SnappyData

A Unified Cluster for Streaming, Transactions, & Interactive Analytics

Version 0.7 | © Snappydata Inc 2017

Barzan Mozafari | Jags Ramnarayan | Sudhir Menon
Yogesh Mahajan | Soubhik Chakraborty | Hemant Bhanawat | Kishor Bachhav

www.Snappydata.io
Mixed Workloads Are Everywhere

- **Transaction**
  - Maintaining state or counters while ingesting streams

- **Stream Processing**
  - Correlating and joining streams with large histories

- **Interactive Analytics**
  - Analytics on mutating data
Mixed Workloads Are Everywhere

**Enrich**
- Requires reference data join
- Reference DB

**Data-in-motion Analytics**
- Interactive OLAP queries
- Updates

**Scalable Store**
- Transaction writes
- Models
- State – manage windows with mutating state
- Analytics – Joins to history, trend patterns
- Model maintenance

**HDFS, MPP DB**

**Application**

**IOT Devices**

**STREAMS**

**Alerts**
Why Supporting Mixed Workloads is Difficult?

Data Structures
- Columnar
- Row stores
- Sketches

Query Processing Paradigm
- Batch Processing
- Point Lookups
- Delta / Incremental

Scheduling & Provisioning
- Long-running
- Short-lived
- Bursty
Lambda Architecture

New Data

1. New Data

2. Batch layer
   - Master Datasheet

3. Serving layer
   - Batch view
   - Batch view

4. Speed layer
   - Real-time View
   - Real-time View

5. Query

Diagram shows the Lambda Architecture with three layers: Batch layer, Serving layer, and Speed layer. New data flows into the Batch layer, which then feeds into the Serving layer and Speed layer. Queries can be made to both the Serving layer and the Speed layer.
Lambda Architecture is Complex

- Scalable Store: HDFS, MPP DB
- Data-in-motion Analytics
- Application
- Interactive Queries
- References DB
- Updates
- Alerts
- Models
- NoSQL: Cassandra, Redis, Hbase, ...
- Enterprise DB: Oracle, Postgres, ...
- IOT Devices
- Storm, Spark Streaming, Samza, ...
- Teradata, Greenplum, Vertica, ...
- Teradata, Greenplum, Vertica, ...
- Cassandra, Redis, Hbase, ...
- Oracle, Postgres, ...
- Storm, Spark Streaming, Samza, ...
Lambda Architecture is Complex

- **Complexity**
  - Learn and master multiple products, data models, disparate APIs & configs

- **Wasted resources**

- **Slower**
  - Excessive copying, serialization, shuffles
  - Impossible to achieve interactive-speed analytics on large or mutating data
Can We Simplify & Optimize?
Our Solution: SnappyData

Single Unified HA Cluster
OLTP + OLAP + Streaming for Real-time Analytics

Spark
Batch design, high throughput
Rapidly Maturing

Real-time design
Low latency, HA, concurrency
Matured over 13 years
We Transform Spark from This ...

USER 1 / APP 1

Spark Master

Spark Executor (Worker)
Framework for streaming SQL, ML...
Immutable CACHE

USER 2 / APP 2

Spark Master

Spark Executor (Worker)
Framework for streaming SQL, ML...
Immutable CACHE

HDFS
SQL
NoSQL

• Cannot update
• Repeated for each User/App

Bottleneck
... Into an “Always-On” Hybrid Database!

- **JVM**: Spark Executor (Worker)
  - Long-lived

- **Framework for streaming SQL, ML...**

- **Spark Driver**

- **Store**
  - In-Memory ROW + COLUMN
  - Start with Indexing
  - Mutable, Transactional

- **Spark Cluster**

- **ODBC**

- **JDBC**

- **Shared Nothing Persistence**

- **HISTORY**

- **HDFS SQL NoSQL**
Snappy Data Server (Spark Executor + Store)

Unified Data Model & API

- Mutability (DML+Trx)
- Indexing
- SQL-based streaming

HYBRID Store

- Probabilistic
- Rows
- Index
- Columns

Data Synopsis Engine

Distributed Membership Service

OLAP

Query Optimizer

Parser

Tables

ODBC/JDBC

Cluster Manager & Scheduler

Add / Remove Server

Stream Processing

RDD

Data Frame

Low Latency

High Latency
Overview

Cluster Manager & Scheduler

Snappy Data Server (Spark Executor + Store)
Hybrid Store

- Unbounded Streams
- Ingestion
- Transactional State Update
- Real-time Sampling
- Probabilistic
- Rows
- Index
- Column
- Random Writes
- (Reference data)
- OLAP
- Stream Analytics
Updates & Deletes on Column Tables

- Row Buffer
- MVCC
  - WRITE
  - Replicate for HA

New Segment

- Periodic Compaction
- One Partition

Time

Column Segment (t1-t2)

Column Segment (t2-t3)

Summary Metadata

<table>
<thead>
<tr>
<th>K11</th>
<th>C11</th>
<th>C21</th>
</tr>
</thead>
<tbody>
<tr>
<td>K12</td>
<td>C12</td>
<td>C22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Probabilistic Store: Synopses + Uniform & Stratified Samples

1. Streaming CMS (Count-Min-Sketch)

Higher resolution for more recent time ranges
Probabilistic Store: Synopses + Uniform & Stratified Samples

1. **Streaming CMS**  
   *(Count-Min-Sketch)*  
   Higher resolution for more recent time ranges

   ![Diagram of traditional CMS vs CMS+Samples]

2. **Top-K Queries w/ Arbitrary Filters**  
   Maintain a small sample at each CMS cell
Probabilistic Store: Synopses + Uniform & Stratified Samples

1. Streaming CMS (Count-Min-Sketch)

Higher resolution for more recent time ranges

2. Top-K Queries w/ Arbitrary Filters

Maintain a small sample at each CMS cell

3. Fully Distributed Stratified Samples

Always include timestamp as a stratified column for streams
Overview

Snappy Data Server (Spark Executor + Store)

HYBRID Store

- Probabilistic
- Rows
- Columns

Data Synopsis Engine

OLAP

TRX

Query Optimizer

Parser

Tables

Index

ODBC/JDBC

Cluster Manager & Scheduler

Add / Remove Server

DataStream Processing

RDD

Stream Processing

Data Frame

Probabilistic Rows Columns

Low Latency

High Latency

H A

H A
• Spark Executors are long running. Driver failure doesn’t shutdown Executors

• Driver HA – Drivers are “Managed” by SnappyData with standby secondary

• Data HA – Consensus based clustering integrated for eager replication
Overview

Snappy Data Server (Spark Executor + Store)
Overview

Snappy Data Server (Spark Executor + Store)
Query Optimization

• Bypass the scheduler for transactions and low-latency jobs

• Minimize shuffles aggressively
  - Dynamic replication for reference data
  - Retain ‘join indexes’ whenever possible
  - Collocate and co-partition related tables and streams

• Optimized ‘Hash Join’, ‘Scan’, ‘GroupBy’ compared to Spark
  - Use more variables (eliminate virtual funcs) to generate better code
  - Use vectorized structures
  - Avoid Spark’s single-node bottlenecks in broadcast joins

• Column segment pruning through metadata
Co-partitioning & Co-location

Spark Executor

Subscriber A-M
KAFKA Queue

Subscriber N-Z
KAFKA Queue

Linearly scale with partition pruning

Spark Executor

Subscriber A-M
Ref data

Subscriber N-Z
Ref data
Overview

Snappy Data Server (Spark Executor + Store)
Approximate Query Processing (AQP): Academia vs. Industry

25+ yrs of successful research in academia
AQUA, Online Aggregation, MapReduce Online, STRAT, ABS, BlinkDB / G-OLA, ...

BUT:

User-facing AQP almost non-existent in commercial world!
Some approximate features in Infobright, Yahoo's Druid, Facebook's Presto, Oracle 12C, ...

WHY?
Approximate Query Processing (AQP): Academia vs. Industry

25+ yrs of successful research in academia
AQUA, Online Aggregation, MapReduce Online, STRAT, ABS, BlinkDB / G-OLA, ...

BUT:
User-facing AQP almost non-existent in commercial world!
Some approximate features in Infobright, Yahoo’s Druid, Facebook’s Presto, Oracle 12C, ...

WHY?

```
select geo, avg(bid)
from adImpressions
group by geo having avg(bid)>10
with error 0.05 at confidence 95
```

<table>
<thead>
<tr>
<th>geo</th>
<th>avg(bid)</th>
<th>error</th>
<th>prob_existence</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI</td>
<td>21.5</td>
<td>± 0.4</td>
<td>0.99</td>
</tr>
<tr>
<td>CA</td>
<td>18.3</td>
<td>± 5.1</td>
<td>0.80</td>
</tr>
<tr>
<td>MA</td>
<td>15.6</td>
<td>± 2.4</td>
<td>0.81</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Approximate Query Processing (AQP): Academia vs. Industry

25+ yrs of successful research in academia
AQUA, Online Aggregation, MapReduce Online, STRAT, ABS, BlinkDB / G-OLA, ...

BUT:
User-facing AQP almost non-existent in commercial world!
Some approximate features in Infobright, Yahoo’s Druid, Facebook’s Presto, Oracle 12C, ...

WHY?

select geo, avg(bid)
from adImpressions
group by geo having avg(bid)>10
with error 0.05 at confidence 95

<table>
<thead>
<tr>
<th>geo</th>
<th>avg(bid)</th>
<th>error</th>
<th>prob_existence</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI</td>
<td>21.5</td>
<td>± 0.4</td>
<td>0.99</td>
</tr>
<tr>
<td>CA</td>
<td>18.3</td>
<td>± 5.1</td>
<td>0.80</td>
</tr>
<tr>
<td>MA</td>
<td>15.6</td>
<td>± 2.4</td>
<td>0.81</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

1. Incompatible w/ BI tools
2. Complex semantics
3. Bad sales pitch!
A First Industrial-Grade AQP Engine

1. **Highlevel Accuracy Contract (HAC)**
   - User picks a single number \( p \), where \( 0 \leq p \leq 1 \) (by default \( p=0.95 \))
   - Snappy guarantees that s/he only sees things that are at least \( p \)% accurate
   - Snappy handles (and hides) everything else!

<table>
<thead>
<tr>
<th>geo</th>
<th>avg(bid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI</td>
<td>21.5</td>
</tr>
<tr>
<td>WI</td>
<td>42.3</td>
</tr>
<tr>
<td>NY</td>
<td>65.6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
A First Industrial-Grade AQP Engine

1. Highlevel Accuracy Contract (HAC)
   • User picks a single number \( p \), where \( 0 \leq p \leq 1 \) (by default \( p = 0.95 \))
   • Snappy guarantees that s/he only sees things that are at least \( p \% \) accurate
   • Snappy handles (and hides) everything else!

2. Fully compatible w/ BI tools
   • Set HAC behavior at JDBC/ODBC connection level
A First Industrial-Grade AQP Engine

1. Highlevel Accuracy Contract (HAC)
   - User picks a single number $p$, where $0 \leq p \leq 1$ (by default $p=0.95$)
   - Snappy guarantees that s/he only sees things that are at least $p\%$ accurate
   - Snappy handles (and hides) everything else!

2. Fully compatible w/ BI tools
   - Set HAC behavior at JDBC/ODBC connection level

3. Better marketing!
   - Concurrency: 10’s of queries in shared clusters
   - Resource usage: everyone hates their AWS bill
   - Network shuffles
   - Immediate results while waiting for final results

<table>
<thead>
<tr>
<th>geo</th>
<th>avg(bid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI</td>
<td>21.5</td>
</tr>
<tr>
<td>WI</td>
<td>42.3</td>
</tr>
<tr>
<td>NY</td>
<td>65.6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

iSight (Immediate inSight)
Benchmarks

• **Mixed Benchmark: Ad Analytics**

  - Ad impressions arrive on a message bus
  - Aggregate by publisher and geo
  - Report avg bid, # of impressions, and # of uniques every few secs
  - Write to a partitioned store
  - Transactionally update the profiles during ingestion
  - Q1: Top-20 ads receiving most impressions per region
  - Q2: Top-20 ads receiving largest bids per geo
  - Q3: Top-20 publishers receiving largest sum of bids overall

• **TPC - H**
Setup

- **HW:** 5 c4.2xlarge EC2 instances
  - 1 coordinator + 4 workers

- **Software:** Latest GA versions available:
  - Kafka 2.10_0.8.2.2
  - Spark 2.0.0
  - Cassandra 3.9
    - with Spark-Cassandra connector 2.0.0_M3
  - MemSQL Ops-5.5.10 Community Edition
    - with Spark-MemSQL Connector 2.10_1.3.3
  - SnappyData 0.6.1
Ad Analytics

1.5-2x faster ingestion
7-142x faster analytics (at 300M records)
Data Synopsis Engine

![Bar chart showing latency for analytical queries]

- **SnappyData**
- **SnappyData (approx)**

**Latency (sec)**

- Q1: Approximately 5 seconds
- Q2: Approximately 5 seconds
- Q3: Approximately 8 seconds

**Analytical Queries**

- Q1
- Q2
- Q3
SnappyData was 2.6x faster than MemSQL & 22.4x faster than Spark 2.0
Conclusion
Where Are We Today?

- **Current customers**
  - Investment banking, Industrial IoT, Telco, Ad Analytics, & healthcare

- **Current release**
  - 0.7 (GA 1.0 in Q1-2017)

- **Next funding round**
  - Q2-2017

- **Upcoming features**
  - Integration of Spark ML w/ our Data Synopsis Engine
  - Cost-based query optimizer
  - Physical designer & workload miner ([http://CliffGuard.org](http://CliffGuard.org))
Lessons Learned

1. A unique experience marrying two different breeds of distributed systems

lineage-based for high-throughput vs. (consensus-) replication-based for low-latency
Lessons Learned

1. A unique experience marrying two different breeds of distributed systems
   lineage-based for high-throughput vs. (consensus-) replication-based for low-latency

2. A unified cluster is simpler, cheaper, and faster
   - By sharing state across apps, we decouple apps from data servers and provide HA
   - Save memory, data copying, serialization, and shuffles
   - Co-partitioning and co-location for faster joins and stream analytics
Lessons Learned

1. A unique experience marrying two different breeds of distributed systems

   lineage-based for high-throughput vs. (consensus-) replication-based for low-latency

2. A unified cluster is simpler, cheaper, and faster

   - By sharing state across apps, we decouple apps from data servers and provide HA
   - Save memory, data copying, serialization, and shuffles
   - Co-partitioning and co-location for faster joins and stream analytics

3. Advantages over HTAP engines: Deep stream integration + AQP

   - Stream processing ≠ stream analytics
   - Top-k w/ almost arbitrary predicates + 1-pass stratified sampling over streams
Lessons Learned

1. A unique experience marrying two different breeds of distributed systems

   lineage-based for high-throughput vs. (consensus-) replication-based for low-latency

2. A unified cluster is simpler, cheaper, and faster

   - By sharing state across apps, we decouple apps from data servers and provide HA
   - Save memory, data copying, serialization, and shuffles
   - Co-partitioning and co-location for faster joins and stream analytics

3. Advantages over HTAP engines: Deep stream integration + AQP

   - Stream processing ≠ stream analytics
   - Top-k w/ almost arbitrary predicates + 1-pass stratified sampling over streams

4. Commercializing academic work is lots of work but also lots of fun
Try our iSight cloud for free: http://snappydata.io/iSight

THANK YOU!