Interactive SQL-on-Hadoop

from Impala to Hive/Tez to Spark SQL to JethroData

Ronen Ovadya, Ofir Manor,
JethroData

JETHRO DATA

About JethroData

- Founded 2012
- Raised funding from Pitango in 2013
- Engineering in Israel, branch in New York
- Launched beta July 2014
- We're hiring!

About Me

Ofir Manor

- Worked in the database industry for 20 years
- Started as developer and DBA
- Worked for Oracle, Greenplum pre-sales roles
- Blogging on Big Data and Hadoop
- Currently Product Manager at JethroData

ofir@jethrodata.com

http://ofirm.wordpress.com/

@ofirm

Agenda

- How we got here?
- How do parallel databases on Hadoop work?
- Impala, Hive/Tez, Spark SQL
- Use case perspective
- JethroData – What? How? Demo

SQL-on-Hadoop?

SQL?

"Stinger"

Apache Tez?

Big SQL

(too serious for a cute logo)

What about NoSQL on Hadoop?

What about Lucene / SOLR?

Sounds Familiar?

"Let's bring all the data that our operational systems create into one place, and keep it all forever.

When we'll analyze that repository, we will surely uncover critical business insights..."

What is this concept called?

- Today - "***Big Data***"
- Last 20 years - "***Enterprise Data Warehouse***" *(EDW)*

Big Data vs. Data Warehouse?

Everyone calls themselves “Big Data” now...

But “Big Data” is lead by large web companies with new requirements:

- **Web scale** – orders of magnitude more data
  - Page views and clicks, mobile apps, sensor data etc
- **Cost** – dramatically lower price per node
- **Aversion from vendor lock-in** - open-source preference
- **Methodology –** correctness vs. agility
  - **Classical EDW / Schema-on-write** – data must be cleaned and integrated before business users can access it (*EDW vs. ADW)*
  - **Big Data / schema-on-read**– let the data science team handle the unfiltered crap *(in addition to some vetted schema-on-write data sets)*

Evolution – 90s

- “Big Data” was “Enterprise Data Warehouse”
- You could either build your own using a big Unix machine and enterprise storage
  - “Scale Up”
- Or just buy a Teradata appliance

World's first production 1TB EDW, 1992

TERADATA *at* Walmart

Evolution – 2000s

- A new generation of parallel databases - ***"10x-100x faster, 10x cheaper"***
  - Netezza, Vertica, Greenplum, Aster, Paraccel etc
  - All used a cluster of cheap servers without enterprise storage to run parallel analytic SQL
  - *Shared-Nothing Architecture /*
    MPP (Massive Parallel Processing) /
    Scale-Out

Evolution – Our Days

- **It's all about Hadoop**
- Hadoop started as a batch platform - ***HDFS*** and ***MapReduce***
- Lately, it became a shared platform for any type of parallel processing framework

Example – Hortonworks slide

SQL-on-Hadoop

Underlining Design

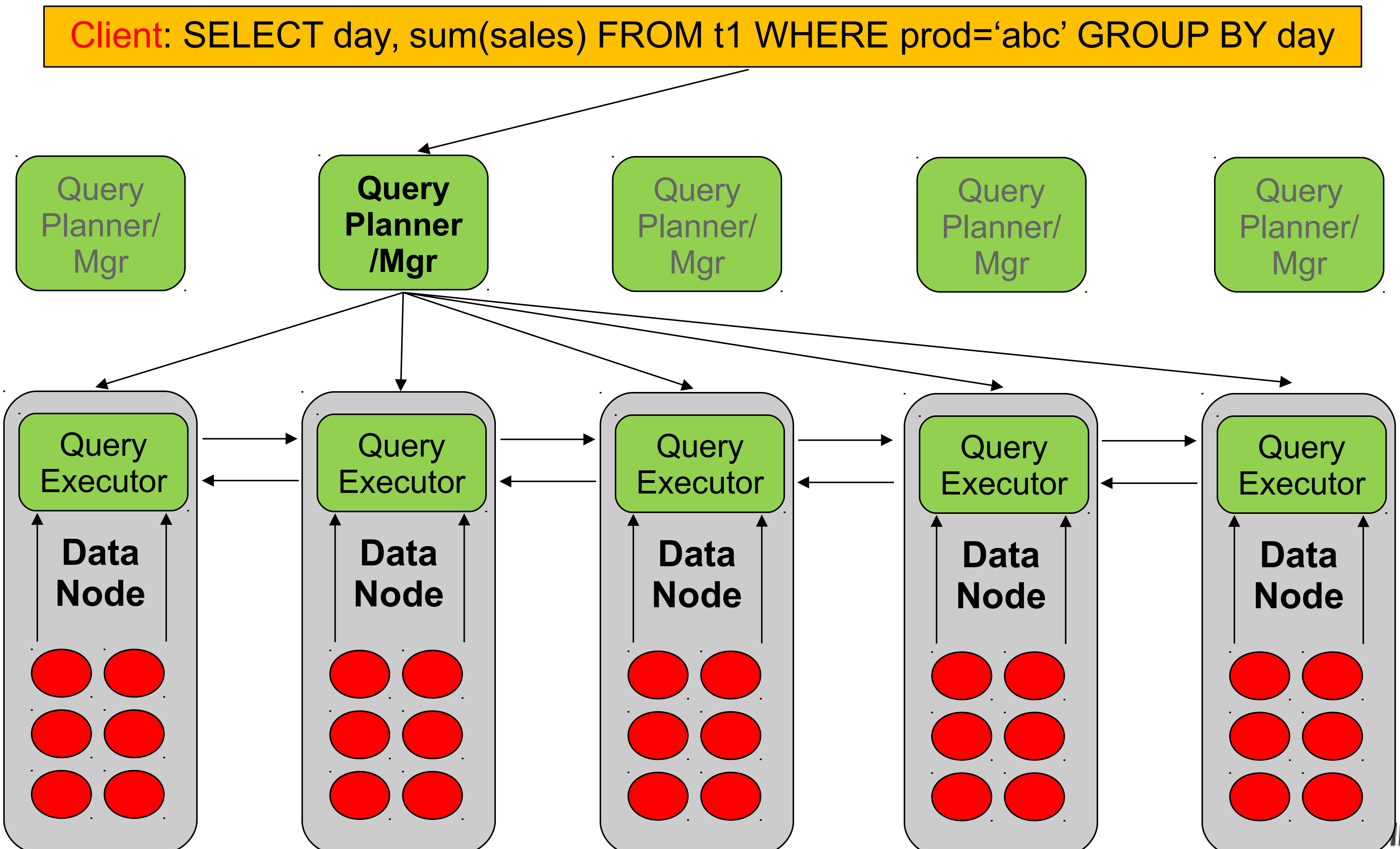
- Hadoop uses the same parallel design pattern as the parallel databases from last decade
- Surprisingly, all SQL-on-Hadoop also uses the same design pattern of previous parallel databases!

  - HDFS
  - MapReduce
  - Tez
  - Spark

  - Hive
  - Impala
  - Presto
  - Tajo
  - Drill
  - Shark
  - Spark SQL

  - Pivotal HAWQ
  - IBM BigSQL
  - Teradata Aster
  - Hadapt
  - Rainstor
  - ...

The Parallel Design
(Shared-Nothing MPP)



The Parallel Design Principles
(Shared-Nothing MPP)

- **Full Scan**
  - Each node reads all its local portion of the table
  - Optimize with large sequantial reads
- **Maximize Locality**
  - minimize inter-node work
  - Work should be evenly distributed to avoid bottlenecks
  - Avoid data and processing skew
- **That leads to a hard design-time trade-off**
  - Greatly depends the physical table organization
  - *Choose partition keys, distribution keys and sort keys*

JETHRO DATA

How to Get Decent Performance
Hive vs. Impala

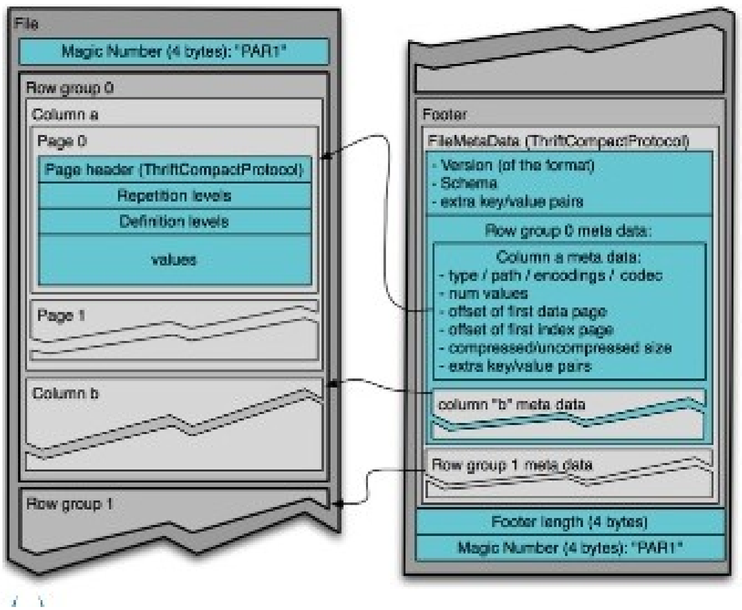
1. Minimize Query I/O
   - Skip reading unneeded data

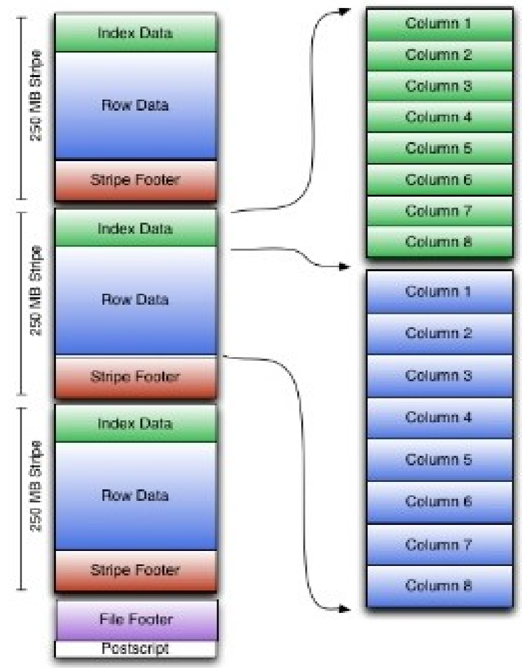
2. Optimize Query Execution
   - Pick best plan
   - Optimize each step
   - Connect the steps efficiently

1. Minimize Query I/O

- **Using Partitioning** (Hive, Impala etc)
  - Full scan of less data
  - Manual tuning – too few vs. too many
- **Using columnar, scan-optimized file formats**
  - "Write once, full scan a thousand times"
  - Skip unneeded columns
  - Full scan smaller files - encode and compress per-column
  - Skip blocks *(for sorted data)*
- Big effort in the Hadoop space, mostly done
  - Built two comparable formats - **ORC** and **Parquet**
  - Use the right one – Hive/ORC or Impala/Parquet

How Parquet and ORC columnar format works?





- Data is divided into blocks - chunks of rows
- In each block, data is physically stored column by column
- Store additional metadata per column in a block, like min / max values

2. Optimize Query Execution

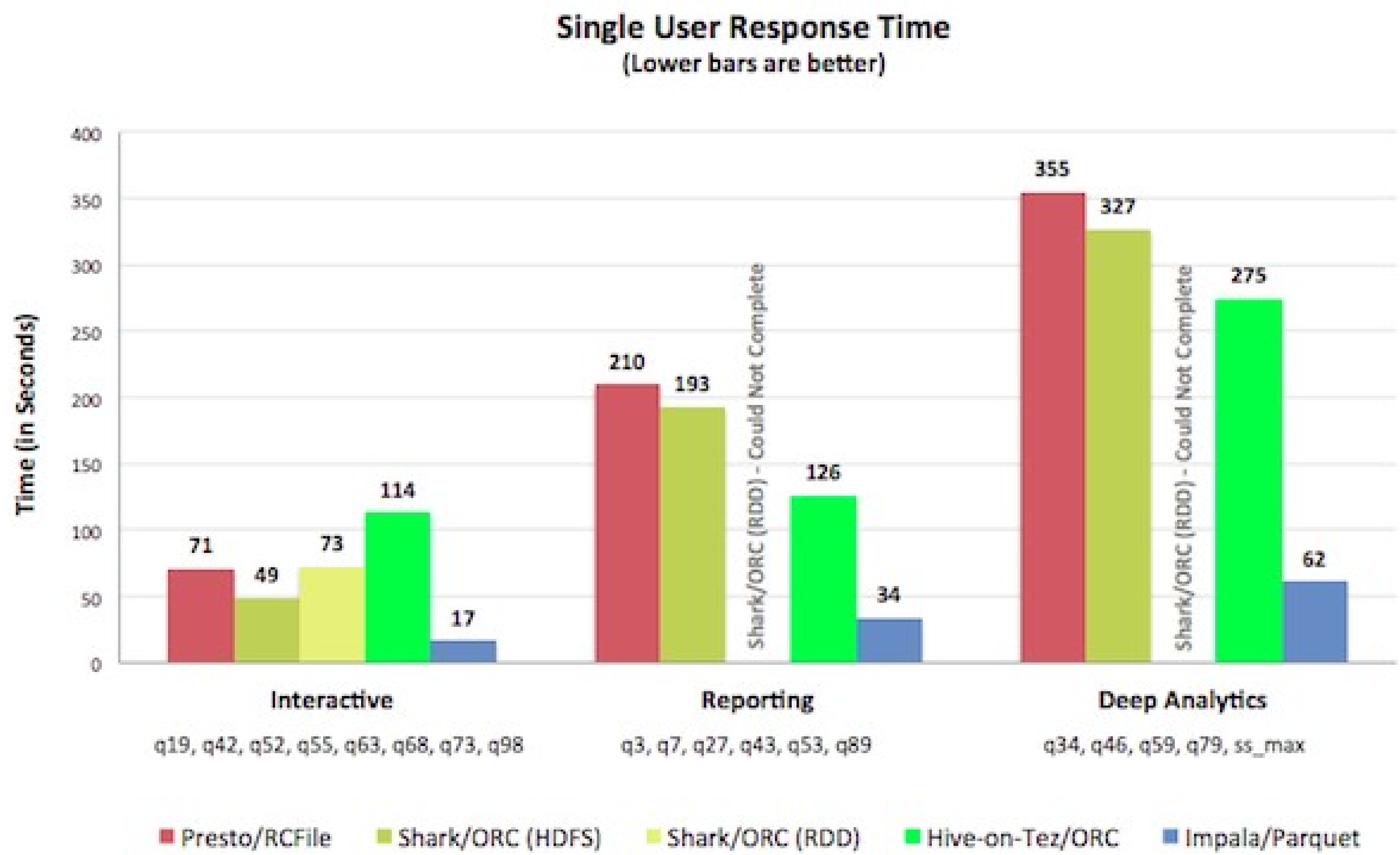
1. Pick the best execution plan – **Query Optimizer**
   - Cost-based Optimizations - currently generally weak
2. Optimize each step – **Efficient Processing**
   - Vectorized operations, expression compilation etc
3. Combine the steps efficiently - **Execution Engines**
   - <u>Batch-oriented</u> (*MapReduce*) – focus on recoverability
     - write intermediate results to disk after every step
   - <u>Streaming-oriented</u> (*Tez, Impala, HAWK etc*) – focus on performance
     - Move intermediate results directly between processes
     - Required much more resources at once
   - <u>Hybrid</u> (*Spark*) – enjoy both worlds
     - Stream and re-use intermediate results, optimize for in-memory
     - But can recover / recompute on node failure

JETHRO DATA

Impala

- ***Impala Highlights***
  - Basic partitioning (partition per key)
  - Optimized I/O with Parquet
  - Built their own streaming execution engine
  - Efficient processing
  - Basic cost-based query optimizations
- ***Impala vs. Presto vs. Drill vs. Tajo***
  - Generally same high-level design
  - Four independent teams implementing it in their own way
  - Impala way ahead of the pack
    - Performance, functionality, market share, support

Impala



- Sample benchmark results from Cloudera (May 2014)
- 20 nodes running TPCDS data, scale factor 15,000 (GB)
- *Impala 1.3.0* vs. *Hive 0.13 on Tez* vs. *Shark 0.9.2* vs. *Presto 0.6.0*

Source: Cloudera blog - New SQL Choices in the Apache Hadoop Ecosystem: Why Impala Continues to Lead

JETHRO DATA

Hive

- ***Hive Highlights***
  - Rich SQL support
  - Basic partitioning (partition per key)
  - Optimized I/O with ORC
  - Cost-based optimizer coming (Hive 0.14)
  - Efficient processing – vectorized query execution
  - Reliable execution (**MapReduce**) or fast execution (**Tez**)
  - Hive on Spark (**Shark**)
    - Have recently reached “End-of-life”, sort-of

Spark SQL

- ***Spark SQL Highlights***
  - Very early days / alpha state
    - Announced Mar 2014
  - A SQL-on-Spark solution written from scratch
  - Should support reading / writing from Hadoop, specifically from/to Hive and Parquet
  - Could become an interesting player next year
    - Mostly vs. Impala

Query Use Cases

Querying Raw Data

Raw Data

- Ad-hoc "Data Science" investigative work
  - Typically in the minutes to hours range
- Need to make sense of many, ever changing data sources
- Need to make sense of text / semi-structured / dynamic schema sources
- Need to mix SQL, text processing (UDFs) and machine learning algorithm in a manual, multi-step fashion

Query Use Cases

Reporting

- Internal analyst teams repeatedly run <u>reports</u> on common data sets
- Likely cleaned and vetted, potentially aggregated, shared across many users
- Use latest Hive/ORC/Tez or Impala/Parquet
  - Improves response time from hours to minutes

Query Use Cases

Pre-Compute Specific Queries (1)

- Queries embedded in external websites / apps / dashboards
  - Customers expecting page loads of a second or two
- **Must pre-compute results to get response time**
- <u>"Old Style"</u> implementation
  1. Daily massive batch computation (***MapReduce***)
  2. Results are typically pushed to an external OLAP solution (***cubes***) or to a key-value store (***HBase*** / Redis etc)
- Great performance but only for a few queries or counters
- "Freshness": data is typically a day or more behind production
- Complexity: multiple steps and systems, massive daily spike in computation, storage, data movement

Query Use Cases

Pre-Compute Specific Queries (2)

<u>“New Style”</u> implementation – **Stream Processing**

- Continuously update pre-computed results as new events arrive
  - Events pile up in some queue (***Apache Kafka***)
  - Process events in micro-batches (***Apache Storm***)
  - The new computation results are constantly stored / updated in a key-value store
- Solves the “freshness” problem
- Not suitable for complex pre-computations - best for counters
- Bleeding-edge technology

Query Use Cases

Fast Ad-hoc Queries

- **Interactive BI is the “Holy Grail” of SQL on Hadoop**
- Analysts / business users want to interact with select data sets from their BI tool
  - drag columns in the BI tool, “slice n' dice”, drill-downs
  - Response time of seconds to tens of seconds from their BI tool
- Existing solutions are too slow – users are stuck with reporting
- Very hard to achieve with existing Hadoop technologies
- Need a different solution – maybe something that have worked for databases in the last 30 years?
- It's time to introduce **JethroData**...

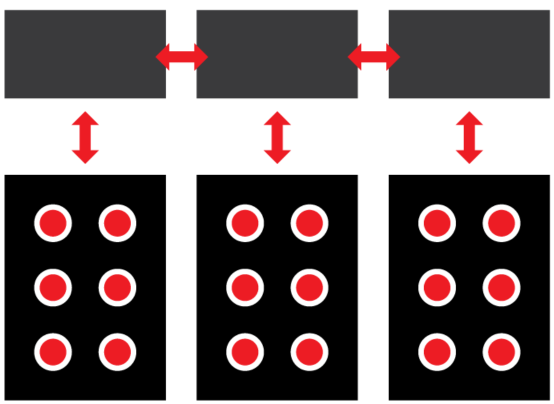
JethroData

- **Index-based, columnar SQL-on-Hadoop**
  - Delivers interactive BI on Hadoop
  - Focused on ad-hoc queries from BI tools
  - The more you drill down, the faster you go
- **SQL-on-Hadoop**
  - Column data and indexes are stored on HDFS
  - Compute is on dedicated edge nodes
    - Non-intrusive, nothing installed on the Hadoop cluster
    - Our own SQL engine, not using MapReduce / Tez
  - Supports standard ANSI SQL, ODBC/JDBC

JethroData
Working Differently

Full Scan / Brute Force
(All SQL-on-Hadoop Solutions)

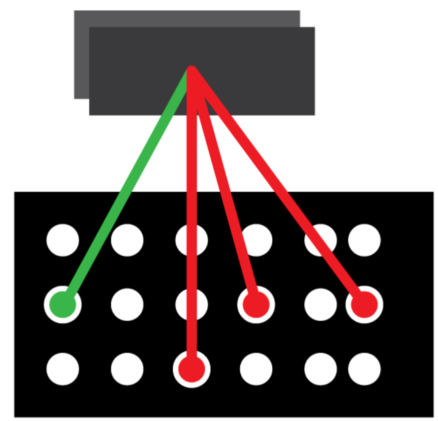
1. Read entire dataset. Every time.



- *Massive # of unnecessary I/Os*
- *Increasing demands on CPU and memory*

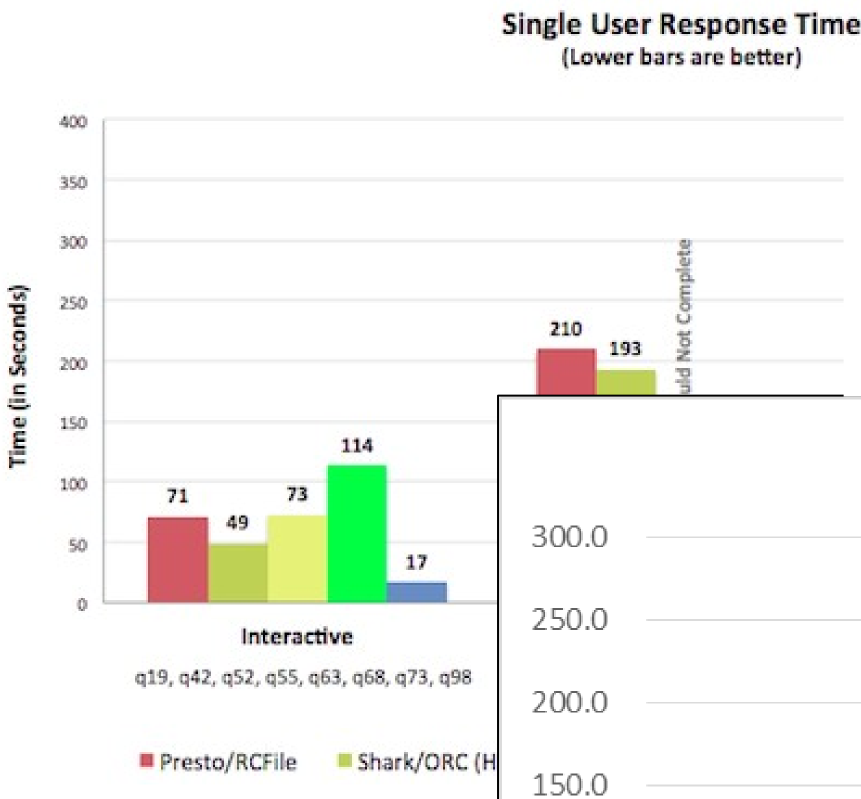
Index
(JethroData)

1. Analyze index
2. Fetch only needed data

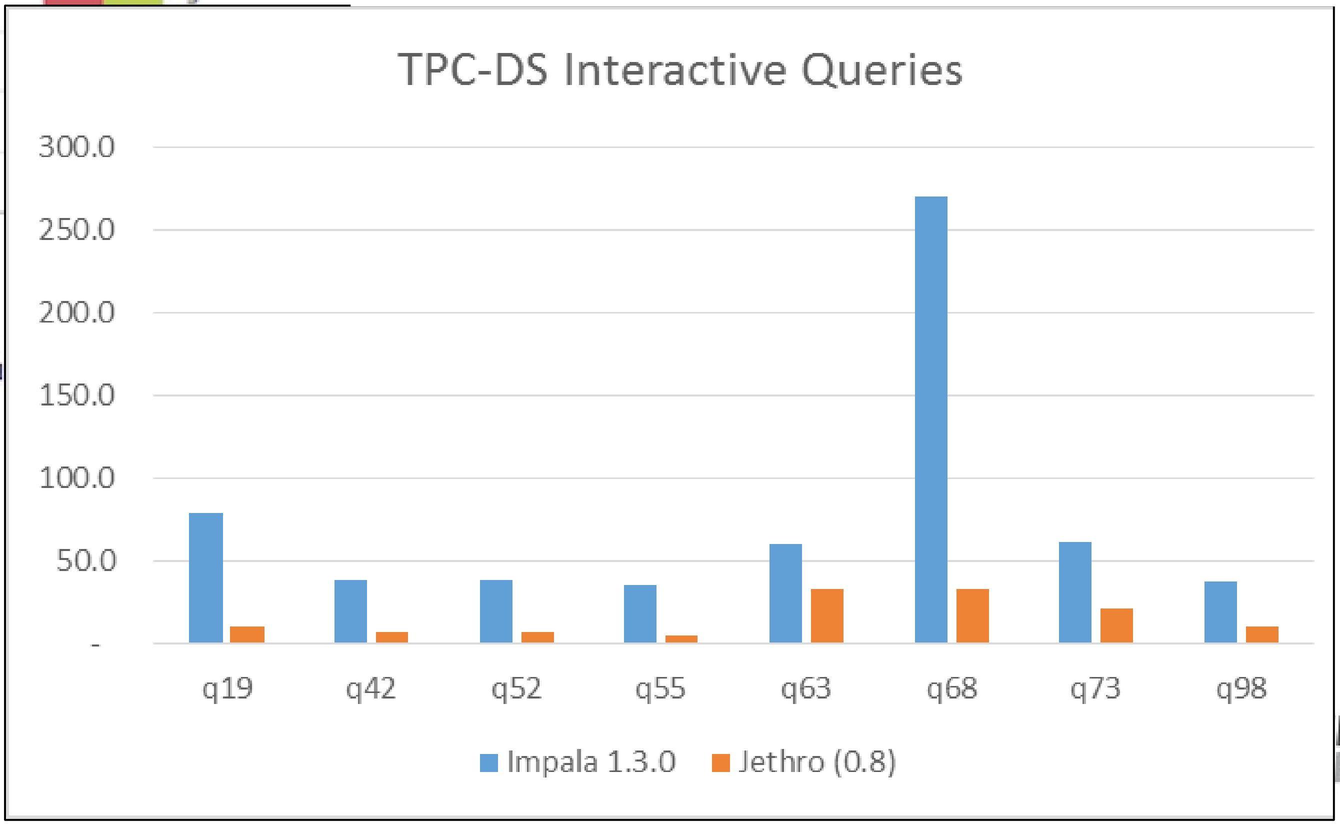


- *Drastically lower cluster load*
- *Low I/O CPU and memory usage*

Comparing to Impala



- Using Cloudera's benchmark (data and SQLs)
- Scale Factor 1000(GB)
- Running on AWS



Demo 1

Access JethroData from a remote JDBC client - SQL Workbench

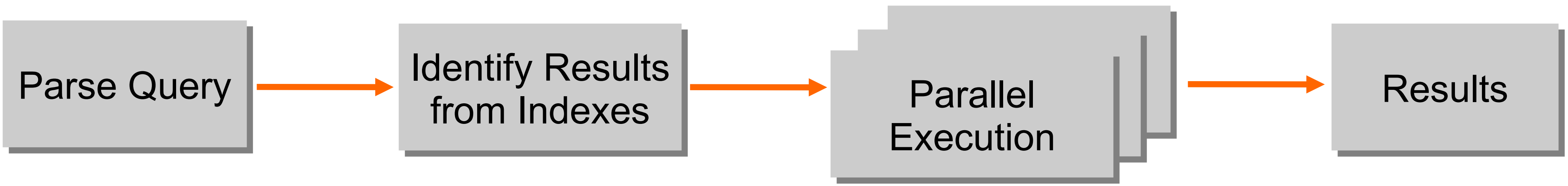
Small CDH5 cluster on AWS

Jethro Indexes

Value	Rows
FR	rows 5,9,10,11,14
IL	rows 1,3,7,12,13
US	rows 2,4,6,8,15

- Indexes map each column value to a set of rows
- Jethro stores indexes as hierarchical compressed bitmaps
  - Very fast query operations – `AND / OR / NOT`
  - Can process the entire `WHERE` clause to a final list of rows.
  - <u>Fully indexed</u> – all columns are automatically indexed
- **INSERT Performance**
  - Jethro Indexes are append-only
  - If needed, a new, repeating entries are allowed
  - `INSERT` is very fast – files are appended, no random read/write
  - Compatible with HDFS
  - Periodic background merge (non-blocking)

Jethro Execution Logic



Index Phase

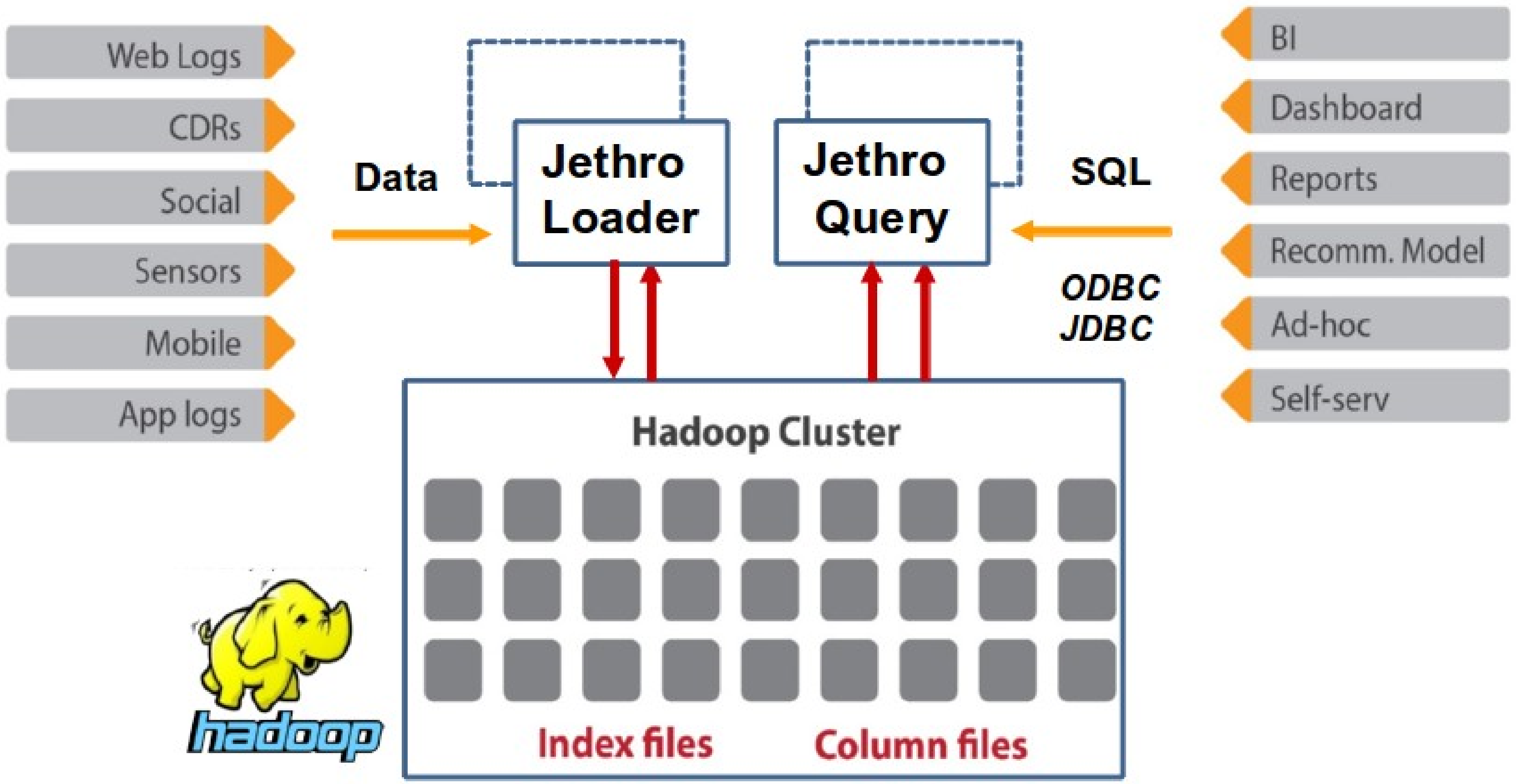
Use all relevant indexes to compute which rows are needed for query results

Divide the work to multiple parallel units of work.

Parallel Execution Phase

- Parallelize both fetch and compute
- Works with or without partitions
- Currently multi-threaded in a single server, designed for multi-node execution

Technical Architecture



Minimize Query I/O
part 1

- Automatically **combine all relevant indexes** for query execution – dramatically reduce fetch I/O
  - Generate a bitmap of all relevant rows after the entire WHERE clause
  - Skip indexes when their value is minimal (`WHERE a>0`)
    or when indexes are not applicable (`WHERE` *`function`*`(col)=5`)
- Use **Jethro columnar file format** to:
  - Skip columns
  - Encode and compress data (smaller files)
  - Skip blocks *(when a column must be scanned)*
- **HDFS I/O optimizations**
  - Optimized fetch size for bulk fetches
  - Using skip scans to convert random reads to single I/O, if possible

Minimize Query I/O
part 2

- **Automatic Local Cache**
  - Simple to set up
  - Metadata is automatically cached locally by priority, in the background
  - Can ask to cache specific columns or tables
  - Improves latency and reduces HDFS roundtrips
- **Use partitioning?**
  - Only small effect – we only read relevant rows, with or without partitioning
  - At the high-end, helps operating on smaller bitmaps
  - Mostly for rolling window operations

Optimize Query Execution

- **Query Optimizer** - pick the best plan (set of steps)
  - Using up-to-date detailed statistics from the indexes
  - For example, <u>star transformation</u>
- **Efficient Processing** – optimize each step
  - Efficient bitmap operations
- **Execution Engines** – combine the steps efficiently
  - Multi-threaded, streaming, parallel execution
  - Parallelize everything:
    Fetch, Filter, Join, Aggregate, Sort etc

Scalability
scale to 100s of billions of rows

- Scale out **HDFS** (Hadoop Cluster), if needed
  - Provide extra storage or extra I/O performance (rare)
- Scale out **Jethro nodes**
  - Jethro query nodes are stateless
  - Add nodes to support additional concurrent queries
- Leverage **partitioning**
  - Support rolling window maintenance at scale
  - Works best with a few billion rows per partition – usually partition for maintenance, not performance

High-Availability

- Leverage all **HDFS HA** goodies
  - DataNode Replication
  - NameNode HA
  - HDFS Snapshots
  - Any HDFS DR solution
- All Jethro nodes are **stateless**
  - Service auto-start on restart
  - Can start and stop them on demand

Demo 2

Access JethroData from Tableau Server over ODBC

Small CDH5 cluster on AWS

Questions?

Talk to us:

ronen@jethrodata.com

ofir@jethrodata.com

Join our beta!

www.jethrodata.com