Cassandra

An introduction to data modeling techniques
an evolution of enterprise architecture

Qualifications

- More than 15 years working on web enabled systems
- World wide project scope in manufacturing, logistics, and Social
- Lead the design, implementation, and deployment of a global test code control and distribution system
- Worked with Spring, Java EE, and Oracle DB

Qualificatons

- AWS ReInvent November 2014
- Cassandra data modeling course December 2014
- Cassandra systems admin course January 2015
- Collaborating with experts in the field
- Routinely performance test at over 40,000 requests per minute
- Cassandra and a micro-service architecture in prod.
- In prod have sustained over 180,000 requests per minute during initial data load

Agenda

- Cassandra and Data modeling
- Materialized views in Social
- Titan
- Architecture

Disclaimer

- This talk has had less than 1 week of advanced notice
- Question Everything

Our Journey

- Evolution to the web influences current thinking
  - Design bias
- Relational databases
  - ER diagram bias
- Social is a natural graph
- Relationships matter

Our Journey

- Neo4J
- Nike changes the game
- Cassandra, Couchbase
- Titan
- An emerging enterprise architecture

What is Cassandra

- NOSQL
- Open Source
- Distributed Data Management System
- Persistence

What is Cassandra

- NOSQL
- Open Source
- Distributed Data Management System
- Persistence
- Masterless Cluster

What is Cassandra

- NOSQL
- Open Source
- Distributed Data Management System
- Persistence
- Masterless Cluster
- Linear scalability

What is Cassandra

- NOSQL
- Open Source
- Distributed Data Management System
- Persistence
- Masterless Cluster
- Linear scalability
- Multiple Data Centers

Hybrid NOSQL Solution

- Hybrid between key value and a column store

Hybrid NOSQL Solution

- Hybrid between key value and a column store
- Column families, aka Table
  - Nothing like a a table in a relational model

Hybrid NOSQL Solution

- Hybrid between key value and a column store
- Column families, aka Table
  - Nothing like a a table in a relational model
  - Distributed multi-dimensional map, indexed by a partition key

Hybrid NOSQL Solution

- Hybrid between key value and a column store
- Column families, aka Table
  - Nothing like a a table in a relational model
  - Distributed multi-dimensional map, indexed by a partition key
- No Joins

Hybrid NOSQL Solution

- Hybrid between key value and a column store
- Column families, aka Table
  - Nothing like a a table in a relational model
  - Distributed multi-dimensional map, indexed by a partition key
- No Joins
- No Sub Query

Hybrid NOSQL Solution

- Hybrid between key value and a column store
- Column families, aka Table
  - Nothing like a a table in a relational model
  - Distributed multi-dimensional map, indexed by a partition key
- No Joins
- No Sub Query
- Does not do much, that’s why its fast and easy to learn

Think differently

- Relational we model the data

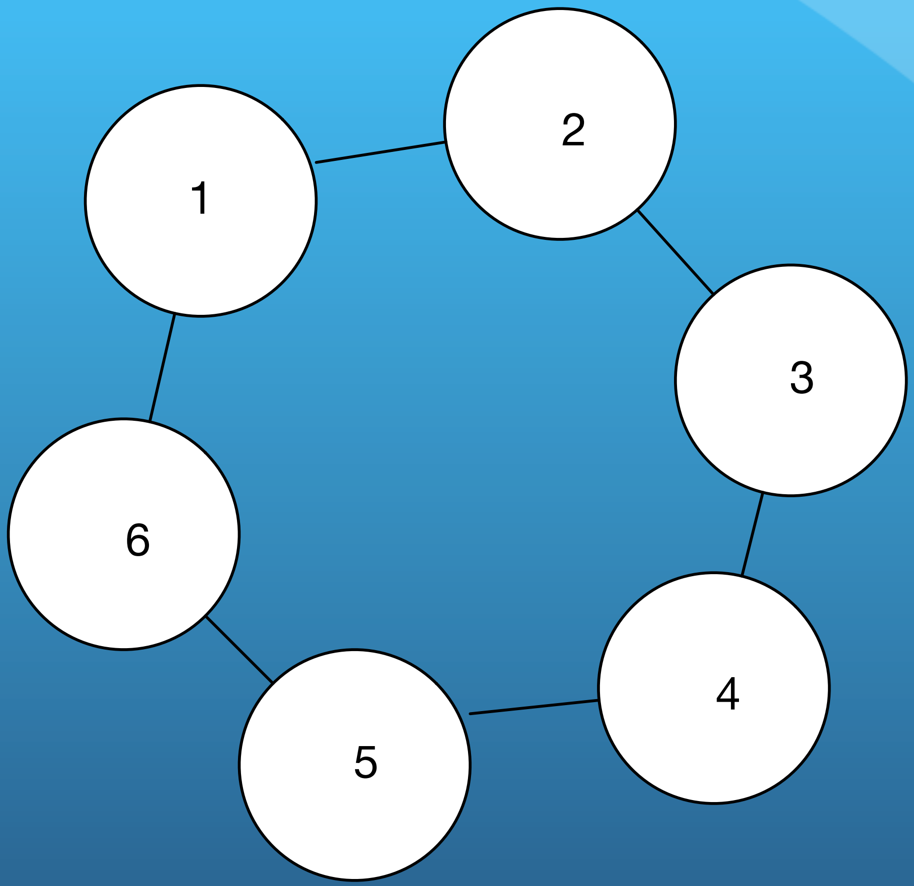
Think differently

- Relational we model the data
- Cassandra we model the query
  - Materialized Views

Think differently

- Relational we model the data
- Cassandra we model the query
  - Materialized Views
- The time to execute moves from runtime query to write time insert and update functions.

Cluster



Data Storage

- Data is not on every node
- Keyspace determines eventual consistency

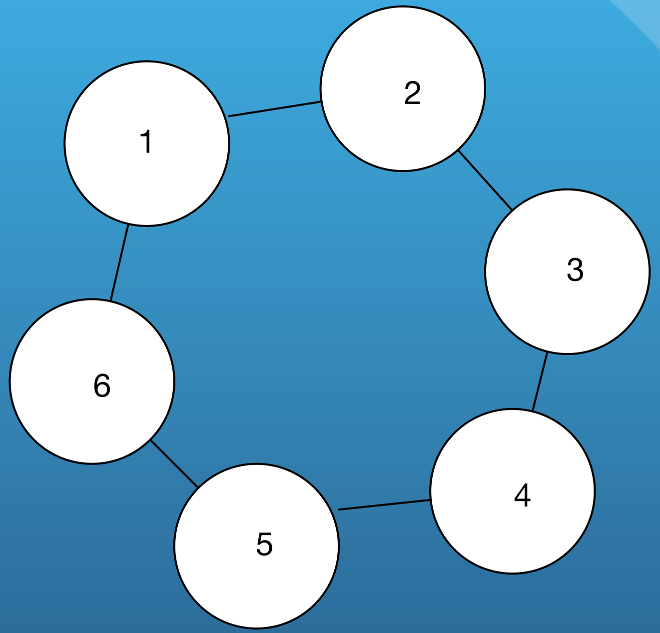
```
CREATE KEYSPACE IF NOT EXISTS social

WITH replication = {

 'class': 'NetworkTopologyStrategy',

  'us-west-2': '3'

};
```

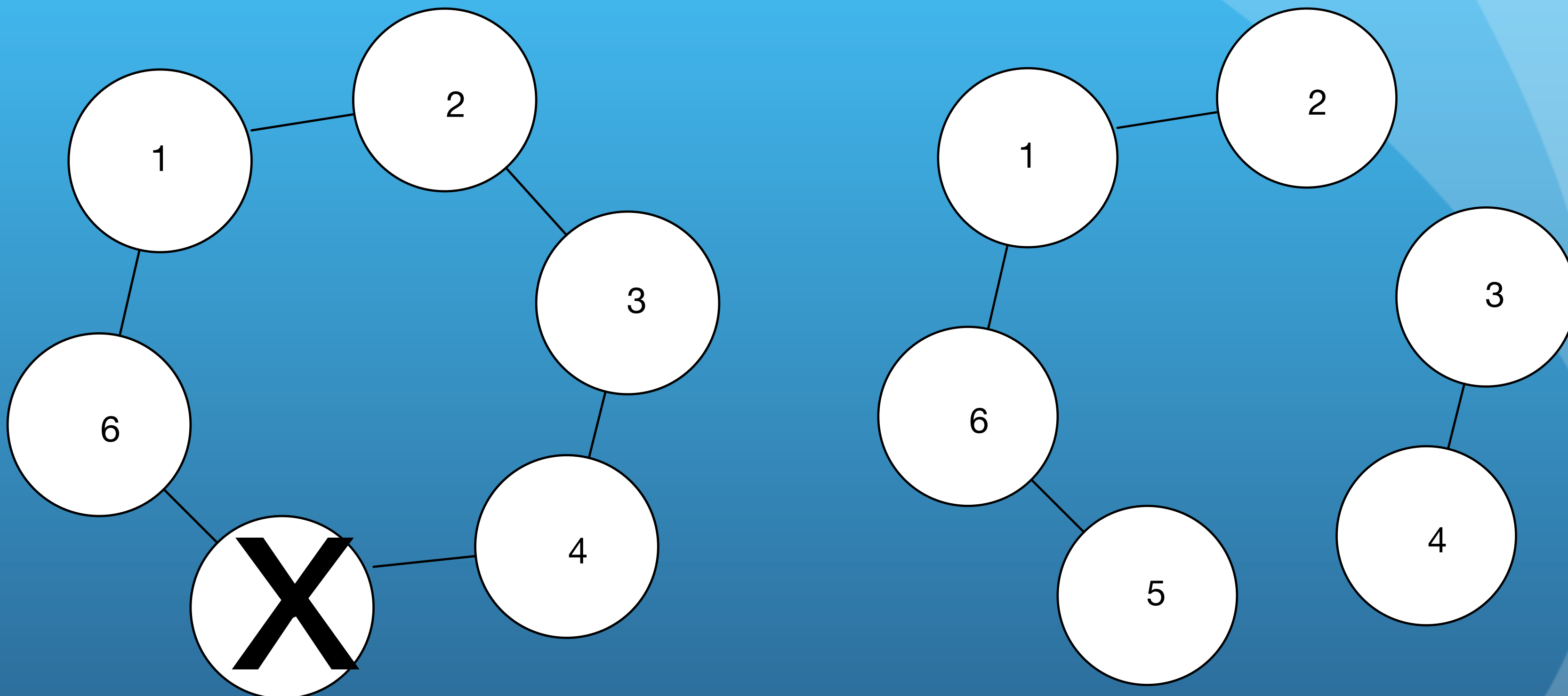


Keyspace

- Analogous to a relational schema
  - Collection of tables and indexes
- Distributed across data centers
- Defines the number of copies of data that should exist in a datacenter.

Failure Scenarios

Node down, partitioned Cluster



Differing Philosophies

- In a relational model we discuss ACID
  - Atomic
  - Consistent
  - Isolated
  - Durable
- In a distributed NOSQL system we discuss CAP Theorem

CAP Theorem

- In a distributed system we cannot achieve all of
  - Consistency
  - Availability
  - Partition Tolerance

CAP Theorem

- In a distributed system we cannot achieve all of
  - Consistency
    - Data is consistent on all <u>expected</u> storage nodes

CAP Theorem

- In a distributed system we cannot achieve all of
  - Consistency
    - Data is consistent on all expected storage nodes
  - Availability
    - Able to handle all data for all requests

CAP Theorem

- In a distributed system we cannot achieve all of
  - Consistency
    - Data is consistent on all <u>expected</u> storage nodes
  - Availability
    - Able to handle all data for all requests
  - Partition Tolerance
    - Tolerance to the cluster partitioning into separate units

Cassandra

Data is replicated

- Across nodes
- Perhaps across data centers

Eventual Consistency

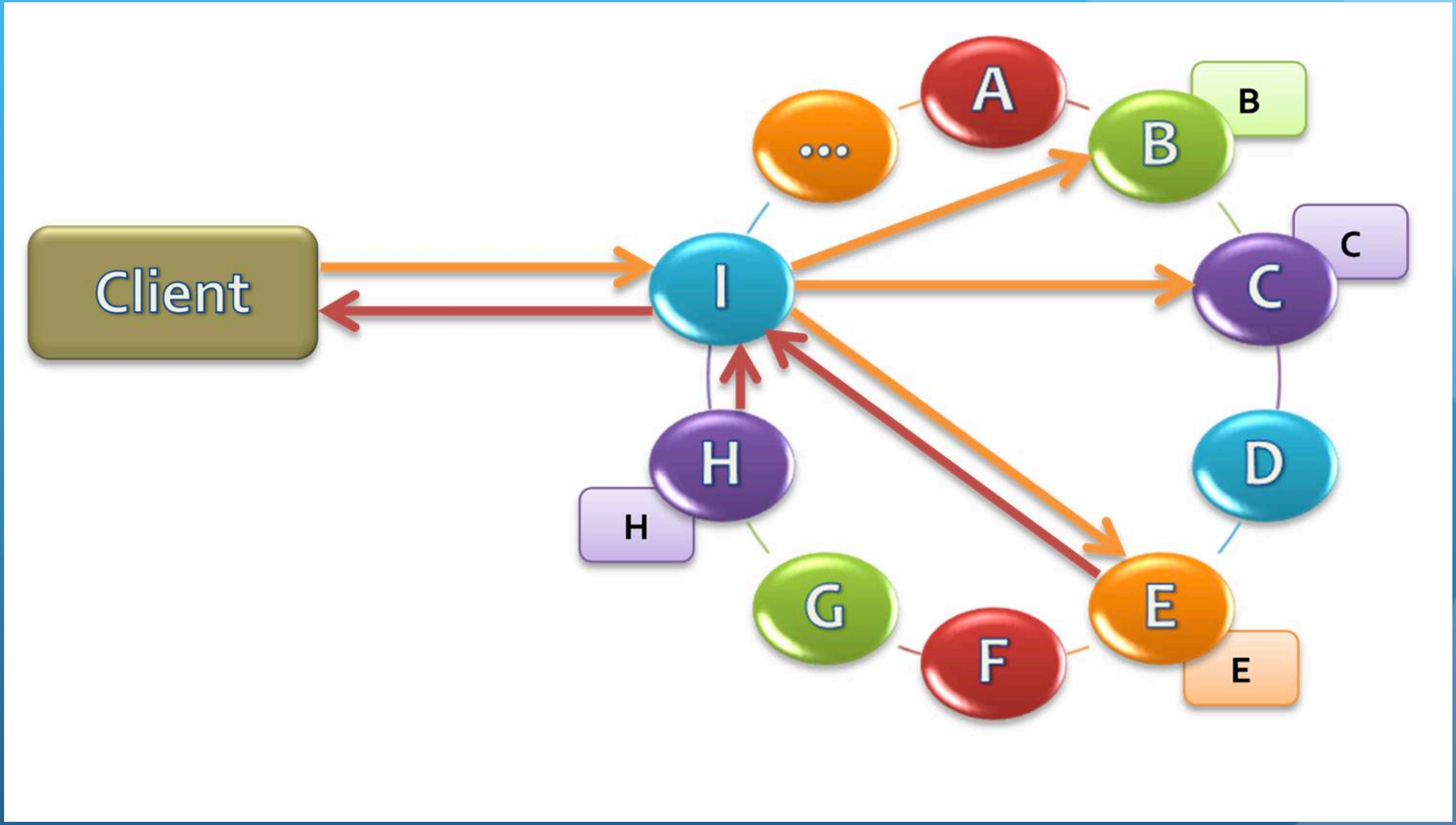
- At any given instant data may be inconsistent
- Eventual is milliseconds
- Last timestamp for write is the source of truth

Eventual Consistency

- At any given instant data may be inconsistent
  - Read and write policies
- Eventual is milliseconds
- Last timestamp for write is the source of truth
  - Think this through, real examples

Masterless

Every node acts as a coordinator



Basics

- Download and place Cassadra on the file system.
- start Cassandra: bin/dse cassandra -f
- bin/cqlsh
- describe keyspaces;
- CREATE KEYSPACE IF NOT EXISTS social…
- use social;
- create table socialgraph(left text, label text, right text, primary key((left, label), right));

Insert records

- cqlsh:social> insert into socialgraph (left, label, right) values ('Joe', 'FRIENDS', 'BILL');
- cqlsh:social> insert into socialgraph (left, label, right) values ('Joe', 'FRIENDS', 'Amy');
- cqlsh:social> insert into socialgraph (left, label, right) values ('Joe', 'FRIENDS', 'Bill');
- cqlsh:social> insert into socialgraph (left, label, right) values ('Joe', 'FRIENDS', 'Elaine');

Question

- How many rows do I have?
  - Zero
  - One
  - Three
  - Four
- How many friends does Joe have?
  - Zero
  - One
  - Three
  - Four

Question

- Assuming we actually stored Joe's friends…
- In what order will they be returned?

Storage

Notice I have 1 row, no values, a timestamp and a rowkey, notice ‘name’ does not match any of the column names {left, label, right}, notice the order of the rows, and that we have BILL and Bill

list socialgraph;

RowKey: Joe:FRIENDS

=> (name=Amy:, value=, timestamp=1426221007206000)

=> (name=BILL:, value=, timestamp=1426220886971000)

=> (name=Bill:, value=, timestamp=1426221027221000)

=> (name=Elaine:, value=, timestamp=1426221068236000)

1 Row Returned.

CQL

```
cqlsh:social> select * from socialgraph;

 left | label    | right
------+---------+--------
  Joe | FRIENDS |    Amy
  Joe | FRIENDS |   BILL
  Joe | FRIENDS |   Bill
  Joe | FRIENDS | Elaine

(4 rows)
```

Remember this…

Distributed multi-dimensional map, indexed by a partition key

Drop table socialgraph

CREATE TABLE IF NOT EXISTS socialgraph (left text, label text,right text,inactive boolean, inactivetimestamp timestamp, PRIMARY KEY ((left, label), right));

Insertion

where we have columns defined that are beyond the primary key

insert into socialgraph(left, label, right, inactive, inactivetimestamp) values ('Joe', 'FRIEND', 'Bill', true, dateof(now()));

select * from socialgraph;

```
 left | label  | right | inactive | inactivetimestamp
------+--------+-------+--------------------------+----------
  Joe | FRIEND |  Bill |     True | 2015-03-12 23:21:59-0700
```

Storage wide row

list socialgraph;

RowKey: Joe:FRIEND

=> (name=Bill:, value=, timestamp=1426229082899000)

=> (name=Bill:inactive, value=01, timestamp=1426229082899000)

=> (name=Bill:inactivetimestamp, value=0000014c11e0bb13, timestamp=1426229082899000)

1 Row Returned.

Insert 2 more friends

- insert into socialgraph(left, label, right, inactive, inactivetimestamp) values ('Joe', 'FRIEND', 'Elaine', true, dateof(now()));
- insert into socialgraph(left, label, right, inactive, inactivetimestamp) values ('Joe', 'FRIEND', 'Amy', true, dateof(now()));

A wide row

RowKey: Joe:FRIEND

=> (name=Amy:, value=, timestamp=1426229747682000)

=> (name=Amy:inactive, value=01, timestamp=1426229747682000)

=> (name=Amy:inactivetimestamp, value=0000014c11eadfe2, timestamp=1426229747682000)

=> (name=Bill:, value=, timestamp=1426229082899000)

=> (name=Bill:inactive, value=01, timestamp=1426229082899000)

=> (name=Bill:inactivetimestamp, value=0000014c11e0bb13, timestamp=1426229082899000)

=> (name=Elaine:, value=, timestamp=1426229745885000)

=> (name=Elaine:inactive, value=01, timestamp=1426229745885000)

=> (name=Elaine:inactivetimestamp, value=0000014c11ead8dd, timestamp=1426229745885000)

1 Row Returned.

Insert a different relationship

```
insert into socialgraph(left, label, right, inactive) values
('Joe', 'LIKES', 'Football', false);
```

2 Wide Rows

2 Entity Lists {Friends List, Likes List}

list socialgraph;

RowKey: Joe:LIKES

=> (name=Football:, value=, timestamp=1426230483652000)

=> (name=Football:inactive, value=00, timestamp=1426230483652000)

RowKey: Joe:FRIEND

=> (name=Amy:, value=, timestamp=1426229747682000)

⇒ (name=Amy:inactive, value=01, timestamp=1426229747682000)

… (Remainder elided for brevity)

2 Rows Returned

Cassandra

- Fixed columns
- Dynamic columns
- Narrow or wide rows
- Wide rows about 100K columns is the practical limit per row, 2B is the advertised limit.
- One I/O operation per row

Capacity

Impacted by

- Nodes in the cluster
- Partition key – sharding
- Clustering column(s)
- Data types and sizes
- Number of columns per entity
- Maps, sets, lists, …
- http://www.sestevez.com/sestevez/CASTableSizer/

Core concepts

- Partition Key
- Clustering Column
- Primary key

Partition key

What happens when I redefine the partition key

From

- create table socialgraph(left text, label text, right text, primary key((left, label), right));

To

- create table socialgraph(left text, label text, right text, primary key((left), label, right));

Answer

This

Joe:Friends | Amy | BILL | BILL | Elaine

Joe:Likes | Fooball

Changes to one wide row ordered by label, right

Joe | Friend:Amy | Friend:BILL | Friend: Bill | Friend:Elaine | Likes:Football

Questions on that?

Wide Rows

- Essentially a linked list

Wide Rows

- Essentially a linked list
- Distributed and accessed by the partition key

Wide Rows

- Essentially a linked list
- Distributed and accessed by the partition key
- Stored and fetched in the order of the clustering columns

Insight

- cqlsh> tracing on;

```
cqlsh:social> select * from socialgraph;

 left | label  | right    | inactive | inactivetimestamp
------+--------+----------+----------+--------------------------
  Joe |  LIKES | Football |    False |                     null
  Joe | FRIEND |      Amy |     True | 2015-03-12 23:55:47-0700
  Joe | FRIEND |     Bill |     True | 2015-03-12 23:44:42-0700
  Joe | FRIEND |   Elaine |     True | 2015-03-12 23:55:45-0700

(4 rows)

Tracing session: 2533e9a0-ca49-11e4-9be7-ff4e6f1f6740
```

Trace Output

```
Tracing session: 2533e9a0-ca49-11e4-9be7-ff4e6f1f6740

 activity                                                                          | timestamp    | source    | sourc
-----------------------------------------------------------------------------------+--------------+-----------+------
                                                                execute_cql3_query | 05:53:43,355 | 127.0.0.1 |
                                      Parsing select * from socialgraph LIMIT 10000; | 05:53:43,355 | 127.0.0.1 |
                                                               Preparing statement | 05:53:43,355 | 127.0.0.1 |
                                                      Determining replicas to query | 05:53:43,356 | 127.0.0.1 |
 Executing seq scan across 0 sstables for [min(-9223372036854775808), min(-9223372036854775808)] | 05:53:43,357 | 127.0.0.1 |
                                                   Read 1 live and 0 tombstoned cells | 05:53:43,358 | 127.0.0.1 |
                                                   Read 3 live and 0 tombstoned cells | 05:53:43,358 | 127.0.0.1 |
                                                         Scanned 2 rows and matched 2 | 05:53:43,358 | 127.0.0.1 |
                                                                  Request complete | 05:53:43,358 | 127.0.0.1 |
```

Cool features

- Store values in a column names, perfectly OK
- Wide rows – one I/O operation
- Partition Key
  - Sharding
    - User, friends
    - User, followers
    - User, Groups
    - User, GeneratedContent
- Clustering column asc, desc
- TTL

Realities

- Can only ‘efficiently’ get to data by using the partition key
- Can only query data in the order of the partition key followed by the exact ordering of clustering columns
- Indexes yes, but…
- Foreign keys no, Joins no

Indexes

- Problem
  - Extract a set of relationships from edges where the edges table is structured as UUID ← Relationship → UUID
  - For each UUID fetch the entity meta data from another table

This is both an Anti-Pattern and it is aligned with Titan's strategy

Findings

13000 Friends Simple Sequential Process

Cost of in application join 10 seconds

Strategy	Time
Index lookup	80 Seconds
Table lookup by each	29 seconds
Table lookup batches of 2000	12 seconds
No lookup	2 seconds
Add AKKA / RxJava Improve Sharding	< 1 second (TBD)

Agenda

- ~~Data modeling~~
- Materialized views in Social
- Titan
- Architecture

Viewpoint

Materialized views solve all problems

Implied Materialized views in Social

- Friends
- Friends filtered by visibility
- Friends sorted by Last Name, filtered by visibility
- Friends sorted by First Name, filtered by visibility
- …
- Internationalized

Viewpoint

Is actually not so sweet

Materialized views in Social

- Friends
- Friends filtered by visibility
- Friends sorted by Last Name, filtered by visibility
- Friends sorted by First Name, filtered by visibility
- …
- Internationalized

YUK

- Marriages, divorces, changes to visibility
- Drift detection and correction in a changing system*

Agenda

- ~~Data modeling~~
- ~~Materialized views in Social~~
- Titan
- Architecture

Two Viewpoints

- DataStax Cassandra
  - Everything is a materialized view
  - Denormalize
  - Data redundancy is the solution
- Titan
  - Everything can be represented as a graph
  - 7 tables will do it

Titan

- Backed by Cassandra (other options)
- Astyanax DB Driver
  - THRIFT (WITH COMPACT STORAGE)
  - Downsides: data is in a blob
- cqlsh> describe keyspace titan;

Titan Tables

- Edgeindex
- Edgestore
- Edgestore_lock_
- System_properties
- Titan_ids
- Vertexindex
- Vertexindex_lock_

Titan Core

- Two key tables Entity and Entity Relationship
  - Vertexes (Neo4J calls nodes)
    - Could be done several ways
      - Key value: key to object (blob)
      - Dynamic columns: object mapped to key value pairs
  - Edges
    - Node1 ← Friend → Node2
    - Relationships are a first class entity
- Pixie Dust
- No materialized views. In application joins.

TinkerPop

TinkerPop2

Blueprints

Blueprints is a property graph model interface with provided implementations. Databases that implement the Blueprints interfaces automatically support Blueprints-enabled applications.

Pipes

Pipes is a dataflow framework that enables the splitting, merging, filtering, and transformation of data from input to output. Computations are evaluated in a memory-efficient, lazy fashion.

Gremlin

Gremlin is a domain specific language for traversing property graphs. This language has application in the areas of graph query, analysis, and manipulation

Frames

Frames exposes the elements of a Blueprints graph as Java objects. Instead of writing software in terms of vertices and edges, with Frames, software is written in terms of domain objects and their relationships to each other.

Furnace

Furnace is a property graph algorithms package. It provides implementations for standard graph analysis algorithms that can be applied to property graphs in a meaningful ways.

Rexster

Rexster is a multi-faceted graph server that exposes any Blueprints graph through several mechanisms with a general focus on REST.

JUnit

- Titan JUnit Tests
- shall7m2:~ shall7$ cd dse
- shall7m2:dse shall7$ bin/dse cassandra -f

Titan

- Purchased by DataStax
- Being incorporated into DataStax Enterprise

Titan

- We did not choose Titan
  - Time to market
  - Complexity
  - Data in blobs

Downsides

- Materialized Views
  - Update and maintenance challenges
- Application Joins
  - Slow, perhaps
  - Functional Reactive Parallel: RxJava
  - Massively parallel: Spark, AKKA
  - Solr

Agenda

- ~~Data modeling~~
- ~~Materialized views in Social~~
- ~~Titan~~
- Architecture

Pragmatic

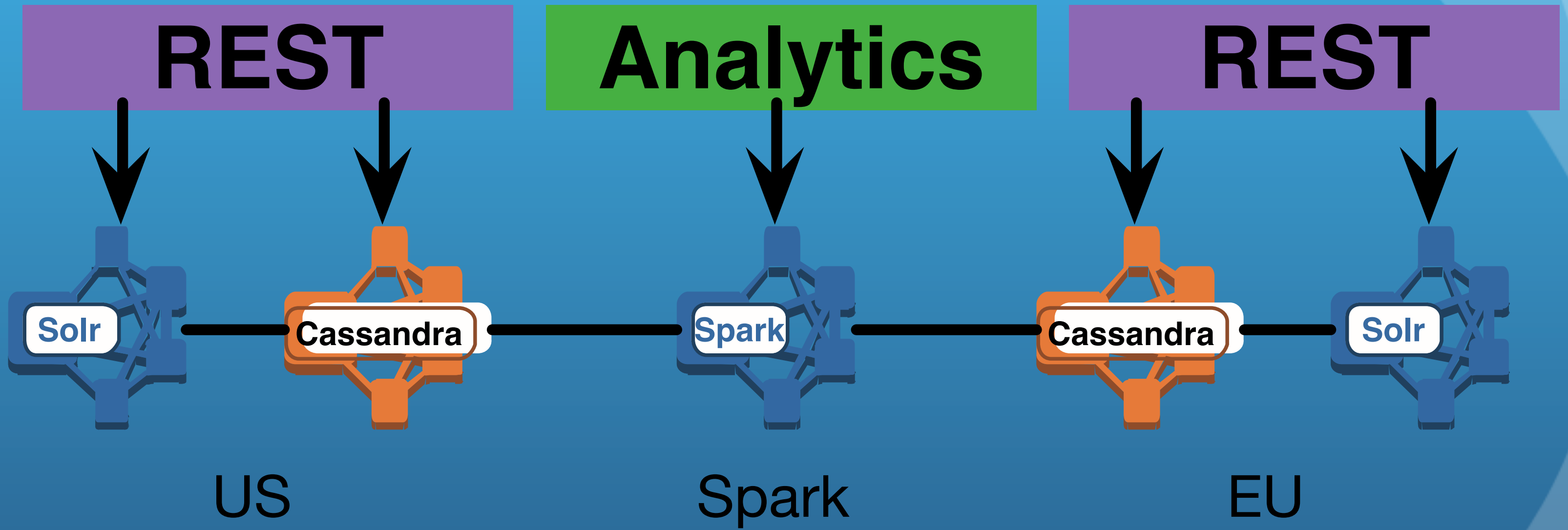

RING
LIST
Health
Normal
9
Medium
0
High
0
Down
0
Data Size
Total Size
1.3 GB
Avg. Node
151 MB
Standard Deviation
+/- 81 MB
Alerts
0
Datacenter us-west-2
3
0
0
0
Datacenter us-west-2-solr
Workload: Search
3
0
0
0
Datacenter us-west-2-spark
Workload: Spark Analytics
3
0
0
0

Emergent Architecture

- Cassandra
- Solr Cassandra
- Spark Cassandra
- Spark Cassandra Graphx
- Spark Job Server

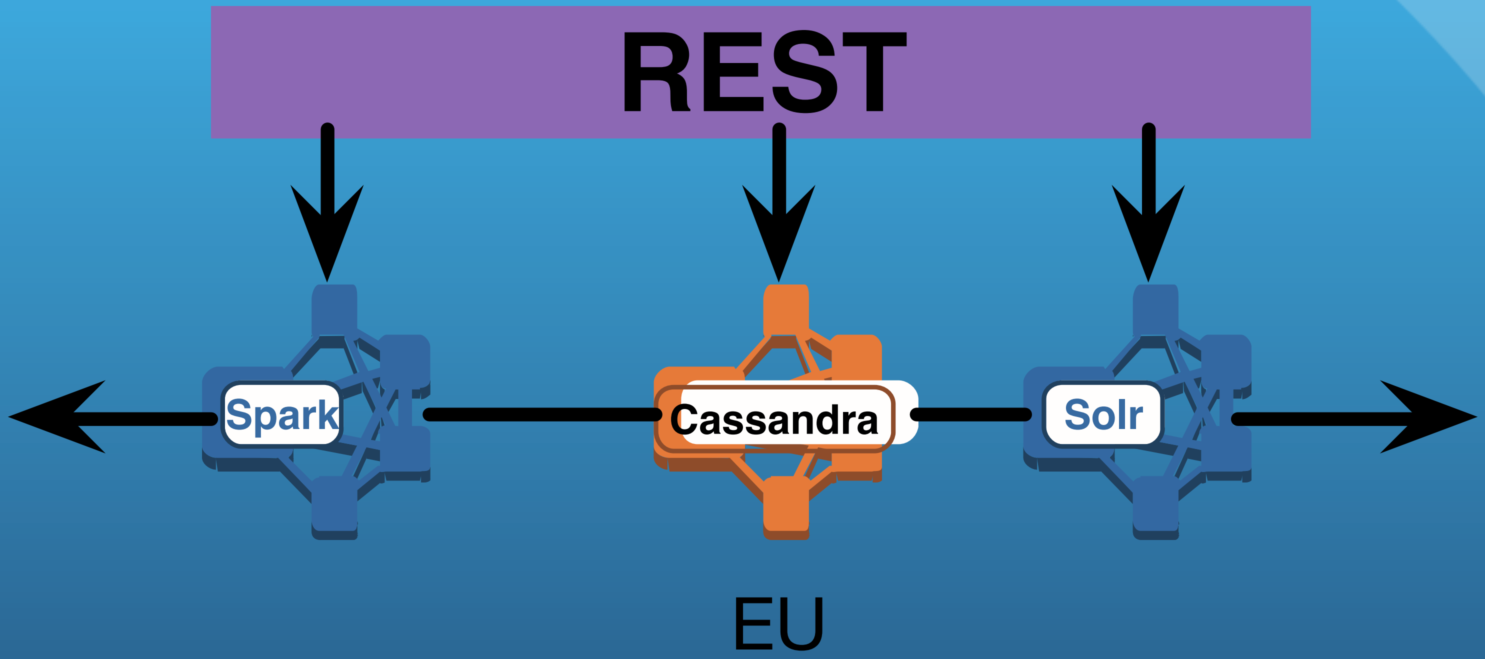
Multi-Region With Analytics

All nodes backed by Cassandra

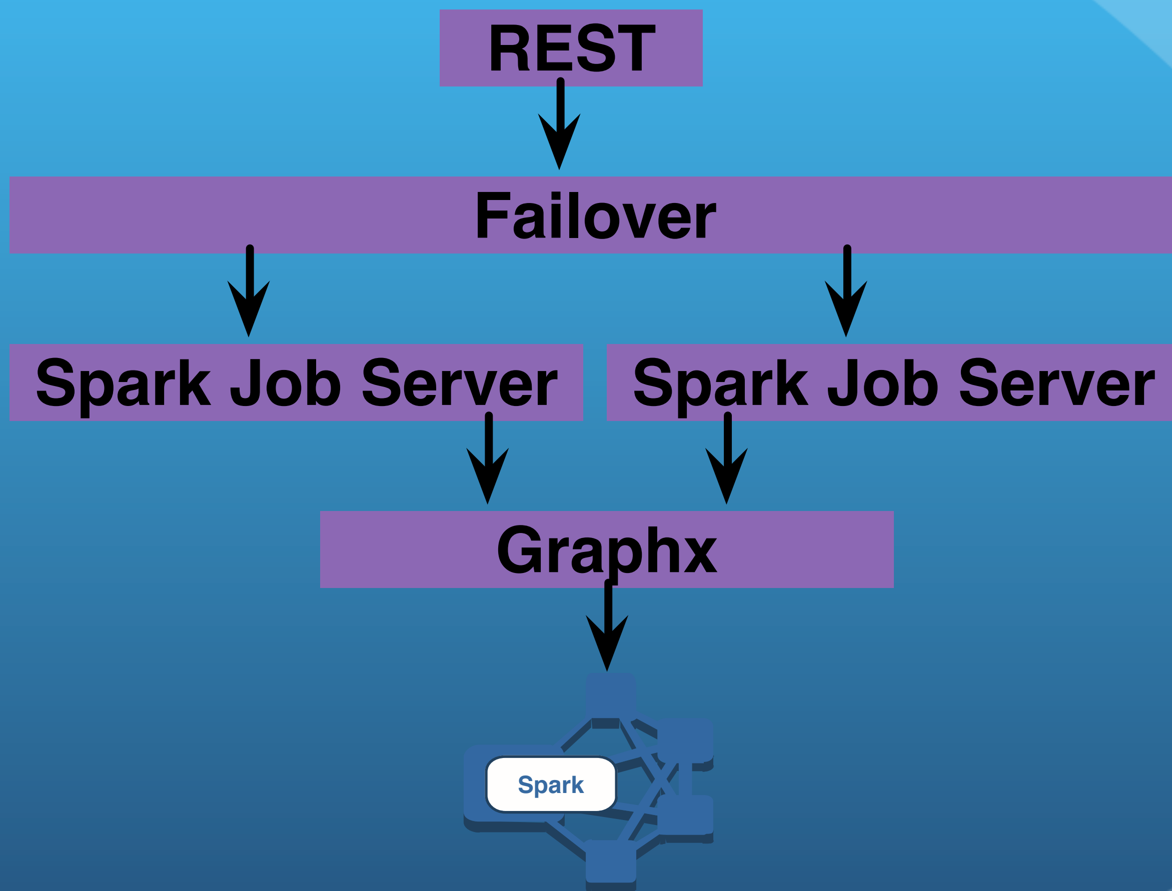


Multi-Region With Graph

All nodes backed by Cassandra



Spark With Graphx



Summary

- Cassandra fastest runtime solution is materialized views

Summary

- Cassandra fastest runtime solution is materialized views
- Materialized views are:
  - Lists of domain objects

Summary

- Cassandra fastest runtime solution is materialized views
- Materialized views are:
  - Lists of domain objects
  - Sharded across the cluster using the partition key

Summary

- Cassandra fastest runtime solution is materialized views
- Materialized views are:
  - Lists of domain objects
  - Sharded across the cluster using the partition key
  - Sorted by clustering column(s)

Summary

- Cassandra fastest runtime solution is materialized views
- Materialized views are:
  - Lists of domain objects
  - Sharded across the cluster using the partition key
  - Sorted by clustering column(s)
  - Accessed by the partition key, then by clustering columns

Summary

- Cassandra fastest runtime solution is materialized views
- Materialized views are:
  - Lists of domain objects
  - Sharded across the cluster using the partition key
  - Sorted by clustering column(s)
  - Accessed by the partition key, then by clustering columns
  - Potentially data repeated throughout many views

Summary

- Titan addresses enterprise data storage using Cassandra without materialized views.

Summary

- Titan addresses enterprise data storage using Cassandra without materialized views.
- Cassandra by itself is not equivalent to a relational database.

Summary

- Titan addresses enterprise data storage using Cassandra without materialized views.
- Cassandra by itself is not equivalent to a relational database.
  - Materialized views obviate many relational db features

Summary

- Titan addresses enterprise data storage using Cassandra without materialized views.
- Cassandra by itself is not equivalent to a relational database.
  - Materialized views obviate many relational db features
  - Finding the partition key may require augmentation with search, such as Solr, Elastic Search, etc.

Summary

- Cassandra natively supports data replication across data centers.

Summary

- Cassandra natively supports data replication across data centers.
  - Multi-region solutions

Summary

- Cassandra natively supports data replication across data centers.
  - Multi-region solutions
  - Same technique can be used to replicate across Cassandra, Solr and Cassandra, Spark and Cassanda clusters.

Summary

- Cassandra natively supports data replication across data centers.
  - Multi-region solutions
  - Same technique can be used to replicate across Cassandra, Solr and Cassandra, Spark and Cassanda clusters.
  - Combine across one region or many

Supported Solutions

- DataStax provides an integrated solution with professional support
  - Cassandra, Solr, Spark
  - Integrated
  - Integration with Titan coming

Intrigued, want to know more

Steven.Hall@nike.com