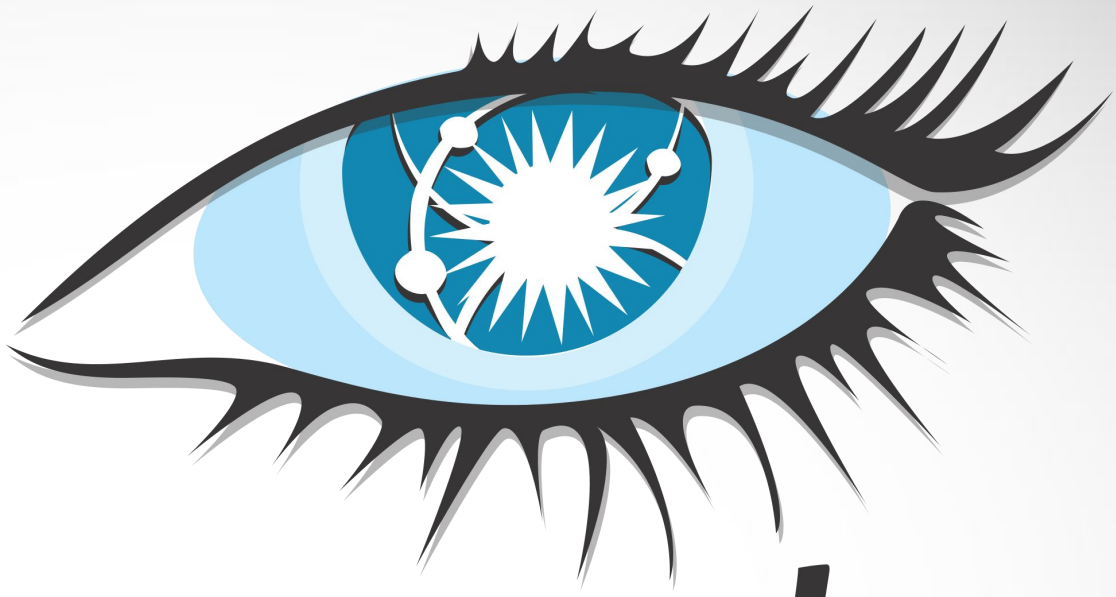


cassandra

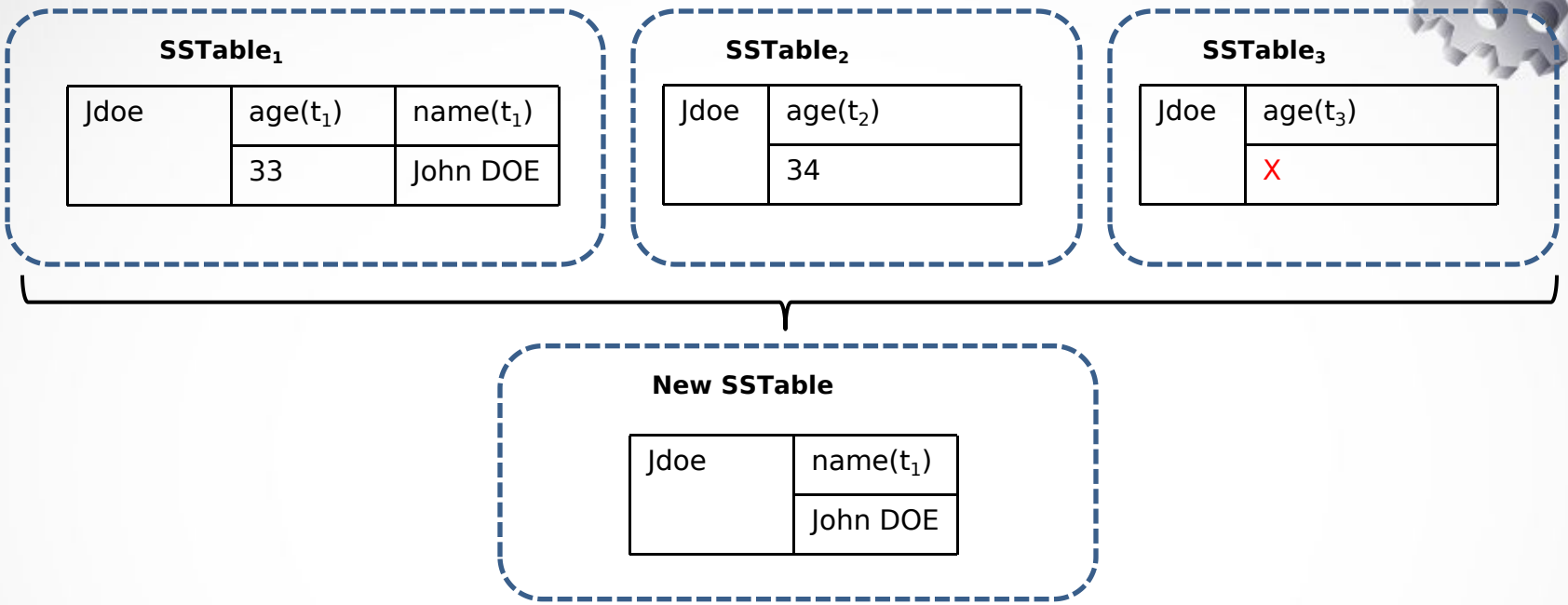
By
Nirmallya Mukherjee

Certified Cassandra Developer
Certified Cassandra Administrator

Part 5

Write and Read path

Compaction



- Related SSTables are merged
- Most recent version of each column is compiled to one partition in one new SSTable
- Columns marked for eviction/deletion & tombstones older than gc_grace_seconds are removed
- Old generation SSTables are deleted
- A new generation SSTable is created (hence the generation number keep increasing over time)

Disk space freed = sizeof(SST1 + SST2 + .. + SSTn) - sizeof(new compact SST)
Commit logs (containing segments) are versioned and reused as well
Newly freed space becomes available for reuse

Compaction

- Three strategies
  - Sizetiered (default)
  - Leveled (needs about 50% more IO than size tiered but the number of SSTables visited for data will be less)
  - DateTiered
- Choice depends on the disk
  - Mechanical disk = Size tiered
  - SSD = Leveled tiered
- Choice depends on use case too
  - Size - It is best to use this strategy when you have insert-heavy, read-light workloads
  - Level - It is best suited for read-heavy workloads that have frequent updates to existing rows
  - Date - for timeseries data along with TTL
- As per Datastax documentation
  - Cassandra 2.1 improves read performance after compaction by performing an incremental replacement of compacted SSTables.
  - Instead of waiting for the entire compaction to finish and then throwing away the old SSTable (and cache), Cassandra can read data directly from the new SSTable even before it finishes writing.
- Can be initiated by nodetool compact BUT this is an antipattern because it is also called as a "Major compaction" and clumps all related SST into a single SST which can become massive!
- Revisiting disk space calculation (http://docs.datastax.com/en//cassandra/2.0/cassandra/architecture/architecturePlanningDiskCapacity_t.html)

NTP is very important

- A micro second time stamp is associated with the columns
- The timestamp used is of the machine in which C* is running (coordinator)
- If the timing of the machines in the cluster is off then C* will be unable to determine which record is latest!
- This also means that the compaction may not function
- You can have a scenario where a deleted record appears again or get unpredictable number of records for your query

Part 6

Modeling I

Modeling guidelines

- Don't think SQL ERD
  - Unlike RDBMS you cannot use any foreign key relationships
  - FK and joins are not supported anyway!
- Know your queries (start with data access)
- Denormalize ("join on write"✔ vs "join on read")
  - Embed details - it will cause duplication but that is alright. Clustering col, UDT, Collections are great ways
  - Duplicate data - may have to create more than one denormalized table to serve different queries
- Please! Please! Please! remember
  - You can easily buy hardware/storage BUT you cannot buy your user's patience ☺

Counters - a bit more

- How do we increment a "counter" in a distributed system?
- A "global lock" is not feasible for many reasons!
- Divide and rule - each node maintains the counter value
- First each node gets a local lock
- Each node increments its value AFTER a read of current value (yes ... read before write!)
- Value of the counter = that of the latest write
- It is stored just like any other table using memtable/sstable

- Implementation is in the class CounterMutation.java (public Mutation apply() also see private void updateWithCurrentValue(...))
- Also look at the information on Stripe here (http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/util/concurrent/Striped.html)

- More details are here (http://www.datastax.com/dev/blog/whats-new-in-cassandra-2-1-a-better-implementation-of-counters)

Choice of primary key

- If a key is not unique then C* will upsert with no warning!
- Good choices
  - Natural key like an email address or meter id
  - Surrogate keys like uuid, timeuuid (uuid with time component)
- Surrogate keys offers greater control/flexibility at the time of row splitting (avoid wide row, up next...)
- CQL provides a number of timeuuid functions
  - now()
  - dateof(tuuid) - extracts the timestamp as date
  - unixtimestampof - raw 64 bit integer
- A classical primary key ensures
  - Uniqueness & Minimality
  - We have columns required to precicely identify a record
- In C* we relax the minimality constraint and include columns that are not necessarily required to uniquely identify a record
  - This is required to service a query filter or simply a query!

Wide row? Use composite PK

Only event type as the PK will store all events ever generated in a single partition. That can result in a huge partition and may breach C* limits?

```
create keyspace eventdb with replication = { 'class':'SimpleStrategy', 'replication_factor': 1 };
use eventdb;
CREATE TABLE all_events (
  event_type int,
  date int,       //format as yyyymmdd
  created_hh int,
  created_min int,
  created_sec int,
  created_nn int,
  data text,
  PRIMARY KEY((event_type, date), created_hh)              //Does this create a wide row? OR the below primary key?
OR
  PRIMARY KEY((event_type), date, created_hh, created_min, created_nn)
);
```

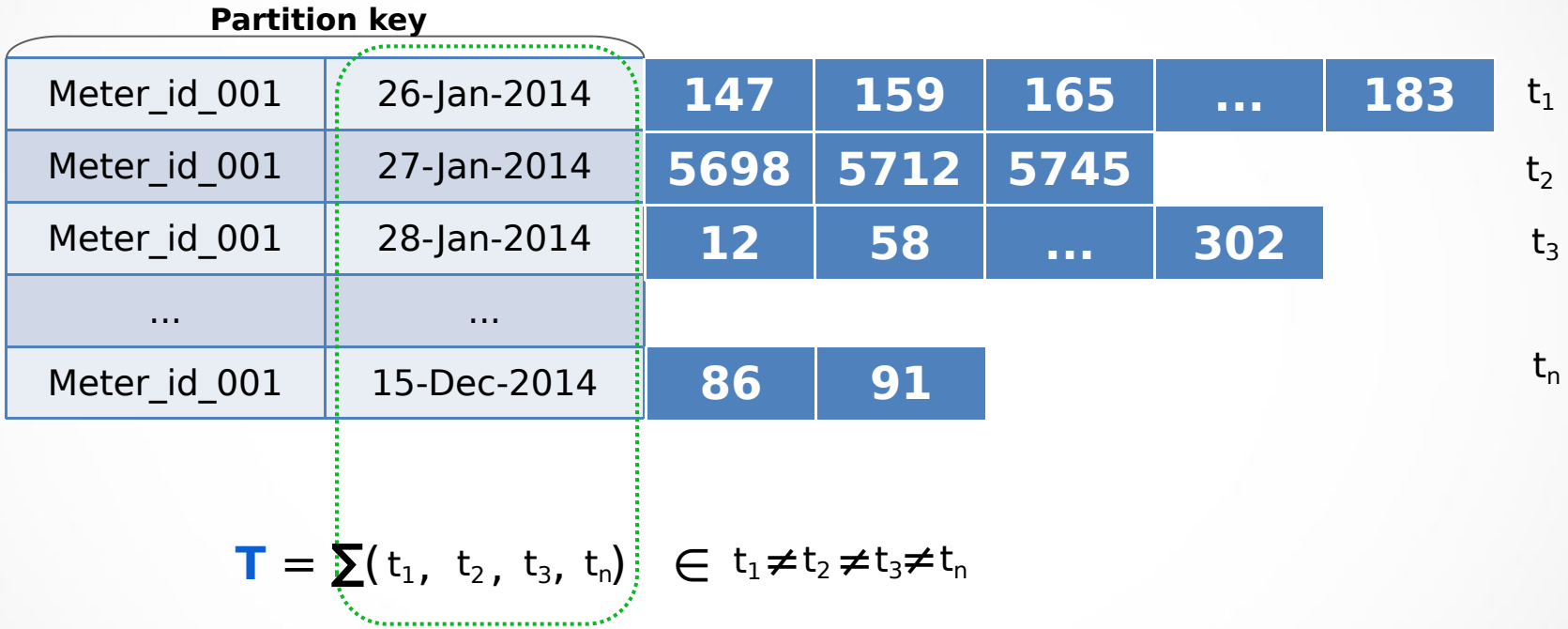
Example of a compound PK ((...), ...) syntax

```
CREATE TABLE trunc_events (
  event_type int,
  date int,
  created_on timestamp,
  data text,
  PRIMARY KEY((event_type, date), created_on)
) WITH CLUSTERING ORDER BY (created_on DESC);

insert into trunc_events (event_type, date, created_on, data) values(1, 20151014, '2015-10-14 12:47:54',
'{"event" : "details of the event will go here"}') USING TTL 300;
```

PK IN clause debate

where meter_id = 'Meter_id_001' and date IN (...)



Is T predictable? What are the factors responsible for determining the overall fetch time T?

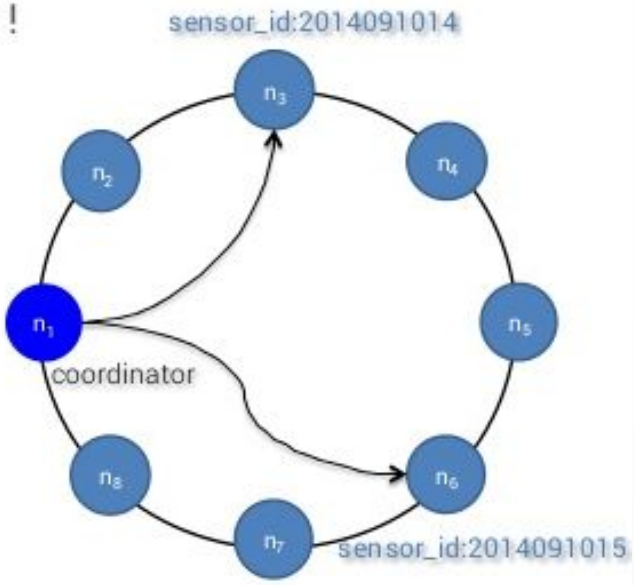
This is also called as "Multi get slice"

PK IN clause (Slicing)

If you want to search across two partition keys then use "IN" in the where clause. You can have a further limiting condition based on cluster columns.

IN clause for #partition is not silver bullet !

- use scarcely
- keep cardinality **low** (≤ 5)



Older then Version 2.2, in a composite PK, IN works only on the last key. Eg Pk ((key1, key2), clust1, clust2). IN will work only on key2 (last column only).

Counters

```
CREATE KEYSPACE IF NOT EXISTS counterks WITH replication =
        {'class':'NetworkTopologyStrategy', 'dc1' : 1};

use counterks;  //See class CounterMutation.java for more details

CREATE TABLE event_counter (
  event_type int,
  event_name varchar,
  counter_value counter,       //You can have more than 1 counter column
  PRIMARY KEY (event_type, event_name)
);

update event_counter
  set counter_value = counter_value + 1
  WHERE event_type=0 AND event_name='Login';
```

UDT mapping to a JSON

```
{
   "productId": 2,
   "name": "Kitchen Table",
   "price": 249.99,
   "description" : "Rectangular table with oak finish",
   "dimensions": {
      "units": "inches",
      "length": 50.0,
      "width": 66.0,
      "height": 32
   },
   "categories": {
      {
         "category" : "Home Furnishings" {
            "catalogPage": 45,
            "url": "/home/furnishings"
         },
         {
         "category" : "Kitchen Furnishings" {
            "catalogPage": 108,
            "url": "/kitchen/furnishings"
      }
   }
}
```

```
CREATE TYPE dimensions (
   units text,
   length float,
   width float,
   height float
);


CREATE TYPE category (
   catalogPage int,
   url text
);

CREATE TABLE product (
   productId int,
   name text,
   price float,
   description text,
   dimensions frozen <dimensions>,
   categories map <text, frozen <category>>,
   PRIMARY KEY (productId)
);
```

```
INSERT INTO product (productId, name, price, description, dimensions, categories)
VALUES (2, 'Kitchen Table', 249.99, 'Rectangular table with oak finish',
   {
     units: 'inches',
     length: 50.0,
     width: 66.0,
     height: 32
   },
   {
     'Home Furnishings': {
       catalogPage: 45,
       url: '/home/furnishings'
     },
     'Kitchen Furnishings': {
       catalogPage: 108,
       url: '/kitchen/furnishings'
     }

   }
);
```

→ dimensions frozen <dimensions>

→ categories map <text, frozen <category>>

Collections mapping to a JSON

```
CREATE TABLE example (
    id int PRIMARY KEY,
    tupleval tuple<int, text>,
    numbers set<int>,
    words list<text>
);

INSERT INTO example (id, tupleval, numbers, words)
VALUES (0, (1, 'foo'), {1, 2, 3, 6}, ['the', 'quick', 'brown', 'fox']);

INSERT INTO example JSON
'{"id": 0,
  "tupleval": [1, "foo"],
  "numbers": [1, 2, 3, 6],
  "words": ["the", "quick", "brown", "fox"]}';
```

Debate - visit this link
http://www.datastax.com/dev/blog/whats-new-in-cassandra-2-2-json-support

and analyze this statement "Columns which are omitted from the JSON value map are treated as a null insert (which results in an existing value being deleted, if one is present)"

Q: What are your observations?

TTL - Time To Live

```
INSERT INTO stockdb.user (user_id, display_name, first_name, last_name)
  VALUES ('u1', 'nm', 'Nirmallya', 'Mukherjee') USING TTL <computed_ttl>;

CREATE TABLE todo (
  id int PRIMARY KEY,
  todo map<text, text>
);

UPDATE users USING TTL <computed_ttl>
  SET todo['Item 1'] = 'First toto item'
WHERE id = 1;
```

Rate limiting is a good idea in many cases. Eg allow password reset to only 3 per day per user. Use TTL sliding window.

A promotion is generated for a customer with a TTL.

A temporary login is generated and is valid for a given TTL period.

One time password (OTP)

Can check how much time is remaining for a given column. See this http://docs.datastax.com/en/cql/3.1/cql/cql_using/use_ttl_t.html

UDF & UDA

- Where does these execute in the cluster?
- Let's evaluate the core aspect by looking at a sample
- Languages supported
  - Java
  - Javascript
  - Others by virtue of including dependencies

Materialized views

- A materialized view is a table that is built from another table's data with a new primary key
- Materialized views update and delete values when the original table is updated and deleted
- Secondary indexes which are suited for low cardinality data, materialized views are suited for high cardinality data
- Certain requirements must be met
  - The columns of the original table's primary key must be part of the materialized view's primary key
  - Only one new column may be added to the materialized view's primary key
- A few performance tips
  - Not the same performance as that of a normal table
  - Built asynchronously after data is inserted into the base table, and can result in delays in updating data
  - Data cannot be written directly into a materialized view
  - Read repair is done to base tables, resulting in materialized view repair

Cassandra.isEvolving().giveMore()

- C* is constantly and rapidly evolving and new features are an every week thing
- Not a good idea to compress all that good work into a few slides ☺

- Let's look at these ...
  - Tools summary - http://docs.datastax.com/en/cassandra/3.x/cassandra/tools/toolsTOC.html
  - CQL capabilities - http://docs.datastax.com/en/cql/3.3/cql/cqlIntro.html
  - CQL syntax Overview - http://cassandra.apache.org/doc/cql3/CQL.html
  - JSON Support - http://www.datastax.com/dev/blog/whats-new-in-cassandra-2-2-json-support

Let's analyse these queries

```
CREATE TABLE mailbox (
login text,
message_id timeuuid,
subject text,
message text,
PRIMARY KEY((login), message_id);
```

Get message by user and message_id (date)

```
SELECT * FROM mailbox
WHERE login=jdoe
and message_id ='2014-07-12 16:00:00';
```

Get message by user and range of messages

```
SELECT * FROM mailbox
WHERE login=jdoe
and message_id <='2014-07-12 16:00:00'
and message_id >='2014-01-12 16:00:00';
```

Debate - are these correct?

Get message by message_id

```
SELECT * FROM mailbox WHERE message_id ='2014-07-12 16:00:00';
```

✗

Get message by date interval

```
SELECT * FROM mailbox WHERE
message_id <='2014-07-12 16:00:00'
and message_id >='2014-01-12 16:00:00';
```

✗

Debate - are these correct?

Get message by user range (range query on #partition)

```
SELCT * FROM mailbox WHERE login >= hsue and login <= jdoe;
```

✗

Get message by user pattern (not exact match on #partition)

```
SELCT * FROM mailbox WHERE login like '%doe%';
```

✗

WHERE clause & restrictions

- Deeper look at WHERE http://www.datastax.com/dev/blog/a-deep-look-to-the-cql-where-clause
- All queries (INSERT/UPDATE/DELETE/SELECT) must provide #partition (where can filter in contiguous data)
- WHERE clause is only possible on columns defined in primary key unless there is a secondary index available for that column
- "Allow filtering" can override the default behaviour of cross partition access but it is not a good practice at all (incorrect data model)
  - select .. from meter_reading where record_date > '2014-12-15'

  (assuming there is a secondary index on record_date)
- Exact match (=) or IN predicate on any partition key is allowed, range queries (<,<=,<=,>) not allowed (C* 2.2+)
  - Prior to C* 2.2, in case of a compound partition key IN is allowed in the last column of the compound key (if partition key has one only column then it works on that column)
- On clustering columns, exact match and range query predicates (<,<=,<=,>, IN) are allowed

  Consider clustering columns col1, col2, col3

  col1=1 and col2>3 and col3=4 //col3 cannot be restricted because the previous uses non eq restriction

  col1=1 and col2>3 and col3>4 //col3 cannot be restricted because the previous uses non eq restriction

  col1=1 and col2 in (3,5) and col3=4 //OK

  col1=1 and col2 in (3,5) and col3>4 //OK
- Order of the clustering filters must match the order of primary key definition otherwise create secondary index (anti-pattern)

Order by restrictions

- If the primary key is (industry_id, exchange_id, stock_symbol) then the following sort orders are valid
  - order by exchange_id desc, stock_symbol desc
  - order by exchange_id asc, stock_symbol asc

- Following is an example of invalid sort order
  - order by exchange_id desc, stock_symbol asc

Alter table ...

- Alter table is usually good for adding columns that are not part of the primary key
- You cannot change the type of a clustering column as well as the columns for which there is an index
- The "from" and "to" datatype must be compatible eg ascii to text
- Partition key column data type can be changed (text to varchar) but cannot be dropped
- If there is a need for alter table then the data model was perhaps not optimized for the query anyway
- Read more here
  - http://docs.datastax.com/en/cql/3.1/cql/cql_reference/alter_table_r.html

Antipatterns - things to watch out!

- Singleton messages - add, consume, remove loop (queue)

- Update columns as nulls in CQL. Tombstone for each null

	age	name	geo_loc	mood	status
jdoe	33	John DOE	☒	☒	☒

- Intense update on a single column

sensor_id	value (t_1)
	45.0034

sensor_id	value (t_{13})
	47.4182

sensor_id	value (t_{36})
	48.0300

- Dynamic schema change (frequent changes) and topology change in production is not a good idea.

- Too many tables (500+) because each table will occupy approx 1MB of JVM space.

Part 7

Modeling II and development

Ex 1 - stock model

- Remember we have to work backwards from how we would like to retrieve the data

- Here are the queries
  - Users have stocks that they track
  - User's portfolio value should be current
  - Latest value for a given stock

Stock CQL model

create keyspace stockdbwith replication = { 'class':'SimpleStrategy', 'replication_factor': 1 };

```
CREATE TABLE stockdb.user(
  user_id int,
  display_name VARCHAR,
  first_name VARCHAR,
  last_name VARCHAR
  PRIMARY KEY (user_id));
```

```
CREATE TABLE exchange(
  exchange_id VARCHAR,
  exchange_name VARCHAR,
  currency_code VARCHAR,
  active BOOLEAN,
  PRIMARY KEY (exchange_id));
```

```
CREATE TABLE stockdb.userstocklist(
  stock_symbol VARCHAR,
  user_id int,
  display_name VARCHAR,
  holding int,
  PRIMARY KEY (user_id, stock_symbol);
```

```
CREATE TABLE industry(
  industry_id INT,
  industry_name VARCHAR,
  sector_id INT,
  PRIMARY KEY (industry_id));
```

```
CREATE TABLE stockdb.stockprice (
  stock_symbol VARCHAR,
  trade_date INT,
  trade_time INT,
  company_name VARCHAR,
  start_price DECIMAL,
  current_price DECIMAL,
  exchange_id VARCHAR,
  industry_id INT,
  PRIMARY KEY ((trade_date, stock_symbol)), trade_time)
  with clustering order by (trade_time desc);
```

```
CREATE TABLE sector(
  sector_id INT,
  sector_name VARCHAR,
  PRIMARY KEY (sector_id));
```

Q: How to get a list of stocks given an industry?

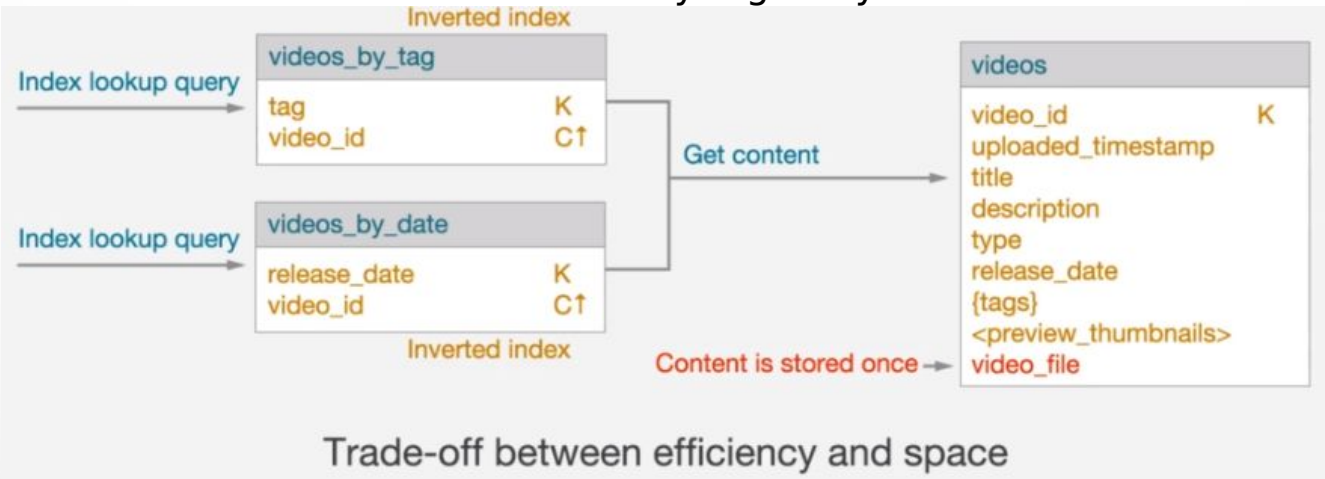
Ex 2 - Relationship/search table

Relationship tables are not uncommon to support different search/fetch criteria - get the list of stocks given an industry in this case from previous example it is not possible with the tables!

Solution is to create a new table StockSearch (like a search index) and reference to industry, exchange and stock tables - tradeoff!

```
CREATE TABLE StockSearch (
  industry_id INT,
  exchange_id VARCHAR,
  stock_symbol VARCHAR,
  PRIMARY KEY (industry_id, exchange_id, stock_symbol)
);
```

Model below shows how to access a video by tag or by release date



The video information can be embedded in the "inverted index" tables as well BUT that will take up lots of space. There needs to be a balance of denormalization vs space consumed!

Ex 3 - "Comment" Model

Consider a scenario where the user can give comments about a video like youtube. How to create a model if the queries are-

1. Get all comments for a given video (optionally filter by user)
2. Get all comments given by a user (optionally filter by video)

```
create table comments_by_video (
  video_id uuid,
  user_id varchar,
  comment_id timeuuid,
  comment text,
  primary key ((video_id), user_id, comment_id)
);
```

```
create table comments_by_user(
  user_id uuid,
  video_id varchar,
  comment_id timeuuid,
  comment text,
  primary key ((user_id), video_id, comment_id)
);
```

- Model both ways - create the same comment twice
- A good example of many to many relationship
- Totally ok to write multiple times!
- Do keep in mind the degree of de-normalization (notice the video and user details are not duplicated)
- Can read both ways - very efficient

Ex 4 - Video tag/genre/actor model

Q: Videos have actors, genre as well as tags. What model can help in fetching the videos based on any of these criteria?

Duplicate tables of the same data with just a different partition keys

videos_by_actor	
actor	K
release_date	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

videos_by_genre	
genre	K
release_date	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

videos_by_tag	
tag	K
release_date	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

A duplicate table with just a different sort order of the clustering column is also fine eg

PRIMARY KEY ((stock_symbol, trade_date), trade_time, exchange_id) can give sorted list based on time followed by exchange

PRIMARY KEY ((stock_symbol, trade_date), exchange_id, trade_time) can give sorted list based on exchange followed by time

Q: Are both queries below are same from a user's perspective, justify?

Select * from stock_tab1 where stock_symbol = 'IBM' and trade_date = 20150424

Select * from stock_tab2 where trade_date = 20150424 and stock_symbol = 'IBM'

Ex 5 - Bitmap type of index

- Multiple parts to a key
- Create a truth table of the various combinations
- However, inserts will increase to the number of combinations
- Eg Find a car (vehicle id) in a car park by variable combinations

Make	Model	Color	Combination
		x	Color
	x		Model
	x	x	Model+Color
x			Make
x		x	Make+Color
x	x		Make+Model
x	x	x	Make+Model+Color

Bitmap type of index

```
CREATE TABLE skldb.car_model_index (
   make varchar,
   model varchar,
   color varchar,
   vehicle_id int,
   lot_id int,
   PRIMARY KEY ((make, model, color), vehicle_id)
);
```

We are pre-optimizing for 7 possible queries of the index on insert.

```
INSERT INTO skldb.car_model_index (make, model, color, vehicle_id, lot_id)
  VALUES ('Ford', 'Mustang', 'Blue', 1234, 8675309);
```

```
INSERT INTO skldb.car_model_index (make, model, color, vehicle_id, lot_id)
  VALUES ('Ford', 'Mustang', '-', 1234, 8675309);
INSERT INTO skldb.car_model_index (make, model, color, vehicle_id, lot_id)
  VALUES ('Ford', '-', 'Blue', 1234, 8675309);
INSERT INTO skldb.car_model_index (make, model, color, vehicle_id, lot_id)
  VALUES ('Ford', '-', '-', 1234, 8675309);
INSERT INTO skldb.car_model_index (make, model, color, vehicle_id, lot_id)
  VALUES ('-', 'Mustang', 'Blue', 1234, 8675309);
INSERT INTO skldb.car_model_index (make, model, color, vehicle_id, lot_id)
  VALUES ('-', 'Mustang', '-', 1234, 8675309);
INSERT INTO skldb.car_model_index (make, model, color, vehicle_id, lot_id)
  VALUES ('-', '-', 'Blue', 1234, 8675309);
```

Bitmap type of index

```
select * from skldb.car_model_index
where make='Ford'
    and model='-'
    and color='Blue';
```

Results

make	model	color	vehicle_id	lot_id
Ford		Blue	1234	8675309

```
select * from skldb.car_model_index
where make='-'
    and model='-'
    and color='Blue';
```

Results

make	model	color	vehicle_id	lot_id
		Blue	1234	8675309

YES, there will be more writes and more data BUT that is an acceptable fact in big data modeling that looks to optimize the query and user experience more than data normalization

Ex 6 - Blob storage strategy

Q: Discuss a strategy for storing large binary data (an email attachment of a HD movie, assuming it ever gets there)☺

- A blob can be significantly large and can easily breach partition size limits
- Break a blob into chunks and store as independent partitions
- In the application fire concurrent queries to get all the chunks and assemble the blob before streaming
- Example below is that of an email attachment of an email application that uses C* as the backend
- Can have a table storing videos or large photos or documents! Type determines the mime type

Ex 7 - Case analysis

1. Consider a shopping aggregator / online eCommerce site. A model stores data (merchandise) for a specific apparel vendor (one partition per vendor). One such vendor announces a sale.

2. Migrating from RDBMS and tables have a sequence generator for keys. What's the strategy now?

3. Online movie streaming site. User can pause a movie anytime, watch another, pause that etc! Upon returning to a specific movie that was paused, need to resume from there. What's the model?

4. Build data for analytics which analyzes approximately how long after the start of the movie the bulk of pauses come from users.

5. Based on my shopping history (shopping trail can include searching products as well). Create a strategy to provide data to the recommendation engine.

6. IOT product, sensors are sending data. Need reports where I can analyze last 24hrs, last 7 days of data and yet not lose any data generated by these sensors. Model?

7. What is the meaning of "read modify write"? Analyze any use case that falls in this category and the impact of using C*.

Fetch on non PK (secondary index)

- Consider the earlier example of the table all_events, what will happen if we try to get the records based on minute?
  - Create a secondary index based on the minute
- Can be created on any column except counter and static
- It is actually another table with the key=<index column> and columns=keys that contain the key

- Do not use secondary indexes if
  - On high-cardinality columns because you then query a huge volume of records for a small number of results
  - In tables that use a counter column
  - On a frequently updated or deleted column (too many tombstones)
  - To look for a row in a large partition unless narrowly queried
  - High volume queries

- Secondary indexes are for searching convenience
  - Use with low-cardinality columns
  - Columns that may contain a relatively small set of distinct values
  - Use when prototyping, ad-hoc querying or with smaller datasets
- Secondary index lookup for UPDATE and DELETE statements is not and will not be supported due to the risk involved with read before write

- See this link for more details (http://www.datastax.com/documentation/cql/3.1/cql/ddl/ddl_when_use_index_c.html)

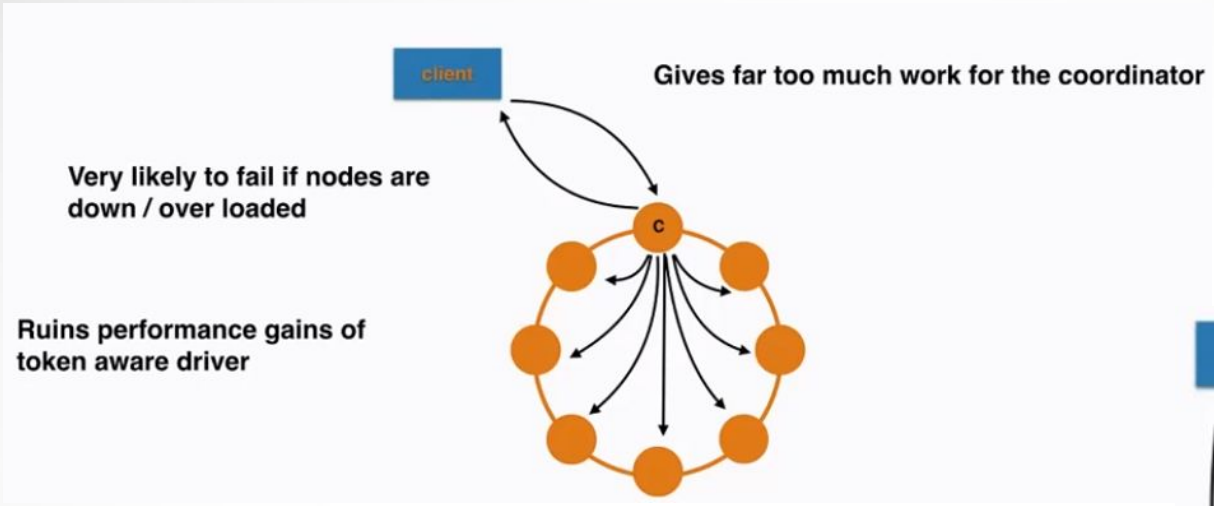
Secondary Index - new stuff!

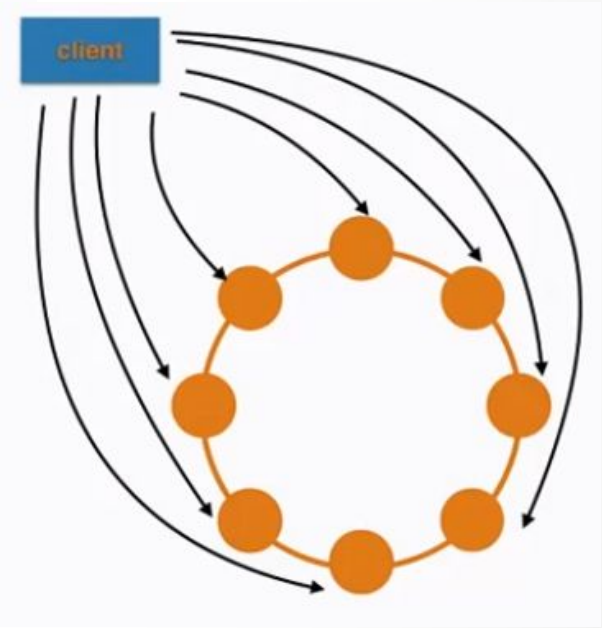
- Look at this https://github.com/xedin/sasi

- Build index B+ trees per SSTable
- Support like clause, OR clause, range queries
- This needs more mem because these are B+ trees to be loaded in mem
- They still have the same caviats of the existing index eg. cardinality (Scatter/Gather is an issue where we end up touching a lot of primary keys from across the cluster)

- Experiments
  - http://docs.datastax.com/en/cql/3.3/cql/cql_using/useSASIIndex.html
  - https://github.com/xedin/sasi

C* batch - Is it always a good idea?





Leverages token aware policies
Cluster load is evenly balanced
Means if one fails we re-try ONLY that query
Not an all or none success model

Tip: batches work well ONLY if all records have the same partition key
In this case all records go to the same node.

Also, the order of statements are not guaranteed.

Triggers

- Feature may still be experimental
- The actual trigger logic resides outside in a Java POJO
- Fired by the coordinator before the mutation
- Has write performance impact
- Better to have application logic do any pre-processing - at least for now!

```
CREATE TRIGGER myTrigger
  ON myTable
  USING 'Java class name';
```

Lightweight transactions

- Transactions are a bit different
  - "Compare and Set" model
- Two requests can agree on a single value creation
  - One will suceed (uses Paxos algo, wikipedia has good text on this subject)
  - Other will fail
  - There will be a bit of latency
- Also called as as "Linearizable consistency" and is internally implemented using Paxos algo

```
INSERT INTO customer_account (customerID, customer_email)
VALUES ('LauraS', 'lauras@gmail.com')
IF NOT EXISTS;
```

```
UPDATE customer_account
SET customer_email='laurass@gmail.com'
IF customerID='LauraS';
```

Cassandra 2.1.1 and later supports non-equal conditions for lightweight transactions. You can use <, <=, >, >=, != and IN operators in WHERE clauses to query lightweight tables.

Part 8

Administration

Nodetool repair - potential strategy

- Run a repair for the node's primary range
  - ./nodetool repair --partitioner-range
- OR the system.local table has all the tokens that the node is responsible for, so create a script that runs a repair using -st <token> -et <same token> in a loop for all tokens for the given node
  - This will allow the repair to run on small ranges and in short time increments

- Care should be taken to connect to the node using -h and use parallel execution (-par)
- In addition consider using DC (with name) or -local
- Spool the output and analyze if everything was correct else can send email or any form of notification to alert the admin
- It may be possible to get an exception like the following if the start token and end tokens are not the same
  - ERROR [Thread-1099288] 2015-03-14 02:48:08,822 StorageService.java (line 2517) Repair session failed: java.lang.IllegalArgumentException: Requested range intersects a local range but is not fully contained in one; this would lead to imprecise repair
  - Does not mean there is a problem

"Coming back to life"

- Let's assume multiple replicas exist for a given row/column
- One node was NOT able to recored a tombstone because it was down
- If this node remains down past the gc_grace_seconds without a repair then it will contain the old record
- Other nodes meanwhile have evicted the tombstone column (compaction OR nodetool repair will do the eviction once the tombstone > gc_grace_seconds)

- When the downed node does come up it will bring the old column back to life (will replicate) on the other nodes!

- To ensure that deleted data never resurfaces, make sure to run repair at least once every gc_grace_seconds, and never let a node stay down longer than this time period

Schema disagreements

- Perform schema changes one at a time, at a steady pace, and from the same node
- Do not make multiple schema changes at the same time
- If NTP is not in place it is possible that the schemas may not be in synch (usual problem in most cases)
- Check if the schema is in agreement http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dml_handle_schema_disagree_t.html
- ./nodetool describecluster

Secondary Index - rebuilding

- While Cassandra’s performance will always be best using partition-key lookups, secondary indexes provide a nice way to query data on small or offline datasets
- Occasionally, secondary indexes can get out-of-sync
- Unfortunately, there is no great way to verify secondary indexes, other than rebuilding them.
- You can verify by a query using indexes, and look for significant inconsistencies on different nodes
- Also note that each node keeps its own index (they are not distributed), so you’ll need to run this on each node
- Must run nodetool repair to ensure the index gets built on current data

```
nodetool rebuild_index my_keysapce my_column_family
```

Performance - MUST watch + read

https://academy.datastax.com/courses/ds210-operations-and-performance-tuning/cassandra-tuning-cassandra-tuning

https://academy.datastax.com/courses/ds210-operations-and-performance-tuning/cassandra-tuning-data-model-tuning

AWESOME! https://tobert.github.io/pages/als-cassandra-21-tuning-guide.html

https://software.intel.com/sites/default/files/Configuration_and_Deployment_Guide_for_Cassandra_on_IA.pdf (See section 3.2)

https://academy.datastax.com/demos/datastax-enterprise-deployment-guide-google-compute-engine

Nodetool tablehistograms - guidelines

- ./nodetool tablehistograms <keyspace> <table>

- Will tell how many SSTables were looked at to satisfy a read
  - With level compaction should ideally never go > 3
  - With size tiered compaction should ideally never go > 12
  - If the above are not in place then possibly compaction is falling behind

  check with ./nodetool compactionstats

  it should say "pending tasks: 0"

- Read and write letency (no n/w latency)
  - Write should not be > 150µs
  - Read should not be > 130µs (from mem)
  - SSD should be single digit µs

Nodetool tablestats (Read performance)

- nodetool [-h <IP> -p <PORT>] tablestats <keyspace>.<table>
  - reports read latency, write latency, SSTable count, space used etc
  - table specification is optional, it will dump all tables for the given keyspace

Nodetool tpstats (Thread pool stats)

- ./nodetool tpstats

- This metric is very important because of SEDA
- Ops center has the table for anything blocked per node in the cluster

- Important to look for
  - Dropped messages indicate that the replication message is being dropped in favor of user messages - means nodes are not able to keep up

- Nice article on interpreting the output (http://www.pythian.com/blog/guide-to-cassandra-thread-pools)

CQL tracing (Read performance)

- For testing CQL performance, tracing can be used, expand can be used to show records as they are
  - cqlsh> tracing on;
  - cqlsh> expand on;

- nodetool –h <host> settraceprobability 1/0
  - 1 enables tracing, 0 disables
  - Set the probability to 0.01 (1%) to collect data at a slow trickle
- Traces all queries for the node until it is disabled, whereas TRACING on in cqlsh only traces queries in that shell
- Turn on this function, check trace output, make sure to disable tracing when you're done!
- One trace equals almost 10 writes, so trace data can add up quickly
- Table system_traces.sessions holds trace data
- TRACING ON in cqlsh is essentially setting the probability to 1
- Useful for troubleshooting – collects data before a problem occurs
- http://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsSetTraceProbability.html

C* and network

- Network statistics R/W latency
  - ./nodetool proxyhistograms
- Network statistics / Repair statistics
  - ./nodetool netstats

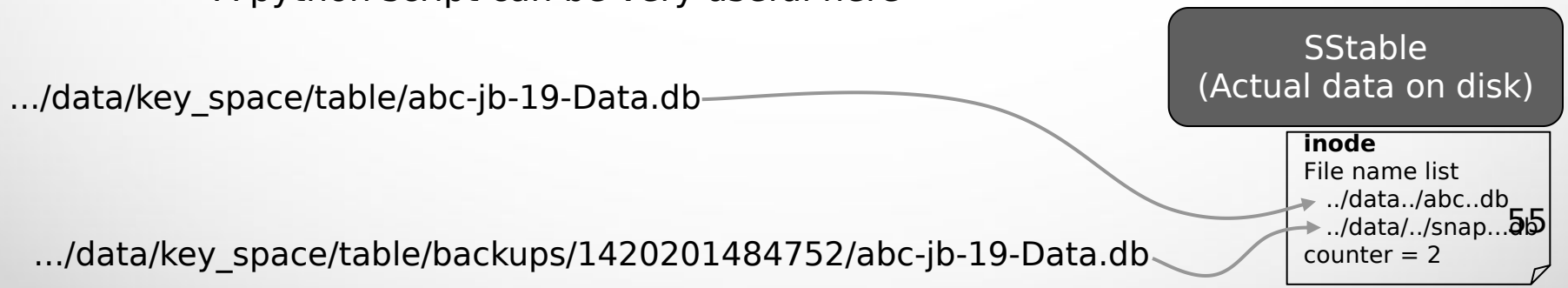
- C* allows for "Coalescing messages"
  - When communicating among nodes
  - Introduced in v2.1
  - You can configure coalescing from cassandra.yaml using the "otc_coalescing_strategy" and "otc_coalescing_window_us" (us = Mu/Micro seconds)
  - The default coalescing strategy in 2.2 is TIMEHORIZON

C* timeout, concurrency, consistency

- Cassandra.yaml has multiple timeouts (these change with versions)
  - read_request_timeout_in_ms
  - write_request_timeout_in_ms
  - cross_node_timeout
- Concurrency
  - concurrent_reads
  - concurrent_writes
- Application
  - Consistency levels affect read/write
- etc (see the yaml file)

Backup and recovery - Inc Backup

- Have incremental_backups set to true in C* yaml (default is false)
- The backup is created when a flush is performed and the newly created SSTable is hardlinked to the backup directory
  - Note: Hard links can only be created to files on the same volume. If a link to a file on a different volume is needed, it may be created with a symbolic link (symlink or soft link, it is a name and path mapping to the file). C* snapshots does not do softlinks.
  - If JNA is not enabled then the SSTable will be copied and disk IO will increase
- The older backups are not auto cleared so a cron needs to be put in place
  - Good idea to clean all backs when a snapshot is taken because the snapshot will include all data (snapshot next ...)
  - Remove everything else except for the latest timestamped directory in the backups folder
  - A python script can be very useful here



About me

Nirmallya Mukherjee

Architect at Independent

Bengaluru Area, India | Information Technology and Services

Current	Independent
Previous	SkroidsLab Technologies Pvt Ltd., Dell Services, Infosys Technologies Limited

www.linkedin.com/in/nirmallya

pointer.next@gmail.com

That's it!