High Performance Clickstream Analytics with Apache HBase/Phoenix

- CDK Global (formerly ADP Dealer Services)
About Me

• Arun Thangamani - Software Architect, BI
  • Past
    • Technical Lead, Monsanto
      • R&D – Technology Pipeline
    • Senior Software Engineer, Caliper
    • Software Engineer, Yahoo
    • University of Alabama – MS Computer Science
      • R&D, Space Research Labs
  • Arun.Thangamani@gmail.com, Arun.Thangamani@cdk.com
    • Twitter : @ArunThangamani

• Joined By
  • Chris Chang – Senior Manager, BI
  • Peter Huang – Director, BI

• CDK Team
  • Philippe Lantin, Nima Imani, Amit Phaltankar, Sundara Palanki, Pradip Thoke, Kim Yee, Shridhar Kasat, Clifton May, Bruce Szalwinski, Branden Makana, Anand Joglekar
Topics Overview

• About CDK Global
  ➢ Clickstream Analytics Use Case

• Architecture
  ➢ HBase Logical and Components Overview
  ➢ HBase and Phoenix Aggregation
  ➢ Timestamp Handling in HBase/Phoenix

• Fundamental Results Achieved
  ➢ Demo with Apache Zeppelin

• Phoenix – Scans, Joins and Secondary Indexes
  ➢ Current and Future of Clickstream Analytics Use Case

• Performance Optimization Variables
  ➢ Comparison Metrics
CDK Global (formerly ADP Dealer Services)

• Provide Integrated Technology Solutions to ~30,000 dealers across the world

• Dealers
  • Auto, Truck, Motorcycle, Marine, Recreational Vehicles and Heavy equipment

• For the purpose of this presentation, we are interested
  • *Dealer Web Sites and Clickstreams from web sites*

• CDK Overall Deals with Various Types Data including (not limited to)
  • Inventory, Sales, Services, Organization Data
  • Customers, Advertisement/Impressions Data
  • Auctions, Open Domain Data, Partner Programs
Clickstream Analytics – Use Case

• Widget Experience
Clickstream Analytics – Widget Experience Fundamentals

• Widget Experience
  • Webpage Widgets ‘react’ to User Intent → deliver experience

• Dealer Analytics
  • Effectiveness of Widget Experiences and Optimizations
  • 60 days worth of data
    • One day load => ~25M rows => ~1.5B rows total
    • Data intake in random chronographic order
  • Report => aggregate with relatively ‘light’ to ‘heavy’ filtering

• Challenges
  • Can we keep the report interactive and live?
  • How do we delete/expire data?
HBase – Quick Overview

- Column family oriented store
- Highly scalable with no central index
- Open source re-incarnation of BigTable
- Great for Aggregating large amounts of data
- Fully Consistent

HBase – Table
NOSQL Key Value Store

<table>
<thead>
<tr>
<th>PK</th>
<th>Column Family -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowkey-1</td>
<td>colname-1</td>
</tr>
<tr>
<td></td>
<td>colname-1</td>
</tr>
<tr>
<td></td>
<td>colname-2</td>
</tr>
<tr>
<td></td>
<td>colname-3</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>rowkey-2</td>
<td>colname-1</td>
</tr>
<tr>
<td></td>
<td>colname-5</td>
</tr>
<tr>
<td></td>
<td>colname-6</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Handle Billions of these with Ease?. How?
HBase Architecture

HBase Table – 1 to N regions

Region-1  Region-2
Region-3  Region-4

HBase Reads
- Connect to Zookeeper
- Find .Meta
- Cache Meta Data
- Find Region/Region-Server
- Check Memstore
- Check Block Cache
- Find HFiles with Bloom Filter
- Use Block Index to find Block
- Scan Block to find row
- Place Block in Block Cache

HBase Writes
- Connect to Zookeeper
- Find .Meta
- Cache Meta Data
- Find Region/Region-Server
- Add Data to Memstore & WAL
- Memstore Flush
- New HFile(s) created
- Major/Minor Compactions
- HFiles merged
- Recovery
- Lost HFiles from HDFS
- Replay WAL to Memstore

HBase Master
Create/ Delete/Split Table Operations

Client

First Call
Second Call
Third Call for Write
Third Call for Read

Zoo Keeper

Region Server1  Region Server2  Region Server3  Region Server4  Region Server5

HDFS
WAL
HDFS
1. For Range scans, block indexes are used
2. For specific row retrieves use Bloom Filter first

1. Instead of checking all HFiles in depth, look at Row Bloom Filter of each File
2. Assume - Bloom Filter has 3 way hash (usually N=>deterministic)
3. For any key, mark the relevant bucket on 3 hash’s
4. If all relevant hash buckets are not filled up, the key doesn’t exist
5. If all relevant hash buckets are filled up, the key might exist – high possibility
6. No false negatives, but false positives possible
7. More Hash’s and more hash buckets => more certainty
## Widget Semi Aggregated Table – Core Table

<table>
<thead>
<tr>
<th>dealer-user details</th>
<th>widget details</th>
<th>aggregate-stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>dealer id</td>
<td>page label</td>
<td>timestamp</td>
</tr>
</tbody>
</table>

### Primary Filter

### Final Group-by

### Primary Key

* Total size of table – 1.5 Billion

Dealer will have ~ [100,000 to 4,000,000] rows
HBase – Aggregation

1. HBase works on Ordered key Storage
2. Ordered key storage creates large/hot Regions for aggregation

2 ways to address the problem in Phoenix
a. Phoenix Guide Posts
b. Phoenix Salted Buckets
1. HBase works on Ordered key Storage
2. Ordered key storage creates large/hot Regions for aggregation

2 ways to address the problem in Phoenix
a. Phoenix Guide Posts
b. Phoenix Salted Buckets
Aggregation data retrieval – we have minimized the disk seeks and block transfers to a few per region server.
Phoenix – Overview

- Born at Salesforce by James Tylor & Mujtaba Chohan
- Phoenix works on top of HBase
- Puts back the SQL on top of HBase
- Phoenix makes HBase more usable with less code
Architecture – Phoenix

Client JDBC calls

HBase Master

Region Server1
Region Server2
Region Server3
Region Server4
Region Server5

Zoo Keeper
Zoo Keeper
Zoo Keeper

Phoenix
Phoenix
Phoenix
Phoenix
Phoenix

HDFS
HDFS

Create/Delete/Split Table Operations

Dynamic Coprocessors
Dynamic Observers

Coprocessors – execute aggregation
Observers – Various Types – observe to sync data
Timestamp Handling in HBase
### Timestamp and Expiration Handling

<table>
<thead>
<tr>
<th>row-key</th>
<th>cf1:c1</th>
<th>cf1:c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td></td>
<td>t1</td>
</tr>
<tr>
<td>r2</td>
<td></td>
<td>t3, t2, t1</td>
</tr>
<tr>
<td>r3</td>
<td></td>
<td>t1</td>
</tr>
<tr>
<td>r4</td>
<td></td>
<td>t3</td>
</tr>
</tbody>
</table>

**Table A**  
Time To Live = 3 days

**HFile: cf1**

- (r1:cf1:c1:t1:value)
- (r2:cf1:c2:t3:value)
- (r2:cf1:c2:t2:value)
- (r2:cf1:c2:t1:value)
- (r3:cf1:c2:t1:value)
- (r4:cf1:c1:t3:value)
- (r1:cf1:c1:t4:value)
- (r6:cf1:c2:t4:value)

Expired data is Auto Filtered On Retrieve

**Major Compaction**
Phoenix/HBase Implementation

- Choose Salt bucket tables
  - Compensate for ‘hot regions’
  - Parallel filtering in regions
  - Utilize parallel aggregation => regions aggregate before final merge in client

- Utilized the timestamp feature of HBase
  - Set Time to Live at Table level, entries are time-stamped
  - Expired data will be auto filtered during reads
  - Expired data deleted along with old HFiles during major compaction

- Total 5 Nodes – Appendix-B for full specifications
Demo – Fundamental Use Case
Visualize Widget Segment And Experience

```
%%phoenix
SELECT VIEW_DATE_TIMESTAMP, WIDGET_INSTANCE_ID, SUM(TOTAL_CLICK_VIEWS), SUM(TOTAL_TIME_ON_PAGE_MS), SUM(TOTAL_VIEWABLE_TIME_MS), SUM(TOTAL_HOVER_TIME_MS), SUM(VIEW_COUNT)
FROM WIDGET_PAGES_STATS
WHERE WEB_ID = 1
AND WEB_PAGE_LABEL in ('honespage')
--AND user_segment LIKE '%silverado%'
--AND widget_context like '%silverado%'
--AND widget_context like '%model%'
--AND VIEW_DATE_TIMESTAMP > 1453800000000
--AND widget_instance_id = 1400843758915
GROUP BY VIEW_DATE_TIMESTAMP, WIDGET_INSTANCE_ID
```

![Graph showing widget experience trends](image-url)
**Widget Experience Report Performance**

Random Dealer Id aggregation for 30 days of data – Warm Cache
30 days is our default report

Filtering on dealer specific attributes
Group-by on 4 different Widget Attributes (relatively high return data)

Total Region Servers – 5
< 1s response time for 2M rows group-by with 4 attributes
Filtering is cheaper and group-by relatively expensive
More rows filtered => the faster response

Please Note: for simple group-by aggregate queries, phoenix probably might respond in <1s for about 4-5M rows
### Scans, Joins and Secondary Indexes

#### widget-pages-stats

<table>
<thead>
<tr>
<th>user details</th>
<th>widget details</th>
<th>aggregate-stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>dealer id</td>
<td>page label</td>
<td>Time stamp</td>
</tr>
<tr>
<td></td>
<td>page label</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>user segments</td>
<td>user segments</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
<th>version</th>
<th>context</th>
<th>stats-1</th>
<th>stats-2</th>
<th>stats-3</th>
<th>stats-4</th>
<th>stats-5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>clicks</td>
<td>views</td>
<td>hover</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### widget-details

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
<th>version</th>
<th>context</th>
<th>start date</th>
<th>end date</th>
<th>target campaign</th>
<th>widget name</th>
<th>widget details</th>
</tr>
</thead>
</table>

#### widget-pages-stats-secondary-widget-index

<table>
<thead>
<tr>
<th>widget details</th>
<th>user details</th>
<th>aggregate-stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>type</td>
<td>version</td>
</tr>
<tr>
<td>context</td>
<td>dealer id</td>
<td>page label</td>
</tr>
<tr>
<td>Time stamp</td>
<td>page label</td>
<td>Type</td>
</tr>
<tr>
<td>device</td>
<td>Type</td>
<td>user segments</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
<th>version</th>
<th>context</th>
<th>dealer id</th>
<th>page label</th>
<th>Time stamp</th>
<th>device</th>
<th>Type</th>
<th>user segments</th>
<th>stats-1</th>
<th>stats-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Simple Hash Join

Or

Sort Merge Join

#### Phoenix – "Covered Global Secondary Index"

A Secondary Table is maintained in Parallel with Original Table

#### Secondary index

#### Covered Columns
Scans, Joins and Secondary Indexes

**Range Scan & Join**

<table>
<thead>
<tr>
<th>widget-pages-stats</th>
<th>widget-details</th>
</tr>
</thead>
<tbody>
<tr>
<td>dealer id</td>
<td>id</td>
</tr>
<tr>
<td>manufacturer-dealer-1</td>
<td>type</td>
</tr>
<tr>
<td>manufacturer-dealer-2</td>
<td></td>
</tr>
<tr>
<td>..</td>
<td></td>
</tr>
<tr>
<td>manufacturer-dealer-n</td>
<td></td>
</tr>
</tbody>
</table>

**Skip Scan & Join**

<table>
<thead>
<tr>
<th>widget-pages-stats</th>
<th>widget-details</th>
</tr>
</thead>
<tbody>
<tr>
<td>dealer id</td>
<td>id</td>
</tr>
<tr>
<td>page-label</td>
<td>type</td>
</tr>
<tr>
<td>abcd</td>
<td></td>
</tr>
<tr>
<td>abcd</td>
<td></td>
</tr>
<tr>
<td>abcd</td>
<td></td>
</tr>
<tr>
<td>HomePage</td>
<td></td>
</tr>
<tr>
<td>VehicleSearchPage</td>
<td></td>
</tr>
<tr>
<td>VehicleDetails</td>
<td></td>
</tr>
</tbody>
</table>

**Skip Scan in Phoenix**

Utilizes HBase `SEEK_NEXT_USING_HINT`.
Skips to the next correct intra region key column.
Will also use guide posts (if available).
Performance Optimizations
Performance Improvement Variables

Block Size, Block Encoding and Block Compression

Sample HFile Data Block: cf1

```
r1:cf1:c1:t60:value
r1:cf1:c1:t59:value
........
........
r1:cf1:c1:t01:value
r1:cf1:c2:t60:value
r1:cf1:c2:t59:value
........
........
r1:cf1:c2:t01:value
........
r1:cf1:c15:t59:value
........
........
r1:cf1:c15:t01:value
```

Special Encoding to eliminate dupes (PREFIX-TREE, FAST-DIFF)
Compression post Encoding (LZ4, GZ, SNAPPY)
## Performance Optimization Variables

### Phoenix
- Number of Salt Buckets
- Width of Guide Posts
- Selecting Primary keys
  - proper filter keys, proper group-by keys
  - light weight primary key
- Query Plan
  - Usage of skip scans
  - Parallel scan using guide posts
  - Utilizations of Secondary indexes
- Phoenix Query
  - Memory Utilization settings

### HBase
- Compression
  - Data Block compression
  - GZ, LZ4, SNAPPY
- Encoding
  - Data Block Encoding
  - FAST DIFF and PREFIX TREE
- HBase Block Size
  - Data block is minimum data read into region server
  - Data block is cached in Block Cache
Performance Optimizations – Encoding, Compression, Buckets/Regions

* These numbers could change based on data size from each node
Appendix – A

Evolution of Clickstream ETL Pipeline

Apache-1 → Kafka → Spark Based Reduce → HDFS → DIM SOURCES

DIM SOURCES → DIM-1 → DIM-2 → DIM-3
DIM SOURCES → DIM-4 → FACT_WIDGET_EVENTS → DIM-5
DIM SOURCES → DIM-6 → DIM-7 → DIM-8

DIM SOURCES → Map Reduce → HBase/Phoenix

Past

Apache-1 → File Mounts → Spring Batch → DB Dedup → DIM SOURCES

DIM SOURCES → DIM-1 → DIM-2 → DIM-3
DIM SOURCES → DIM-4 → FACT_WIDGET_EVENTS → DIM-5
DIM SOURCES → DIM-6 → DIM-7 → DIM-8

DIM SOURCES → Standalone Java App → Document Caching Store

Worst Case Scenario Per Day
25 M rows total
Total Time to Phoenix – 1 hrs
Insert into Phoenix – 5 mins

Present

Worst Case Scenario Per Day
25 M rows total
Total Time to Cache – 6 hrs
Insert into Cache – 90 mins
**Appendix-B**  

**Cluster Hardware**

- **Name Node, Resource Manager, HBase Master Primary & Secondary**
- **Ambari, Hue, Knox Ranger, Nagios, Ganglia, Oozie, History Server, Falcon, Hive2**
- **Edge Nodes**

---

**Data Node**

- **Node 1**
  - Region Server 1
  - 256 GB RAM
  - 24 GB RAM for RS
  - 25 * 1 TB Disks
  - 10000 RPM
  - 24 Cores
  - 2.7 GHz

- **Node 2**
  - Region Server 2
  - 256 GB RAM
  - 24 GB RAM for RS
  - 25 * 1 TB Disks
  - 10000 RPM
  - 24 Cores
  - 2.7 GHz

- **Node 3**
  - Region Server 3
  - 256 GB RAM
  - 24 GB RAM for RS
  - 25 * 1 TB Disks
  - 10000 RPM
  - 24 Cores
  - 2.7 GHz

- **Node 4**
  - Region Server 4
  - 256 GB RAM
  - 24 GB RAM for RS
  - 25 * 1 TB Disks
  - 10000 RPM
  - 24 Cores
  - 2.7 GHz

- **Node 5**
  - Region Server 5
  - 256 GB RAM
  - 24 GB RAM for RS
  - 25 * 1 TB Disks
  - 10000 RPM
  - 24 Cores
  - 2.7 GHz

---

**10 GB - Ethernet Switch**
Questions?