According to CQL the Cassandra Data Model is described in terms of:
  - Keyspaces,
  - Tables (column families) containing rows of columns that are (name, value, timestamp) triplets,
  - A column may be of the collection type (set, list, or map),
    - Collection type columns are used to de normalize tables
- Tables of a Cassandra database are defined to satisfy specific queries in an efficient way by intentionally introducing data redundancy and by exploiting knowledge about the database physical structure