Hadoop Application Architectures: Architecting a Next Generation Data Platform

Strata Data Conference, New York 2017

tiny.cloudera.com/app-arch-newyork
tiny.cloudera.com/nyquestions

Mark Grover | @mark_grover
Jonathan Seidman | @jseidman
Gwen Shapira | @gwenshap
Logistics

- Break at 3:00 – 3:30 PM
- Questions at the end of each section
- Slides at tiny.cloudera.com/app-arch-newyork
- Code at https://github.com/hadooparchitecturebook/Taxi360
About the book

- @hadooparchbook
- hadooparchitecturebook.com
- github.com/hadooparchitecturebook
- slideshare.com/hadooparchbook
About the presenters

Mark Grover

- Product Manager at Lyft
- Formerly Software Engineer on Spark at Cloudera
- Committer on Apache Bigtop, PMC member on Apache Sentry, Apache Spot (incubating)
- Contributor to Apache Spark, Hadoop, Hive, Sqoop, Pig, Flume
About the presenters

Gwen Shapira

- Product Manager at Confluent
- PMC for Apache Kafka.
- Previously software engineer at Cloudera
- @gwenshap on twitter
About the presenters

Jonathan Seidman

▪ Software Engineer at Cloudera
▪ Contributor to Apache Sqoop.
▪ Previously Technical Lead on the big data team at Orbitz, co-founder of the Chicago Hadoop User Group and Chicago Big Data
Case Study Overview
Internet of Things and Entity 360
Customer 360

Questions?
tiny.cloudera.com/nyquestions
Connected Cars
Entity (Taxi) 360 View

- Streaming Vehicle Data
- Geo-location/Traffic Data
- Customer Data
- Maintenance Data
- Other Data Sources

Questions?
tiny.cloudera.com/nyquestions
What Makes Hadoop a Fit?

The early days…

Data Sources → Extract → Transform → Load
What Makes Hadoop a Fit?

Today…

Servers
Marts
EDWS
Documents
Storage
Search
Archive

ERP, CRM, RDBMS, Machines
Files, Images, Videos, Logs, Clickstreams
External Data Sources

Questions?
tiny.cloudera.com/nyquestions
Enabling a Range of New Use Cases…

- Fraud Detection
- Market Transactions
- Internet of Things
- Network Security

Questions?
tiny.cloudera.com/nyquestions
Hadoop Challenges

Kafka

Kafka Connect

Spark

Flink

presto

Apache Drill

Apache Solr

Questions?
tiny.cloudera.com/nyquestions
Challenges – Architectural Considerations

- Reliable and scalable ingress of multiple data types and sources:
  - High volume event data? Batch data?
- Reliable and scalable storage to support multiple workloads and access patterns
  - Historical data? Real-time search? Analytics?
- Processing engines (for background processing):
  - Stream processing? Batch processing?
- Data Modeling
  - Modeling data for real-time random access? Analytic access? Batch access?
Requirements

- Allow users (technical and non-technical) to analyze and visualize data…
Requirements

- Provide analysts with query capabilities via a standard interface…
Requirements

- Provide developers the ability to perform batch processing on historical data…
Requirements

- To support all this, we need:
  - Reliable ingestion of streaming and batch data.
  - Ability to perform transformations on streaming data in flight.
  - Ability to perform sophisticated processing of historical data.
High level architecture

Walkthrough
High level architecture

Source
- Data Producers

Transport
- Pub-Sub

Stream Processing
- Processing & Ingestion Engine

Storage
- Nested Tables
- Relational Tables
- Entity Time Series Lookup
- Indexed Cube

Access
- Batch Processing
- SQL
- NRT REST
- NRT Dashboard

Questions?
tiny.cloudera.com/nyquestions
Data Sources
Considerations
High level architecture

Source

Transport

Stream Processing

Storage

Access

Data Producers

Pub Sub

Processing & Ingestion Engine

Nested Tables

Batch Processing

Relational Tables

SQL

Entity Time Series Lookup

NRT REST

Indexed Cube

NRT Dashboard

Batch Processing

#StrataData

Questions?
tiny.cloudera.com/nyquestions
Key to Customer 360 Success

Your project is only as good as the quality and variety of data sources

- Geo-location/Traffic Data
- Files: CSV? XML? JSON?
- Database
- Maintenance Data
- Customer Data
- Salesforce?
- Other Data Sources
- Twitter? Mainframe?

Streaming Vehicle Data

Questions?
tiny.cloudera.com/nyquestions
Data Producers: Flume vs. Kafka

- Flume – well integrated with Hadoop.
  - Part of Hadoop ecosystem
  - Great choice when ingesting data into HDFS.
  - Can support simple transformations.
  - Minimal coding – built in support for common data sources.

- Kafka – flexible, get-everything pipe
  - Producers in ~ 20 languages
  - REST API
  - Huge connector ecosystem
Kafka Clients

Apache Kafka Clients

Ecosystem Clients

Java
Scala
Clojure
Ruby
Perl
php

C
Python
Go

R

.NET
node.js
Erlang
REST Proxy
Talking to Non-native Kafka Apps and Outside the Firewall

- Provides a RESTful interface to a Kafka cluster
- Simplifies message creation and consumption
- Simplifies administrative actions

Diagram:
- Non-Java Applications
- Native Kafka Java Applications
- REST Proxy
- REST / HTTP
Kafka Connect
Streaming Data Capture

- Fault tolerant
- Manage hundreds of data sources and sinks
- Preserves data schema
- Part of Apache Kafka project
- Includes simple transformations

Kafka Connect API

- Sources
  - JDBC
  - Logs
  - MQTT

- Kafka

- Connectors
  - Connector
  - Connector
  - Connector
  - Connector

- Sinks
  - RDBMS
  - Key/Value
  - HDFS

Questions?
tiny.cloudera.com/nyquestions
## Ecosystem of Connectors

<table>
<thead>
<tr>
<th>Databases</th>
<th>Datastore/File Store</th>
<th>Analytics</th>
<th>Applications / Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC</td>
<td>hadoop</td>
<td>VERTICA</td>
<td>Amazon web services</td>
</tr>
<tr>
<td>DATASTAX</td>
<td>Ignite</td>
<td>elastic</td>
<td>S3</td>
</tr>
<tr>
<td>ORACLE</td>
<td>FTP</td>
<td>mixpanel</td>
<td>syncsort</td>
</tr>
<tr>
<td>Just One Data</td>
<td>syslog-ng</td>
<td>Solr</td>
<td>Attunity</td>
</tr>
<tr>
<td>GOLDEN GATE</td>
<td>hadelcast</td>
<td>Apache</td>
<td>Bloomberg</td>
</tr>
<tr>
<td>Couchbase</td>
<td></td>
<td></td>
<td>Twitter</td>
</tr>
<tr>
<td>DynamoDB</td>
<td></td>
<td>Kudu</td>
<td>MQTT</td>
</tr>
<tr>
<td>InfluxDB</td>
<td></td>
<td>RethinkDB</td>
<td></td>
</tr>
<tr>
<td>HBASE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Just One Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions? [tiny.cloudera.com/nyquestions](tiny.cloudera.com/nyquestions)
How Connect Works?

REST API

Log Connector

Log Task

MQTT Task

Log Task

MQTT Connector

Logs

MQTT

Questions?
tiny.cloudera.com/nyquestions
Schema Registry

Get different data sources to talk the same language

Prevent backwards incompatible changes

Define the expected fields for each Kafka topic

Automatically handle schema changes (e.g. new fields)
High level architecture

Source

Transport

Stream Processing

Processing & Ingestion Engine

Storage

Nested Tables

Relational Tables

Entity Time Series Lookup

Indexed Cube

Access

Batch Processing

SQL

NRT REST

NRT Dashboard
But wait!

What about batch data?
Buffering
Buffering Data

- What do we mean by “buffering” and why do we need it?

- Network partitions happen
- Producers and Consumers work at different rates
- Reliable storage is hard
  Stream processing is hard
  Lets do one at a time

This is bad!
Buffering Data – Message Brokers
High level architecture

Source

Buffer

Stream Processing

Processing & Ingestion Engine

Storage

Nested Tables

Relational Tables

Entity Time Series Lookup

Indexed Cube

Access

Batch Processing

SQL

NRT REST

NRT Dashboard

#StrataData

Questions?
tiny.cloudera.com/nyquestions
What is Kafka?

- It’s like a message queue, right?
- Actually, it’s a “distributed commit log”
- Or “streaming data platform”
Topics and Partitions

- Messages are organized into topics, and each topic is split into partitions.
  - Each partition is an immutable, time-sequenced log of messages on disk.
  - Note that time ordering is guaranteed within, but not across, partitions.
In Our Architecture

Producers:
- Taxi Trip Data

Kafka Topic:
- taxi-trip-input

Consumers:
- Stream Processing (Analytic)
- Stream Processing (Lookup)
- Stream Processing (Search)
- Stream Processing (Long Term)
### Input Events

<table>
<thead>
<tr>
<th>vendor_name</th>
<th>Trip_Pickup_DateTime</th>
<th>Trip_Dropoff_DateTime</th>
<th>Passenger_Count</th>
<th>Trip_Distance</th>
<th>Start_Lon</th>
<th>Start_Lat</th>
<th>Rate_Code</th>
<th>store_and_forward</th>
<th>End_Lon</th>
<th>End_Lat</th>
<th>Payment_Type</th>
<th>Fare_Amt</th>
<th>surcharge</th>
<th>mta_tax</th>
<th>Tip_Amt</th>
<th>Tolls_Amt</th>
<th>Total_Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT,2009-01-05 08:31:55</td>
<td>2009-01-05 08:37:50</td>
<td>1</td>
<td>0.90000000000000002</td>
<td>-73.977936999999997</td>
<td>40.745919000000001</td>
<td>-73.983609000000001</td>
<td>40.755051000000002</td>
<td>Credit</td>
<td>5.2999999999999998</td>
<td>0</td>
<td>0.79000000000000004</td>
<td>0</td>
<td>6.0899999999999999</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Kafka Considerations – Reliability

- But remember there are tradeoffs…
Kafka Considerations – Reliability

- Different reliability levels for topics:
  - Taxi Trip Data
  - Twitter
  - Kafka

  100% – dups are ok ("At least once")
  <=100% ("At most once")

News Flash: Kafka's Exactly Once Producer is on the way
Kafka Reliability – Replication

Producer

Broker
  Leader
  Partition 1
    Partition 2
      Partition 3
Kafka Reliability – Replication

Producer

Partition 1

Partition 2

Partition 3
Kafka Reliability – Replication
Kafka Reliability – Replication

Questions?
tiny.cloudera.com/nyquestions
Kafka Reliability – Replication

Questions?
tiny.cloudera.com/nyquestions
Kafka Reliability – Replication

- So how does this relate to our application?

```
kafka-topics --zookeeper ZKHOST:ZKPORT -partition 2 --replication-factor 3 \
    --create --topic taxi-trip-input
```

```
kafka-topics --zookeeper ZKHOST:ZKPORT -partition 2 --replication-factor 1 \
    --create --topic customer-sentiment
```
Kafka Reliability – Producers

```
Producer
Taxi Trip Data

Kafka
```

```
taxi_trip_input
  Partition 1
  Partition 2
  Partition 3

Topic B
  Partition 1
  Partition 2
  Partition 3
```

Message failure?

acks=all

Resend message

Questions?
tiny.cloudera.com/nyquestions
Kafka Reliability – Producers

▪ What about duplicates?

Taxi Trip Data

<table>
<thead>
<tr>
<th>ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2009-01-04 03:02:00,1,2,629,...</td>
</tr>
<tr>
<td>1001</td>
<td>2009-01-04 03:38:00,3,4,549...</td>
</tr>
<tr>
<td>1001</td>
<td>2009-01-04 03:38:00,3,4,549...</td>
</tr>
</tbody>
</table>
Kafka Scaling – Partitions

Producer

Kafka

taxi-trip-input

Partition 1
Partition 2
Partition 3

Consumer Group

Consumer
Consumer
Consumer

Questions?
tiny.cloudera.com/nyquestions
Kafka Scaling – Partitions

Producer

Producer

More resources (memory)

Higher throughput

Kafka
taxi-trip-input

Partition 1

Partition 2

Partition 3

Partition 4

Partition 5

Higher throughput

Consumer Group

Consumer

Consumer

Consumer

Consumer

More resources (file handles)

Questions?
tiny.cloudera.com/nyquestions
How many partitions?

- Adding partitions late in the game is painful
- Basic formula:
  total desired throughput / throughput of slowest consumer or producer
- Or ~25GB disk space
- Not too many because:
  - Each partition takes broker heap memory and file handles
  - Each partition slows down node shutdown / recovery
  - 1000 – 4000 partitions per broker max
  - Producers will produce smaller batches – lower throughput
Kafka Scaling – Producers

Producer

Kafka

taxi-trip-input

Partition 1
Partition 2
Partition 3
Partition 4
Partition 5

Consumer Group

Consumer
Consumer
Consumer
Consumer
Consumer

Questions?
tiny.cloudera.com/nyquestions
Guarding Against Message Loss

- **Producer** – What happens if the producer loses connection to Kafka and the buffer overflows?
  - You get an exception. You can choose to… block? Write to file?

- **Source** – What happens if events are lost before getting sent to producer?
  - Once again use some kind of buffer to provide sufficient retention of data.
Stream Processing
Considerations
High level architecture

Source
- Custom Producer or

Transport
- kafka

Stream Processing
- Processing & Ingestion Engine

Storage
- Nested Tables
- Relational Tables
- Entity Time Series Lookup
- Indexed Cube

Access
- Batch Processing
- SQL
- NRT REST
- NRT Dashboard

Questions?
tiny.cloudera.com/nyquestions
Streaming agenda

▪ What do we mean by streaming?
▪ Streaming use-cases
▪ Streaming semantics
▪ Which streaming engine to choose?
▪ Streaming in our use-case
What do we mean by streaming?
What do we mean by streaming?

- **Constant low milliseconds & under**
- **Low milliseconds to seconds, delay in case of failures**
- **10s of seconds or more, re-run in case of failures**

- **Real-time**
- **Near real-time**
- **Batch**

Questions?
tiny.cloudera.com/nyquestions
What do we mean by streaming?

- **Real-time**
- **Near real-time**
- **Batch**

**Constant low milliseconds & under**

**Low milliseconds to seconds, delay in case of failures**

**10s of seconds or more, re-run in case of failures**
But, there’s no free lunch

Constant low milliseconds & under

Low milliseconds to seconds, delay in case of failures

10s of seconds or more, re-run in case of failures

Real-time

Near real-time

Batch

“Difficult” architectures, lower latency

“Easier” architectures, higher latency
Streaming use-cases
Streaming Use-cases

- Ingestion (most relevant in our use-case)
- Simple transformations
  - Decision (e.g. anomaly detection)
  - Enrichment (e.g. add a state based on zipcode)
- Advanced usage
  - Machine Learning
  - Windowing
#1 - Simple ingestion

Buffer → Stream Processing → Long term storage

Event e → Event e
#2 - Enrichment

Diagram:
- Buffer
- Event e
- Stream Processing
- Event e'
- Storage
- Context store
- e' = enriched event e
#2 - Decision

\[ e' = e + \text{decision} \]

Diagram:
- Buffer
- Event e
- Stream Processing
- Event e'
- Storage
- Rules
#3 – Advanced usage

e’ = aggregation or windowed aggregation

Buffer → Event e → Stream Processing → Event e’ → Storage

Model
#1 – Simple Ingestion

1. **Zero transformation**
   - No transformation, plain ingest
   - Keep the original format – SequenceFile, Text, etc.
   - Allows to store data that may have errors in the schema

2. **Format transformation**
   - Simply change the format of the field
   - To a structured format, say, Avro, for example
   - Can do schema validation

3. **Atomic transformation**
   - Mask a credit card number
#2 - Enrichment

\[ e' = \text{enriched event } e \]

Diagram:
- Buffer
- Event e
- Stream Processing
- Event e'
- Storage
- Context store
- Need to store the context somewhere
Where to store the context?

1. Locally Broadcast Cached Dim Data
   - Local to Process (On Heap, Off Heap)
   - Local to Node (Off Process)
2. Partitioned Cache
   - Shuffle to move new data to partitioned cache
3. External Fetch Data (e.g. HBase, Memcached)
#1a - Locally broadcast cached data

Could be On heap or Off heap
#1b - Off process cached data

Data is cached on the node, outside of process. Potentially in an external system like Rocks DB.
#2 - Partitioned cache data

Data is partitioned based on field(s) and then cached.
#3 - External fetch

Data fetched from external system

Dimension data in external system

Questions?
tiny.cloudera.com/nyquestions
Partitioned cache + external

Node
Process
Partitioned Data

Node
Process
Partitioned Data

Node
Process
Partitioned Data

Dimension data in external system

Questions?
tiny.cloudera.com/nyquestions
Streaming semantics
Delivery Types

▪ **At most once**
  - Not good for many cases
  - Only where performance/SLA is more important than accuracy

▪ **Exactly once**
  - Expensive to achieve but desirable

▪ **At least once**
  - Easiest to achieve
Semantics of our architecture

Source System 1 → Ingest → Message broker
Source System 2 → Ingest → Message broker
Source System 3 → Ingest → Message broker

Message broker → Extract → Streaming engine → Push → Destination system

- At least once
- At least once
- Ordered Partitioned
- It depends
- It depends

Questions?
tiny.cloudera.com/nyquestions
Classification of storage systems

- File based
  - S3
  - HDFS
- NoSQL
  - HBase
  - Cassandra
- Document based
  - Search
- NoSQL-SQL
  - Kudu
Classification of storage systems

- File based
  - S3
  - HDFS
- NoSQL
  - HBase
  - Cassandra
- Document based
  - Search
- NoSQL-SQL
  - Kudu

De-duplication at file level

Semantics at key/record level
Which streaming engine to choose?
High level architecture

Source

Transport

Stream Processing

Processing & Ingestion Engine

Storage

Nested Tables

Relational Tables

Entity Time Series Lookup

Indexed Cube

Access

Batch Processing

SQL

NRT REST

NRT Dashboard

#StrataData

tiny.cloudera.com/nyquestions
Requirements

- Fault-tolerant and distributed
- Effectively once semantics
- Handle processing time vs. event time
- Allow stateful transformations
Spark Streaming

- Micro batch based architecture
- Allows stateful transformations
- Feature rich
  - Windowing
  - Sessionization
  - ML
  - SQL (Structured Streaming)
Questions?
tiny.cloudera.com/nyquestions
Spark Streaming - Gaps

- Not as low of a latency
  - Efforts towards reducing latency e.g. RISElab’s Drizzle

- Global consistent execution state
  - Stop overall execution of distributed computation
  - Eagerly persist records in transit meaning larger snapshots
Flink

- True “streaming” system, but not as feature rich as Spark
- Much better event time handling
- Good built-in backpressure support
- Allows stateful transformations
- Lower Latency
  - No Micro Batching
  - Asynchronous Barrier Snapshotting (ABS)
Flink - ABS

Operator

Buffer
Flink - ABS

Questions?
tiny.cloudera.com/nyquestions
Flink - ABS

Questions?
tiny.cloudera.com/nyquestions
Flink - ABS

Both Barriers Hit

Buffer can be flushed out

Buffer can be flushed out

Barrier is combined and can move on

Check Point

Operator

Buffer

Operator

Buffer
Storm

- Old school
- Didn’t manage state – had to use Trident
- No good support for batch processing
Samza

- Good integration with Kafka
- Doesn’t support batch
- Forked by Kafka Streams
Flume

- Well integrated with the Hadoop ecosystem
- Allowed interceptors (for simple transformations)
- Supports buffering
  - Memory
  - File
  - Kafka
- But no real fault-tolerance
- No state management
Kafka Streams

- Good integration with Kafka
- Light-weight library (not a framework)
- No micro-batching, uses Kafka as internal messaging layer
- Maintains local state per node (in RocksDB, or in memory hash map)
- Handles late events
- Stream-to-stream joins
Kafka Streams architecture

Topic
Partition 1
Partition 2

Task 1
Task 2

Re-partition topic
Partition 1
Partition 2

Task 3
Task 4
Apache Beam

- Abstraction on top of Streaming Engines
- Best support for Google Dataflow
Others

- Apache Apex
- Heron
Streaming in our use-case
Spark Streaming

- We chose Spark Streaming because:
  - Same execution engine for batch and streaming
  - Similar code for batch and streaming
  - Support for security, Kafka integration
  - Thriving community
High level architecture

Source: kafka
Transport: kafka
Stream Processing: Spark Streaming
Storage: Nested Tables, Relational Tables, Entity Time Series Lookup, Indexed Cube
Access: Batch Processing, SQL, NRT REST, NRT Dashboard

Questions?
tiny.cloudera.com/nyquestions
Storage Layer
Considerations
High level architecture
Data Modeling
## Structured Landing Zones

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>Traditional SQL</td>
</tr>
<tr>
<td>Nested</td>
<td>Optimized for nested Structures like JSON</td>
</tr>
<tr>
<td>Time Series</td>
<td>Optimized Entity 360 and time base access</td>
</tr>
<tr>
<td>Reversed Indexed</td>
<td>Optimized faceted charts and reverse index look ups</td>
</tr>
<tr>
<td>Graph</td>
<td>Optimized for node and edges</td>
</tr>
<tr>
<td>Special</td>
<td>Optimized for special use cases</td>
</tr>
</tbody>
</table>

Questions?

[tiny.cloudera.com/nyquestions](tiny.cloudera.com/nyquestions)
# Structured Landing Zones

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>Traditional SQL</td>
</tr>
<tr>
<td>Nested</td>
<td>Optimized for nested structures like JSON</td>
</tr>
<tr>
<td>Time Series</td>
<td>Optimized Entity 360 and time base access</td>
</tr>
<tr>
<td>Reverse Indexed</td>
<td>Optimized faceted charts and reverse index look ups</td>
</tr>
<tr>
<td>Graph</td>
<td>Optimized for node and edges</td>
</tr>
</tbody>
</table>
Background Information
Compression Styles and Entropy

Questions?
tiny.cloudera.com/nyquestions
Compression Styles and Entropy

Row Group

Column

Column

Column

Row Group

Column

Column

Column
Compression Codecs

- Snappy: 2x-3x : Fast Read, Fast Write
- Lzo : 2x-3x : Fast Read, Fast Write
- Gzip : ~8x: ~Fast Read, Normal Write
- Default : ~8x: ~Fast Read, Normal Write
- BZip2 : ~10x ~Fast Read, Slow Write
- Others ..

- Always be skeptical
  - All data compresses differently
  - Use your own data
Introducing the Hive Metastore

- Hive Metastore
- Adds a table like metadata layer over a file system, block store, NoSql, or other
- Allows for SQL access
- Allows for greater security options
- Allows for external metadata
- Allows for partitioning
Typical Hive Table

- ParentFolder
  - TableFolder
    - Date=20171212
      - DataFiles
      - DataFiles
    - Date=20171211
      - DataFiles
      - DataFiles
Access Patterns

- Partitioning
- Filter push down
- Indexing should be considered poor
- Ideal for large scans
Relational Storage
Thinking about Object/Tables

1. Lets start off easy

1. Use Case: We are a Netflix type company and we have a log of users and movies watched that looks something like this:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Age</th>
<th>Account Start Date</th>
<th>Category Of User</th>
<th>Movie Watched</th>
<th>Movie Category</th>
<th>Start Time</th>
<th>Events List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>42</td>
<td>12/12/2012</td>
<td>Basic</td>
<td>Die Hard</td>
<td>Action</td>
<td>5/4/2016 12:00</td>
<td>Play 0, pause at 15, FF at 40 to 55, E at 90</td>
</tr>
<tr>
<td>Kat</td>
<td>31</td>
<td>12/12/2012</td>
<td>Platum</td>
<td>Beauty and the Beast</td>
<td>Family</td>
<td>5/4/2016 12:00</td>
<td>Play 0, pause at 15, FF at 40 to 55, E at 90</td>
</tr>
</tbody>
</table>
Thinking about Object/Tables

1. To make this into objects we need to do some separation
Query Considerations

- Data is normally big so
- Partition respectively to access patterns
- Join with care
- Consider sampling or local testing before experimenting
- Data is files
- Latency to accessibility it high – seconds, minutes or more.
Look for big tables
Mutation Patterns

- File is written once and can not be mutated
- Fine for append or snapshot use cases
- Mutation will require a compaction
## Compaction Recap

<table>
<thead>
<tr>
<th>Key</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>103</td>
</tr>
<tr>
<td>H</td>
<td>3</td>
<td>103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>103</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>H</td>
<td>3</td>
<td>103</td>
</tr>
</tbody>
</table>

### Questions?
[Link to questions form](tiny.cloudera.com/nyquestions)
View Strategies

Hive Relational Model

Hive Nested Model

Hive Normal Views

Hive Materialized Table Views

Use only for tables that filter records/ columns or use for marking fields

Use in the cases where the view requires a join that is done through a shuffle
Relational Storage Options
Kudu network architecture

Master tablet  | Tablet 1 | Tablet 2 | \cdots | Tablet n
---|---|---|---|---
Master Server A | Tablet 1 LEADER | Tablet 2 FOLLOWER | \cdots | Tablet n FOLLOWER
Master Server B | Tablet 1 FOLLOWER | Tablet 2 FOLLOWER | \cdots | Tablet n LEADER
Master Server C | Tablet 1 FOLLOWER | Tablet 2 FOLLOWER | \cdots | Tablet n FOLLOWER

Tablet Server W
Tablet Server X
Tablet Server Y
Tablet Server Z

Questions?
tiny.cloudera.com/nyquestions
Kudu Use Cases

- Among other things, NRT availability of streaming data.
  - A good fit for our application.
- Also things like machine learning, time series, etc.
def sendEntityToKudu(taxiEntityTableName: String, it: Iterator[(String, NyTaxiYellowEntityStateWrapper)],
kuduClient: KuduClient): Unit = {
  val table = kuduClient.openTable(taxiEntityTableName)
  val session = kuduClient.newSession()
  session.setFlushMode(FlushMode.AUTO_FLUSH_BACKGROUND)

  it.foreach(r => {
    val state = r._2.state
    val entity = r._2.entity

    val operation: Operation = if (state.equals("New")) {
      table.newInsert()
    } else if (state.equals("Modified")) {
      table.newUpdate()
    } else {
      null
    }

    ...
  })
Kudu In Our Architecture

... sqlContext.read.options(kuduOptions).format("org.apache.kudu.spark.kudu").load.
    registerTempTable("ny_taxi_trip_tmp")

    //Vector
    val vectorRDD:RDD[Vector] = sqlContext.sql("select * from ny_taxi_trip_tmp").map(r => {
        val taxiTrip = NyTaxiYellowTripBuilder.build(r)
        generateVectorOnly(taxiTrip)
    })

    println("--Running KMeans")
    val clusters = KMeans.train(vectorRDD, numOfCenters, numOfIterations)
    println(" > vector centers:")
    clusters.clusterCenters.foreach(v => println(" >> " + v))
...
Nested Structures
Nested

- Less Space than Denormalization
- Still have tables but the cost of joins is all but gone
- Also great for cartesian joins
  - $N \times M$ vs $N + M$
- Not really supported yet with Kudu or HBase with SQL
Nested Example

CREATE TABLE fact_contacts (id BIGINT, name STRING, address STRING) STORED AS PARQUET;
CREATE TABLE dim_phones
(
  contact_id BIGINT,
  category STRING,
  international_code STRING,
  area_code STRING,
  exchange STRING,
  extension STRING,
  mobile BOOLEAN,
  carrier STRING,
  current BOOLEAN,
  service_start_date TIMESTAMP,
  service_end_date TIMESTAMP
)
Nested Example

CREATE TABLE contacts_detailed_phones
(
    id BIGINT, name STRING, address STRING
    , phone ARRAY < STRUCT <
        category: STRING
        , international_code: STRING
        , area_code: STRING
        , exchange: STRING
        , extension: STRING
        , mobile: BOOLEAN
        , carrier: STRING
        , current: BOOLEAN
        , service_start_date: TIMESTAMP
        , service_end_date: TIMESTAMP
>>
)
) STORED AS PARQUET;

https://www.cloudera.com/documentation/enterprise/latest/topics/impala_complex_types.html
De-normalized vs Nested

- Nested Pros
  - Co-location
  - Faster to group by
  - Faster to window
  - Joins are free
  - Less data
  - Better compression
  - Tables and Columns can be read without penalty from one not read
  - Great for limiting the effort are Cartesian Joins

- Nested Cons
  - Size limitation of parent row
  - Adding child requires the re-write the whole parent record
Options for appending Nested

- It is all about the parent record
- We can add more than one Partition key for the parent
- In our use case
  - User & watch month or day
Storage and In Memory

- Also don’t limit the idea of nested to just tables
- In Spark they can be used as in memory constructs to
  - conserve on networking
  - In memory cost
Nested Writing Example in Spark

```json
{
  "id": "0001",
  "type": "donut",
  "name": "Cake",
  "ppu": 0.55,
  "batters":
  {
    "batter":
    [
      { "id": "1001", "type": "Regular" },
      { "id": "1002", "type": "Chocolate" },
      { "id": "1003", "type": "Blueberry" },
      { "id": "1004", "type": "Devil's Food" }
    ],
  },
  "topping":
  [
    { "id": "5001", "type": "None" },
    { "id": "5002", "type": "Glazed" },
    { "id": "5005", "type": "Sugar" },
    { "id": "5007", "type": "Powdered Sugar" },
    { "id": "5006", "type": "Chocolate with Sprinkles" }
  ]
}
```
Nested Writing Example in Spark

val jsonDF = hiveContext.read.json(jsonRDD)

jsonDF.write.parquet("./parquet")

hiveContext.createExternalTable("jsonNestedTable", "./parquet")
Nested In Our Architecture

```scala
hiveContext.sql("create table " + hdfsTaxiNestedTableName + "( " +
   " vender_id string," +
   " trip array<struct< " +
   "   passenger_count: INT," +
   "   payment_type: STRING, " +
   "   total_amount: DOUBLE, " +
   "   fare_amount: DOUBLE " +
   " >>" +
   " ) stored as parquet")

val emptyDf = hiveContext.sql("select * from " + hdfsTaxiNestedTableName + " limit 0")

hiveContext.createDataFrame(newNestedDf, emptyDf.schema).registerTempTable("tmpNested")

hiveContext.sql("insert into " + hdfsTaxiNestedTableName + " select * from tmpNested")
```

Time Series
Time Series Options

- HBase and Cassandra
Entity Centric Time Series

- Partition by Entity ID
- Order by Time
- Allows for free windowing
- Allows for fetching of single time window of single entity at web scale
HBase Entity Time Series
HBase Entity Time Series

Rest Call → Short Scan

- Cust-A, 10
- Cust-A, 20
- Cust-A, 40
- Cust-C, 10
- Cust-C, 20
- Cust-C, 30
- Cust-C, 40
- Cust-B, 10
- Cust-B, 20
- Cust-B, 30
- Cust-B, 40
- Cust-F, 20
- Cust-F, 30
- Cust-F, 40
- Cust-D, 10
- Cust-D, 20
- Cust-D, 40
- Cust-G, 10
- Cust-G, 20
- Cust-G, 30
- Cust-G, 40
HBase Entity Time Series
HBase Entity Time Series

Mapper

Cust-A, 10
Cust-A, 20
Cust-A, 40
Cust-C, 10
Cust-C, 20
Cust-C, 30
Cust-C, 40

Mapper

Cust-B, 10
Cust-B, 20
Cust-B, 30
Cust-B, 40
Cust-F, 20
Cust-F, 30
Cust-F, 40

Mapper

Cust-D, 10
Cust-D, 20
Cust-D, 40
Cust-G, 10
Cust-G, 20
Cust-G, 30
Cust-G, 40
What is meant by Bucketing and Sorting

- Partitioning on a Key
- Then sorting on that key + another field(s)
- Example
  - User_id + Watch Event Time
Example of Bucketed Sorted

Cust-A, 10
Cust-A, 20
Cust-A, 40
Cust-C, 10
Cust-C, 20
Cust-C, 30
Cust-C, 40
Cust-B, 10
Cust-B, 20
Cust-B, 30
Cust-B, 40
Cust-F, 10
Cust-F, 20
Cust-F, 40
Cust-D, 10
Cust-D, 20
Cust-D, 40
Cust-G, 10
Cust-G, 20
Cust-G, 30
Cust-G, 40
Good for Appendixing Nested Data

New Data
- Cust-A, 50
- Cust-A, 60
- Cust-B, 50
- Cust-B, 60
- Cust-C, 50
- Cust-D, 50
- Cust-G, 50

Existing Data
- Cust-A, 10
- Cust-A, 20
- Cust-A, 40
- Cust-B, 10
- Cust-B, 20
- Cust-B, 30
- Cust-B, 40
- Cust-C, 10
- Cust-C, 20
- Cust-C, 30
- Cust-C, 40
- Cust-F, 10
- Cust-F, 20
- Cust-F, 40
- Cust-G, 10
- Cust-G, 20
- Cust-G, 30
- Cust-G, 40
Good for Appending Nested

New Data

Cust-A, 50
Cust-A, 60
Cust-B, 50
Cust-B, 60
Cust-C, 50
Cust-D, 50
Cust-G, 50

Existing Data

Cust-A, 10
Cust-A, 20
Cust-A, 40
Cust-C, 10
Cust-C, 20
Cust-C, 30
Cust-C, 40
Cust-B, 10
Cust-B, 20
Cust-B, 30
Cust-B, 40
Cust-F, 10
Cust-F, 20
Cust-F, 40
Cust-D, 10
Cust-D, 20
Cust-D, 40
Cust-G, 10
Cust-G, 20
Cust-G, 30
Cust-G, 40

Shuffle Join
Good for Appending Nested

New Data

Existing Data

Cust-A, 50
Cust-A, 60
Cust-C, 50

Cust-B, 50
Cust-B, 60
Cust-G, 50

Cust-D, 50
Cust-C, 10
Cust-C, 20
Cust-C, 40

Cust-A, 10
Cust-A, 20
Cust-A, 40

Cust-B, 10
Cust-B, 20
Cust-B, 30
Cust-B, 40

Cust-D, 10
Cust-D, 20
Cust-D, 40

Cust-G, 10
Cust-G, 20
Cust-G, 30
Cust-G, 40

Cust-D, 50
Cust-C, 50

Merge Join

Questions?
tiny.cloudera.com/nyquestions
Good for Appending Nested

Questions?
tiny.cloudera.com/nyquestions
What else could be use Bucketing and Sorting For

- Windowing
- Point retrieval
Bucketed & Sorted for Windowing

Questions?
tiny.cloudera.com/nyquestions
Bucketed Sorted in a NoSQL

Rest Call -> Short Scan

- Cust-A, 10
- Cust-A, 20
- Cust-A, 40
- Cust-C, 10
- Cust-C, 20
- Cust-C, 30
- Cust-C, 40
- Cust-B, 10
- Cust-B, 20
- Cust-B, 30
- Cust-B, 40
- Cust-F, 10
- Cust-F, 20
- Cust-F, 40
- Cust-D, 10
- Cust-D, 20
- Cust-D, 40
- Cust-G, 10
- Cust-G, 20
- Cust-G, 30
- Cust-G, 40
NoSQL

- Columnar
What is a NoSQL

- It’s not NO SQL
- It’s not a Database
- Think of it more like a
  - HashMap
  - Log
  - Bucketed and Ordered
Hash Map

- There is a Key and a Value
- It is really fast to grab a key/value
- It is really fast to add a key/value
- Iteration is also possible

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
</tr>
</tbody>
</table>
Log with Compactions

- When new records come in they don’t rewrite the old
- They compact in

<table>
<thead>
<tr>
<th>Key</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>103</td>
</tr>
<tr>
<td>H</td>
<td>3</td>
<td>103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>103</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>H</td>
<td>3</td>
<td>103</td>
</tr>
</tbody>
</table>
Log with Compactions

- Write Path
  - Get Local for Record (Cached)
- First to WAL
- Then to Memstore
  - Sorting & batching
- Flush to New Hfile
- Later Hfiles will be compacted
Ordered

- All Records Columns are ordered
- Ordering allows for simpler indexing
- Ordering allows for simpler compactions
- We will also use this ordering
  - Windowing
  - Time series
  - Local scanning
Bucketing or Partitions

- HBase
  - Out of the Box:
    - Range
    - Desired:
      - Salt
  - Cassandra
    - Out of the Box:
      - HashMod
      - Bucketed HashMod
So what about SQL

- Well SQL could totally work
- CQL for cassandra
- Hive and SparkSQL on HBase
- Why is it not the best idea
  - Built more for point look ups
  - Scans are not as fast as parquet
  - However the mutability may be more important than speed
- Partitioning is not simple
  - It must be put into the key
Let’s talk about CAP for a Minute

- Strong Consistency
- HBase & Kudu
- Variable Consistency
- Cassandra
HBase Model

- Region Server owns range splits
- Region Server 1 fails
- Master needs to figure that out
- Master needs to assign new Region Server to own splits
- Region Server 2 has to get organized
- Region Server 2 is read to server reads and writes
Cassandra Model

Client

Replica Node (Random Node)

Replica Node (Has Replica)

Replica Node (Has Replica)

Replica Node (Has Replica)

Replica Node (Has Replica)

Questions?
tiny.cloudera.com/nyquestions
Cassandra Model
Cassandra Model (Common Models)

### 3 Write - 1 Read

- Client
- Replica Node (Has Replica)
- Client
- Replica Node (Has Replica)
- Client
- Replica Node (Has Replica)

### 1 Write - 3 Read

- Client
- Replica Node (Has Replica)
- Client
- Replica Node (Has Replica)
- Client
- Replica Node (Has Replica)

### 1 Write - 1 Read

- Client
- Replica Node (Has Replica)
- Client
- Replica Node (Has Replica)
- Client
- Replica Node (Has Replica)
NoSQL - Others

- Document
  - Mongo
  - CouchBase
- Spanner-Inspired
  - Kudu
  - CockroachDB
  - Druid.IO
NoSQL - Transitions

- Some have them
- Think about Kafka
Indexed Search
Lucene Indexing (Features)

- We don’t have enough time in this whole class
  - Ordering logic
  - NGrams
  - Weights
  - Text Indexing
  - Translations
  - Facets *
Lucene Indexing (Facets)

- Facets are a side effect of our wonderful indexes
- It allows us to count all the documents that belong to given indexes to produce
  - Grouped Counts
  - Charts and Graphs (kibana or Banana)
- People will also call this access pattern cubing a dataset
Lucene Indexing (Kibana & Banana)
## Lucene Indexing (Facets Example)

### Time Series Example

<table>
<thead>
<tr>
<th>Document ID</th>
<th>Hour of Day</th>
<th>User</th>
<th>State</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>4201</td>
<td>MD</td>
<td>click</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>4202</td>
<td>VA</td>
<td>click</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>4203</td>
<td>VA</td>
<td>click</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4201</td>
<td>MD</td>
<td>click</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4202</td>
<td>VA</td>
<td>view</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4204</td>
<td>CA</td>
<td>click</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4205</td>
<td>VA</td>
<td>view</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4201</td>
<td>MD</td>
<td>click</td>
</tr>
</tbody>
</table>
# Lucene Indexing (Facets Example)

<table>
<thead>
<tr>
<th>Document ID</th>
<th>Hour of Day</th>
<th>User</th>
<th>State</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>4201</td>
<td>MD</td>
<td>click</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>4202</td>
<td>VA</td>
<td>click</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>4203</td>
<td>VA</td>
<td>click</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4201</td>
<td>MD</td>
<td>click</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4202</td>
<td>VA</td>
<td>view</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4204</td>
<td>CA</td>
<td>click</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4205</td>
<td>VA</td>
<td>view</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4201</td>
<td>MD</td>
<td>click</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>4204</td>
<td>CA</td>
<td>click</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hour of Day</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MD</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>VA</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>CA</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4201</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4202</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4203</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4204</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4205</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>click</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>view</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions?
[ tiny.cloudera.com/nyquestions ]
Lucene Indexing (Facets Example)

- Events per hour
- Simple array count

<table>
<thead>
<tr>
<th>Hour of Day</th>
<th>12</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
Lucene Indexing (Facets Example)

- Events per hour by State
  - Simple array count

<table>
<thead>
<tr>
<th>Hour of Day</th>
<th>12</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>MD</th>
<th>VA</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>VA</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>CA</td>
<td>6</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Questions?
tiny.cloudera.com/nyquestions
Lucene Indexing (Facets Example)

- Note the bucketing and ordered pattern

<table>
<thead>
<tr>
<th>Hour of Day</th>
<th>MD</th>
<th>VA</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hour of Day 2</th>
<th>State MD</th>
<th>State VA</th>
<th>State CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lucene Indexing (Facets Example)

- Note the bucketing and ordered pattern

<table>
<thead>
<tr>
<th>Hour of Day 2</th>
<th>State MD</th>
<th>State VA</th>
<th>State CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hour of Day 2</th>
<th>State MD</th>
<th>State VA</th>
<th>State CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

+1 CA
Lucene Indexing (Facets Example)

Note the bucketing and ordered pattern

<table>
<thead>
<tr>
<th>Hour of Day 2</th>
<th>State MD</th>
<th>State VA</th>
<th>State CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

+1 VA

<table>
<thead>
<tr>
<th>Hour of Day 2</th>
<th>State MD</th>
<th>State VA</th>
<th>State CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

+1 MD

<table>
<thead>
<tr>
<th>Hour of Day 2</th>
<th>State MD</th>
<th>State VA</th>
<th>State CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

+1 CA

Questions?
tiny.cloudera.com/nyquestions
Partitioning

- SolR and Elastic Search partition the document o land on all nodes
- This means
  - You have the power of the cluster when querying
  - This mean you are accessing the cluster when querying
Writing Latency

- Lucene Indexing is more expensive than NoSQL work
- Think of it as micro batching
  - Larger batches ~= better throughput
- Compaction is also invalid
- Deletes impact storage and performance until they are compacted
Storage Cost

- TTL is your friend
- Think of Lucene based systems as great if
  - You dataset is manageable in size
  - You have a good TTL strategy
  - You have a boat load of money
Thinking in terms of Graphs
- Nodes and Edges
Thinking in terms of Graphs

- Use cases
- Querying
  - Cassandra with Sparkle
  - Neo4j
- Batch operations
  - Giraph
  - GraphX
  - GraphLab
BSP Bulk Synchronous Parallel

- Process every Node Atomically
- Node gets all messages sent to it
- Nodes can mutate themselves and their edges
- Nodes can send messages to other nodes
  - But nothing is received yet
- BSP waits until all the Node processing is done
- Then send messages to the right partition
- Repeat
High level architecture

Source: Kafka
Transport: Kafka
Stream Processing: Spark Streaming
Storage: HBase, Apache Solr, Jetty
Access: Spark, Spark, Hue
Batch Processing

Considerations
High level architecture

Source: Kafka
Transport: Kafka
Stream Processing: Spark Streaming
Storage:
- Nested Tables
- Relational Tables
- Entity Time Series Lookup
- Indexed Cube
Access:
- Batch Processing
- SQL
- NRT REST
- NRT Dashboard

Questions?
tiny.cloudera.com/nyquestions
Why have batch processing?

- When you need a larger context
  - Say, to train a model
- Complex periodic job that does something
  - Convert data to a nested structure for reduced number of shuffles
- In our use-case,
  - Kudu -> HDFS Nested is batch processing
  - KMeans calculation is also in bash
Batch processing options

- Spark (+ MLlib)
- MapReduce (+ Mahout)
- Flink (+ Flink ML)
Spark

- Pretty popular
- Much faster than MapReduce
- Thriving community
MapReduce

- Sloo0000ow
Flink

- Pretty popular
- Batch is a special case of Streaming
- Developing community
In our use-case

- We chose Spark
  - We were using Spark Streaming anyways
  - Similar code between Spark and Spark Streaming
  - Thriving community
Interactive Data Access Considerations
High level architecture

Source

Transport

Stream Processing

Storage

Access

kafka

kafka

Spark Streaming

Nested Tables

Relational Tables

Entity Time Series Lookup

Indexed Cube

Batch Processing

SQL

NRT REST

NRT Dashboard

Questions?

tiny.cloudera.com/nyquestions
Types of data access

- REST server/APIs for querying entities and aggregates
- UI for displaying search facets
- SQL engine
REST servers
Considerations
Why have REST server?

- Tired of business people telling us how to access data
- Serves as an interface between the data engineers and business folks
- Lets business folks decide access patterns
- Engineers to optimize those patterns
- Brownie points from your boss
- And, it’s not that difficult to write!
Don’t believe me?

```scala
import org.mortbay.jetty.Server
import org.mortbay.jetty.servlet.{Context, ServletHolder}
...

val server = new Server(port)
val sh = new ServletHolder(classOf[ServletContainer])
sh.setInitParameter("com.sun.jersey.config.property.resourceConfigClass", "com.sun.jersey.api.core.PackagesResourceConfig")
sh.setInitParameter("com.sun.jersey.config.property.packages", "com.hadooparchitecturebook.taxi360.server.hbase")
sh.setInitParameter("com.sun.jersey.api.json.POJOMappingFeature", "true")
val context = new Context(server, "/", Context.SESSIONS)
context.addServlet(sh, "/*")
server.start()
server.join()
```
Then, write a ServiceLayer

@GET
@Path("vender/{venderId}/timeline")
@Produces(Array(MediaType.APPLICATION_JSON))

def getTripTimeLine (@PathParam("venderId") venderId: String,
                     @QueryParam("startTime") startTime: String = Long.MinValue.toString,
                     @QueryParam("endTime") endTime: String = Long.MaxValue.toString):

    Array[NyTaxiYellowTrip] = {

Use REST! Say no to business people!

- Access data like so:

http://<serverURL>:8080/vendor/{vendorId}/timeline
UI Considerations
UI requirements

Something that can

▪ Represent search results really well
▪ Integrates with Apache Solr on Hadoop
UI options

- Hue
- Banana
- Kibana
We choose Hue

- Because it’s included
- Please look at the others
SQL engines
Considerations
SQL engine criteria

- Low latency SQL access
- Allows for high concurrency
- JDBC/ODBC integration
- Capable of large scale aggregation
- Optionally integrates with Kudu for real-time updates to SQL tables
Apache Hive

- Good JDBC integration
- Not really low latency, even when using Tez
- Doesn’t integrate with Kudu
- Can run with MapReduce, Spark, or Tez
Presto

- Low latency SQL engine from Facebook
- Provides JDBC/ODBC access
- Is only in-memory, large aggregations can lead to OOM errors
- Doesn’t integrate with Kudu
Apache Impala

- Low latency SQL access
- Provides JDBC/ODBC access
- Excellent concurrency support
- Integrates with Kudu for real-time SQL
Apache Drill

- Similar in architecture to Impala
- Provides JDBC/ODBC access
- Doesn’t integrate with Kudu
Spark SQL

- Builds on top of Spark
- JDBC/ODBC access only via Spark Thrift Server
  - Doesn’t scale well with larger number of concurrent users
  - Doesn’t fully provide secure access.
We choose

- Spark SQL
- Impala
Overall Architecture

Review
High level architecture

Source

Transport

Stream Processing

Processing & Ingestion Engine

Storage

Nested Tables

Relational Tables

Entity Time Series Lookup

Indexed Cube

Access

Batch Processing

SQL

NRT Rest

NRT Dashboard

#StrataData

Questions?

tiny.cloudera.com/nyquestions
High level architecture

Source

Transport

Stream Processing

Storage

Access

- Nested Tables
- Relational Tables
- Entity Time Series Lookup
- Indexed Cube

- Batch Processing
- SQL
- NRT REST
- NRT Dashboard
High level architecture

Source
- Kafka

Transport
- Kafka

Stream Processing
- Spark Streaming

Storage
- HBase
- Kudu
- Solr

Access
- Batch Processing
- SQL
- NRT REST
- NRT Dashboard

Questions?
tiny.cloudera.com/nyquestions
High level architecture

Source
- kafka

Transport
- kafka

Stream Processing
- Spark Streaming

Storage
- Apache HBase
- Apache Solr

Access
- Spark
- HUE
- jetty:

Questions?
tiny.cloudera.com/nyquestions
High level architecture

Source
- kafka

Transport
- kafka

Stream Processing
- Spark Streaming

Storage
- Spark
- Apache HBase
- Apache Solr

Access
- Jetty
- Hue

#StrataData
Questions?
tiny.cloudera.com/nyquestions
Where else to find us?

strataconf.com  
#StrataData
Other Sessions

- Ask Us Anything session – Thursday, 1:15 PM
- The Three Realities of Modern Programming: the Cloud, Microservices, and the Explosion of Data (Gwen) – Thursday 11:20 AM
- One Cluster Does Not Fit All: Architecture Patterns for Multicluster Apache Kafka Deployments (Gwen) – Thursday 2:05 PM
- Managing Successful Big Data Projects (Ted Malaska and Jonathan) – Thursday 4:35 PM
Thank you!

@hadooparchbook
tiny.cloudera.com/app-arch-newyork
Mark Grover | @mark_grover
Gwen Shapira | @gwenshap
Jonathan Seidman | @jseidman