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

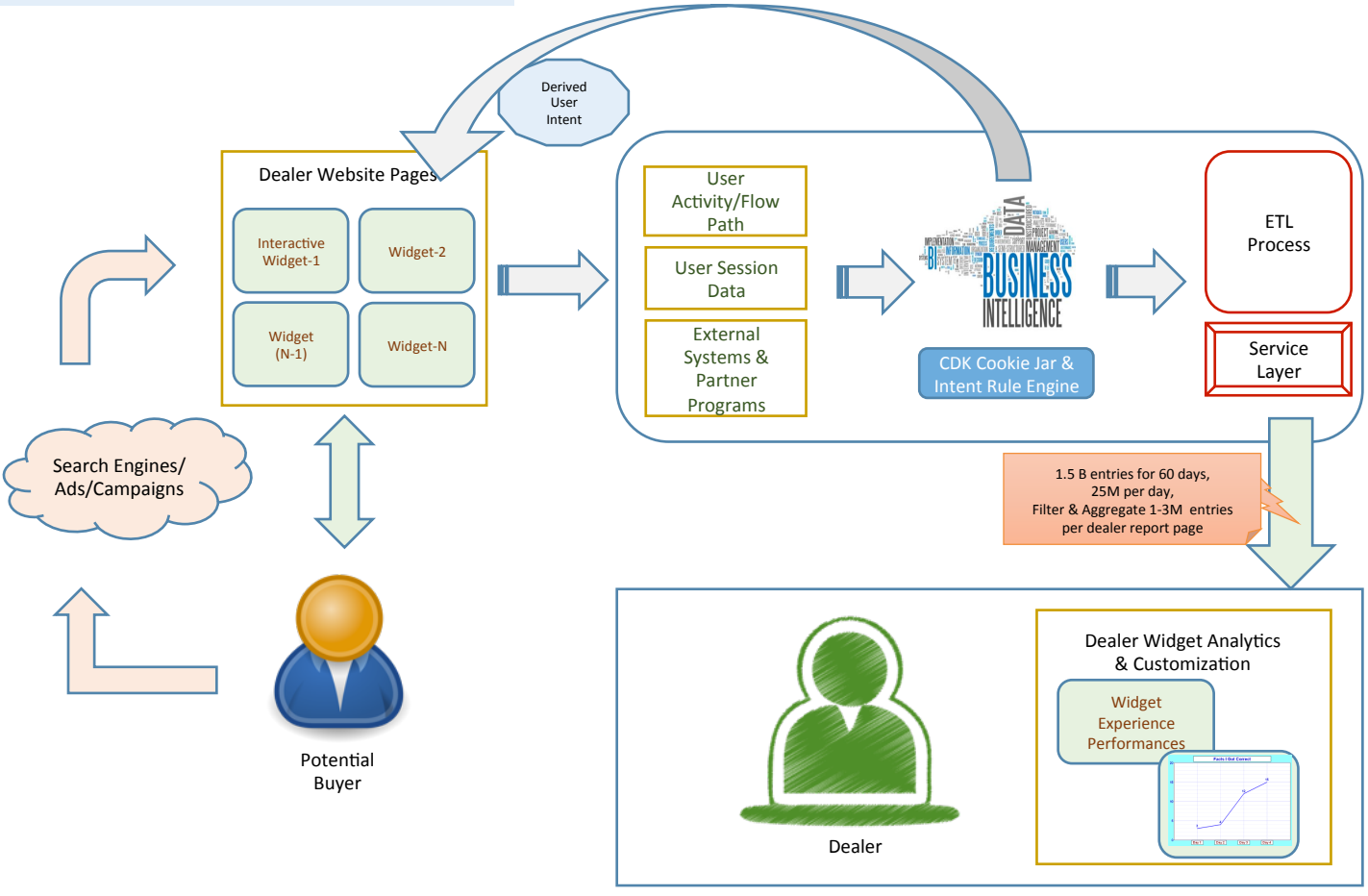
- For the purpose of this presentation, we are interested
  - ***Dealer Web Sites and Clickstreams from web sites***

- 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 Use Case



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

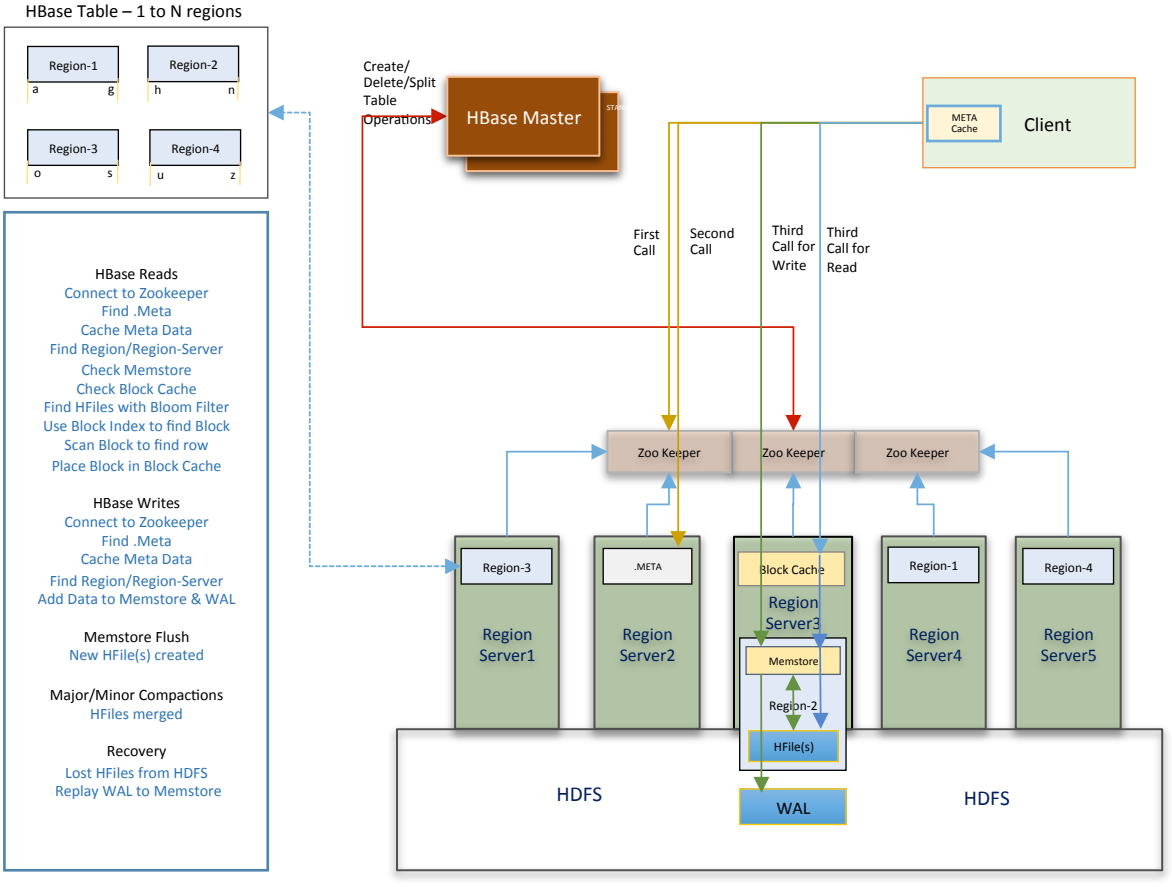
- Column family oriented store
- Highly scalable with no central index
- Open source re-incarnation of BigTable
- Great for Aggregating large amounts of data
- Fully Consistent

HBase – Table
NOSQL Key Value Store

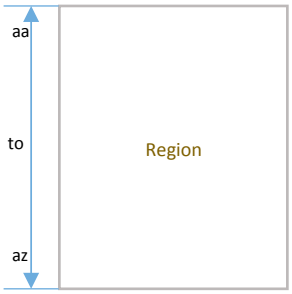
PK	Column Family -1		
rowkey-1	colname-1	value-2	timestamp-2
	colname-1	value-1	timestamp-1
	colname-2	value-2	timestamp-1
	colname-3	value-3	timestamp-4
	...	...	....
rowkey-2	colname-1	value-5	timestamp-4
	colname-5	value-1	timestamp-1
	colname-6	value-2	timestamp-0
	...	...	....

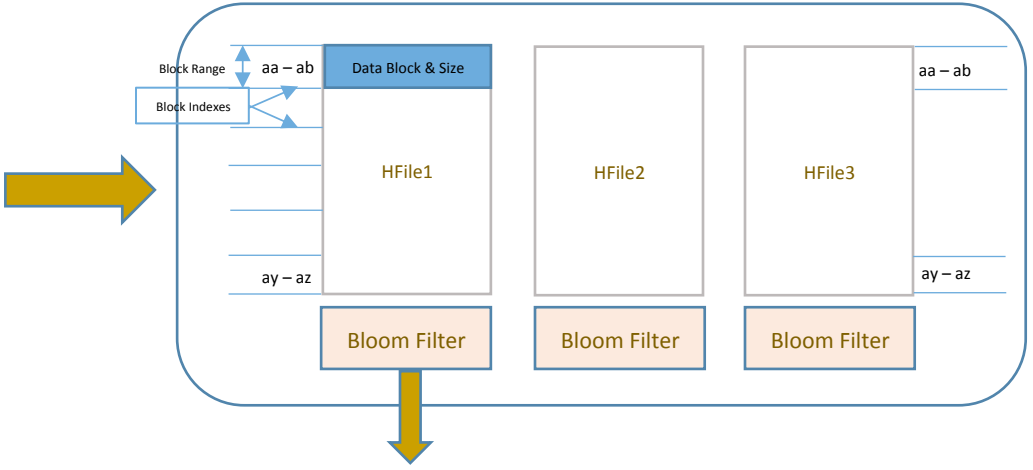
Handle Billions of these with Ease?. How?

HBase Architecture

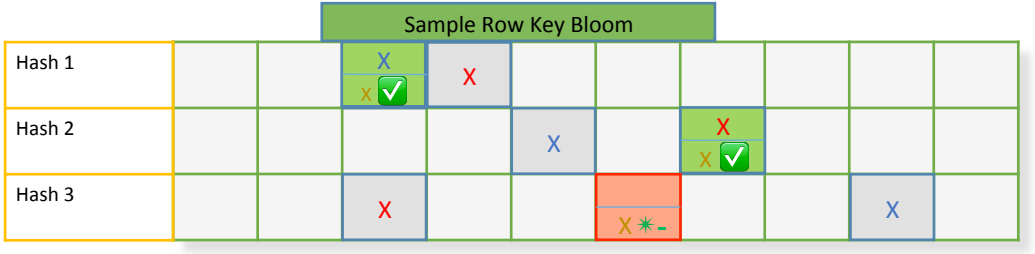


HBase - Finding data within Region Files
Region Blocks and Bloom Filter





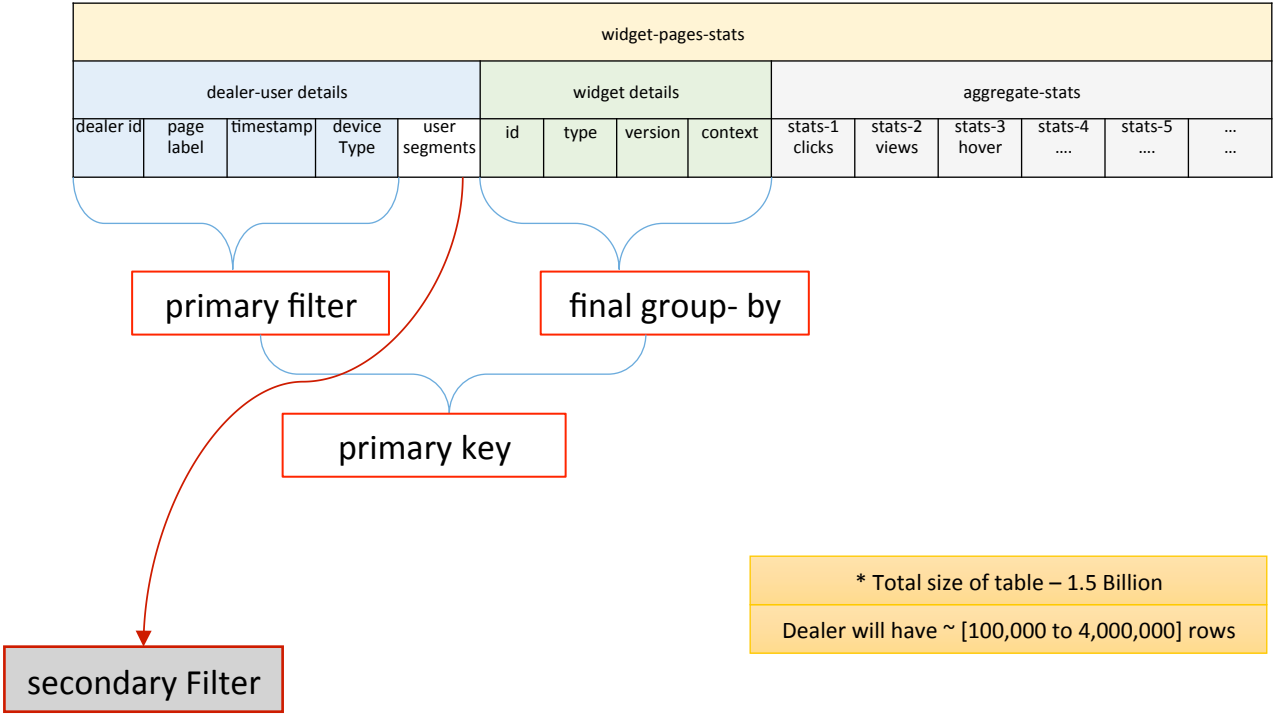
AAXY
Hash 3 marker is empty;
So, it doesn't exist in HFile1



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

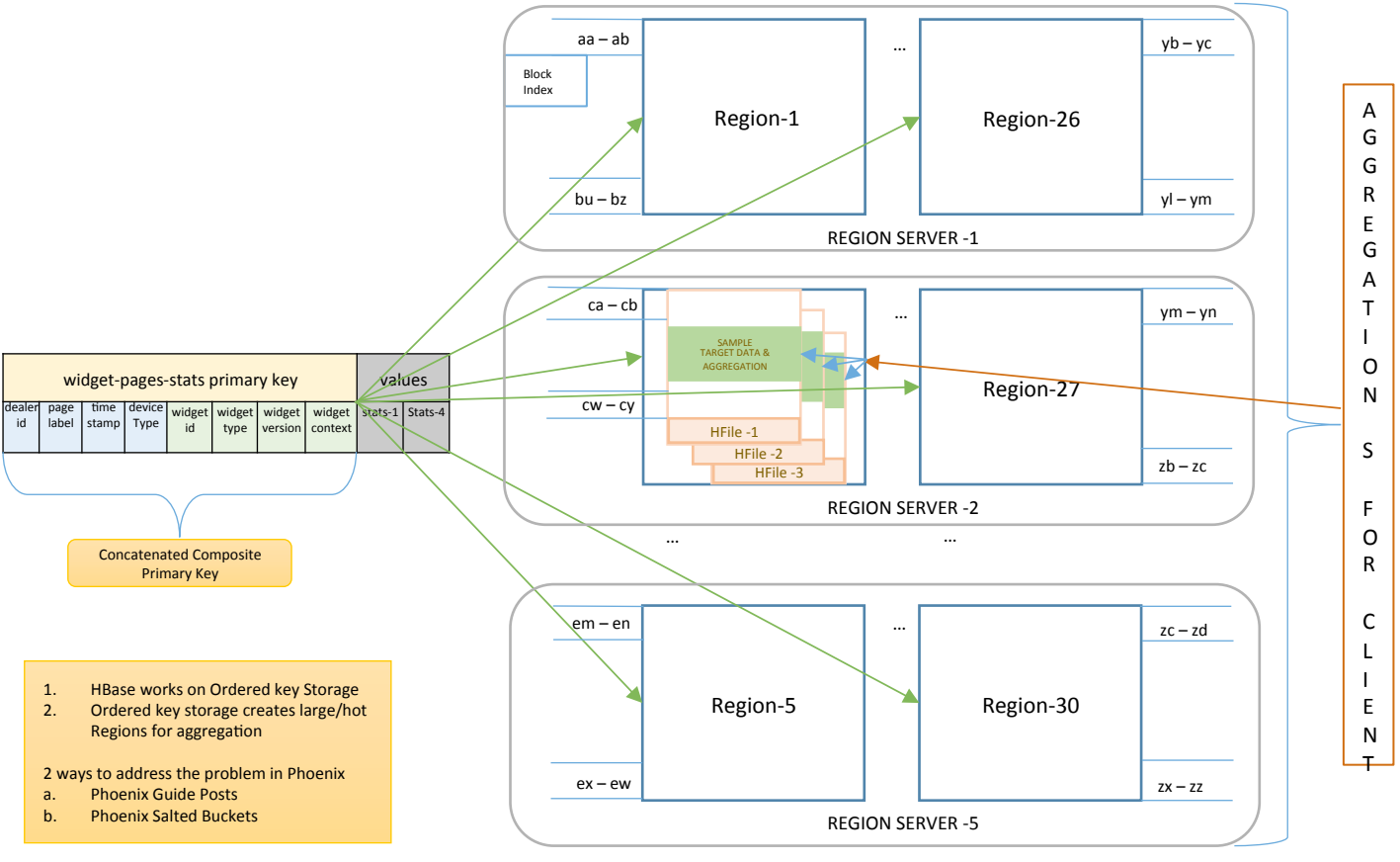
Widget Semi Aggregated Table – Core Table



* Total size of table – 1.5 Billion

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

HBase – Aggregation

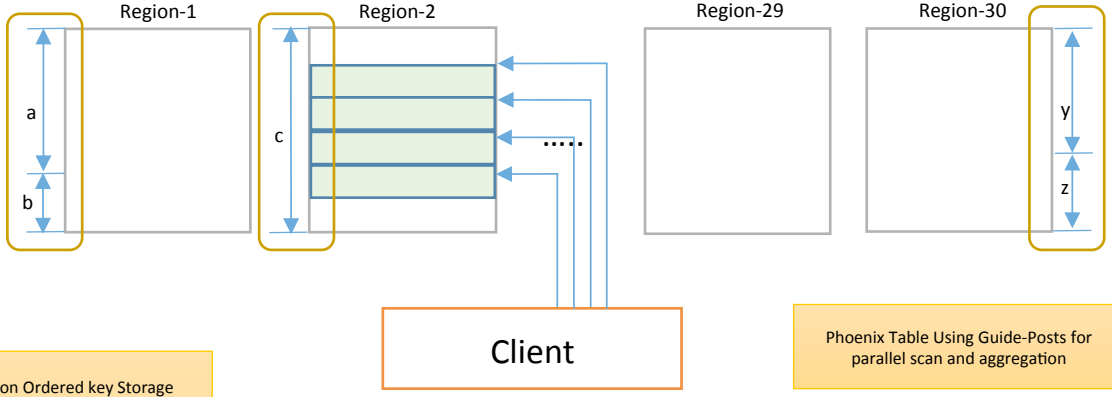


1. HBase works on Ordered key Storage
2. Ordered key storage creates large/hot Regions for aggregation

2 ways to address the problem in Phoenix

a. Phoenix Guide Posts
b. Phoenix Salted Buckets

Phoenix Aggregation

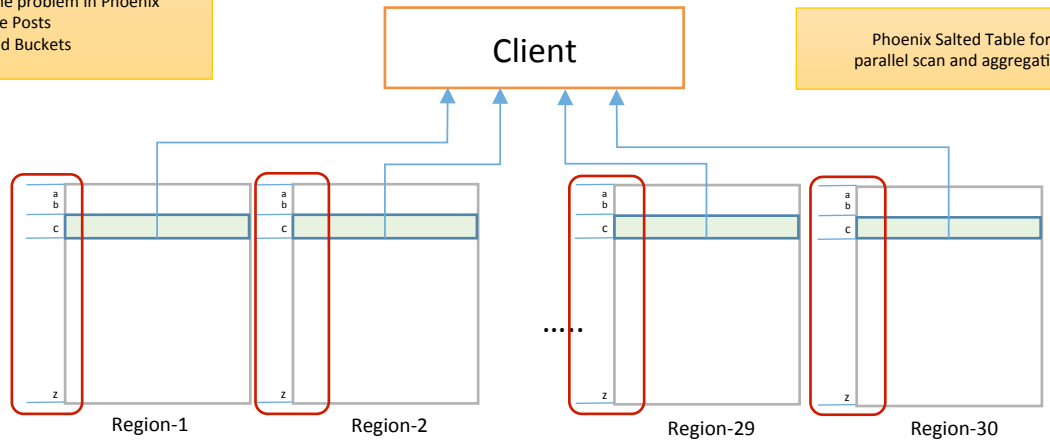


Phoenix Table Using Guide-Posts for parallel scan and aggregation

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

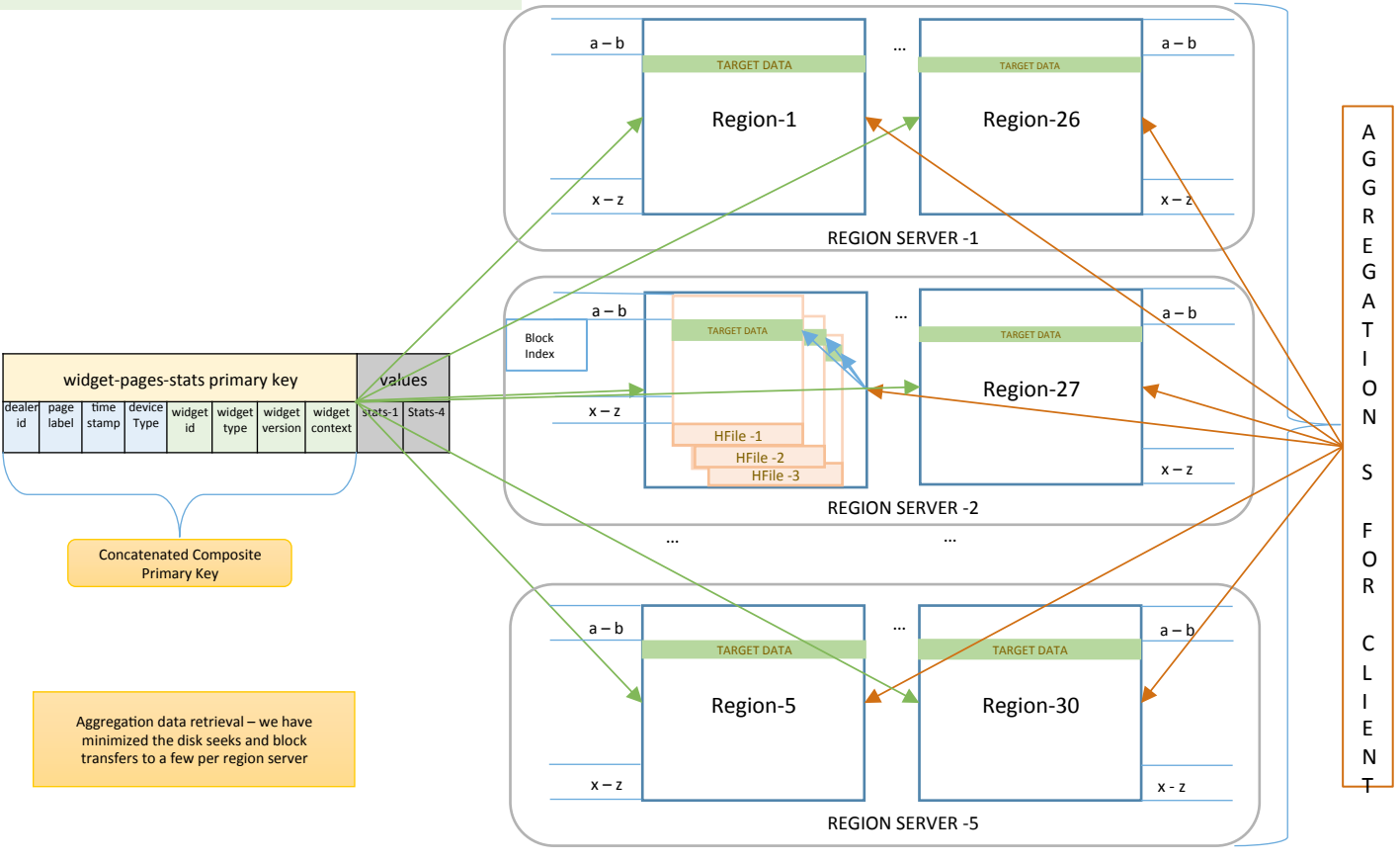
2 ways to address the problem in Phoenix

a. Phoenix Guide Posts
b. Phoenix Salted Buckets



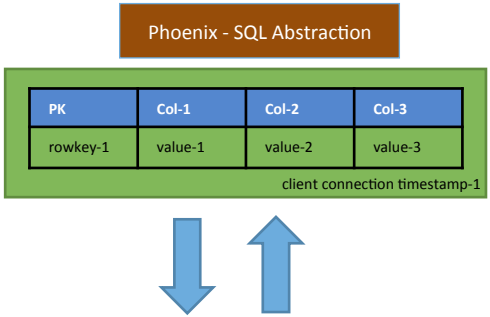
Phoenix Salted Table for parallel scan and aggregation

Phoenix – Salted Buckets Aggregation
Initially Regions = Salted Buckets

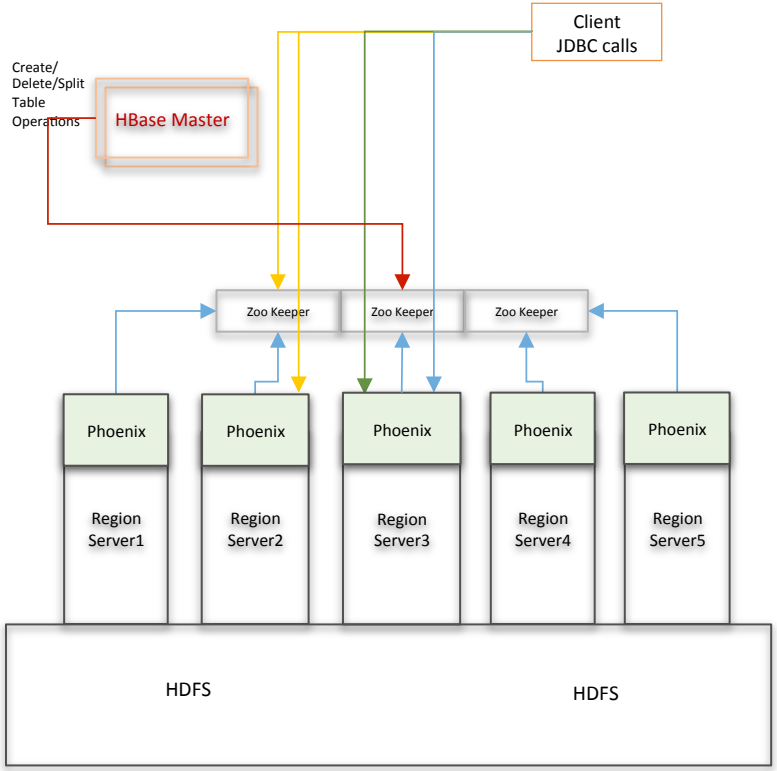


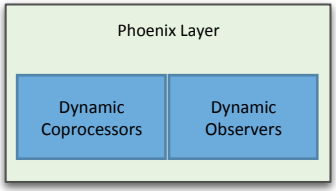
Phoenix – Overview

- 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



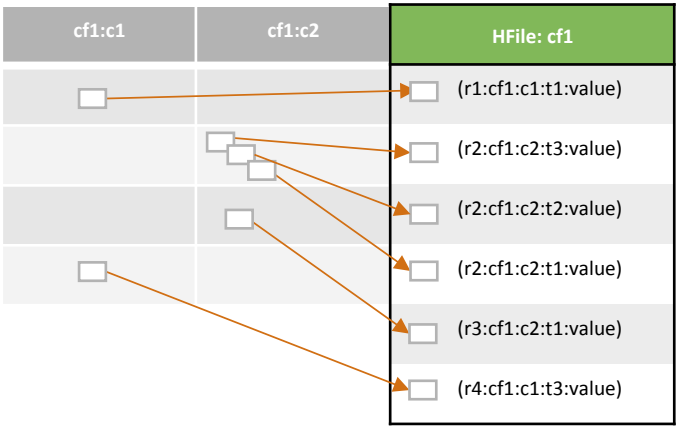


Coprocessors – execute aggregation
Observers – Various Types – observe to sync data

Timestamp Handling in HBase

Timestamp and Expiration Handling

row-key	cf1:c1	cf1:c2
r1	t1	
r2		t3, t2, t1
r3		t1
r4	t3	



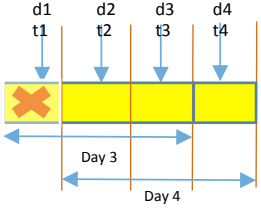
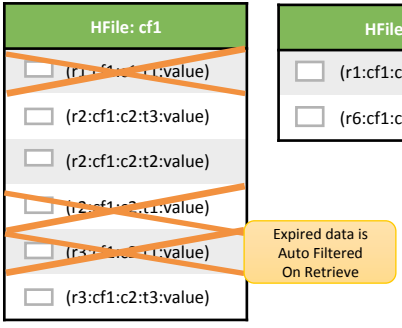
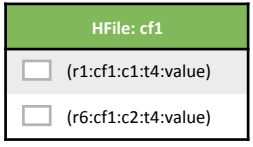
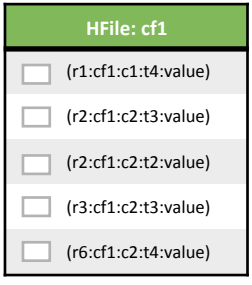


Table A
Time To Live = 3 days





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

localhost:8080/#/notebook/2BB6J25YG

Zeppelin Notebook Interpreter

Search in your notebooks

Connected

Widget Experience Playground

1d

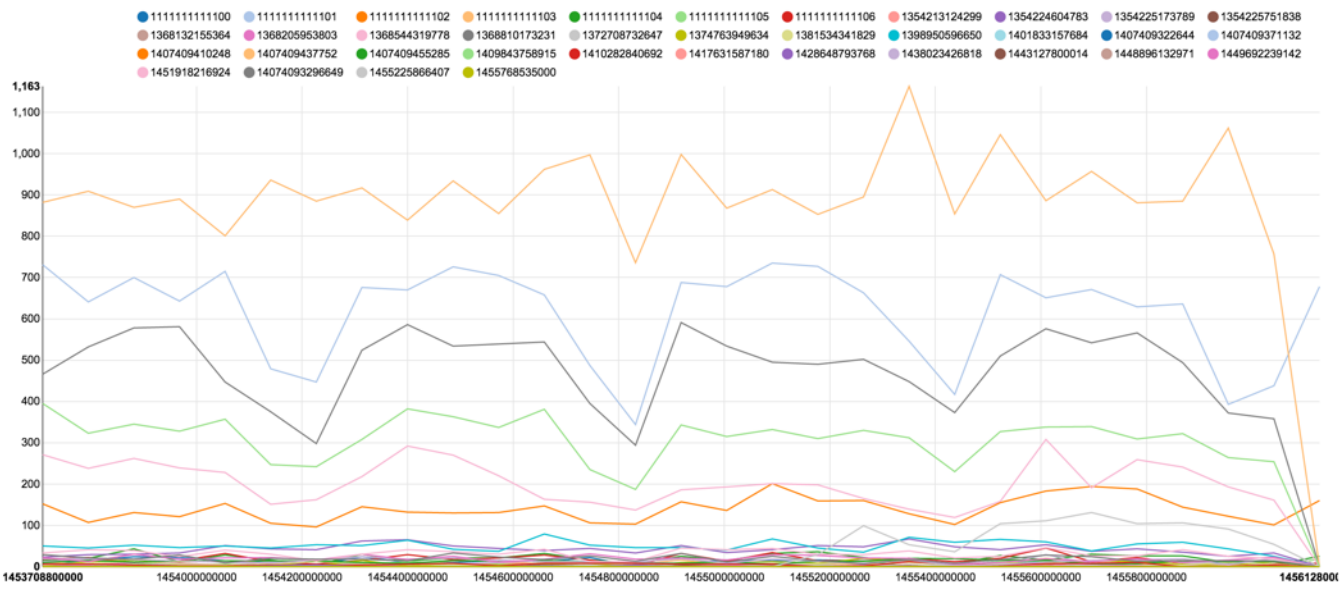
default

Visualize Widget Segment And Experience

FINISHED

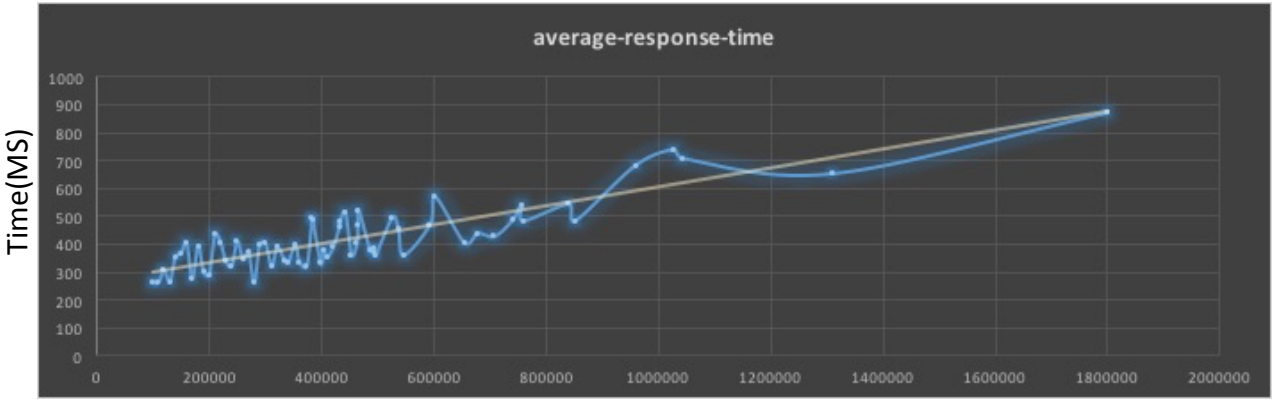
```
%phoenix
SELECT VIEW_DATE_TIMESTAMP, WIDGET_INSTANCE_ID, SUM(TOTAL_CLICK_VIEWS), SUM(TOTAL_TIME_ON_PAGE_MS),SUM(TOTAL_VIEWABLE_TIME_MS),SUM(TOTAL_HOVER_TIME_MS),SUM(VIEW_COUNT)
FROM WIDGET_PAGES_STATS
WHERE WEB_ID = '[redacted]'
AND WEB_PAGE_LABEL in ('homepage')
--AND user_segment LIKE '%silverado%'
--and widget_context like '%silverado%'
--and widget_context like '%model%'
--AND VIEW_DATE_TIMESTAMP > 1453900000000
--and widget_instance_id = 1409843758915
GROUP BY VIEW_DATE_TIMESTAMP, WIDGET_INSTANCE_ID
```

settings



Took 1 seconds

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 Severs – 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													
user details					widget details				aggregate-stats				
dealer id	page label	Time stamp	device Type	user segments	id	type	version	context	stats-1 clicks	stats-2 views	stats-3 hover	stats-4	stats-5 ....

widget-details									
id	type	version	context	start date	end date	target campaign	widget name	widget details	...

Simple Hash Join
Or
Sort Merge Join

Phoenix – "Covered Global Secondary Index"

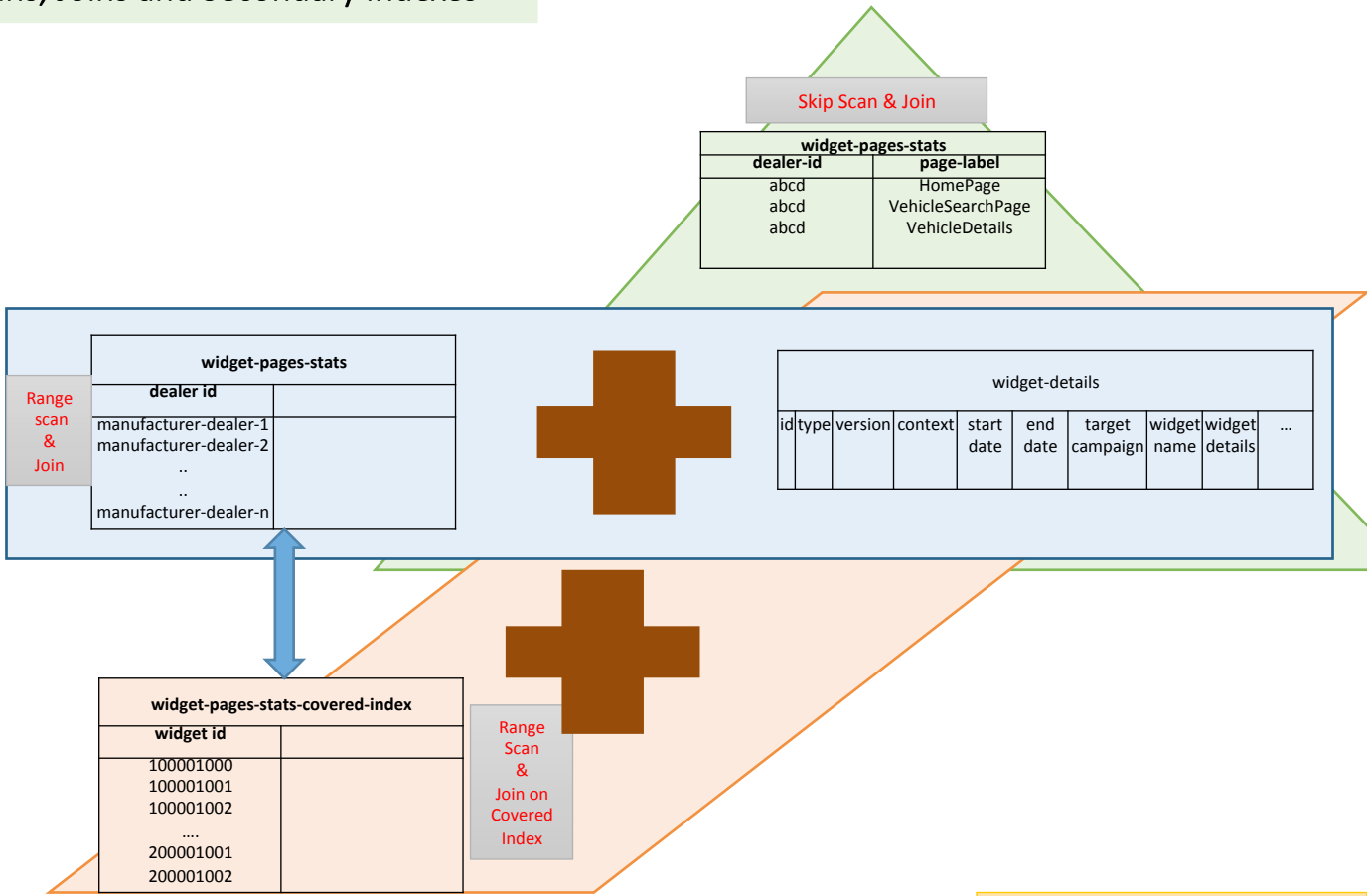
A Secondary Table is maintained in Parallel with Original Table

widget-pages-stats-secondary-widget-index										
widget details				user details					aggregate-stats	
id	type	version	context	dealer id	page label	Time stamp	device Type	user segments	stats-1 clicks	stats-2 views

Secondary index

Covered Columns

Scans, Joins and Secondary Indexes



Skip Scan in Phoenix

Utilizes HBase SEEK_NEXT_USING_HINT

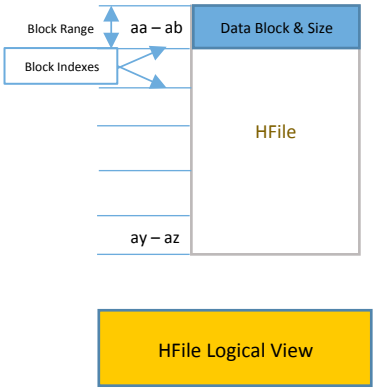
Skips to the next correct intra region key column

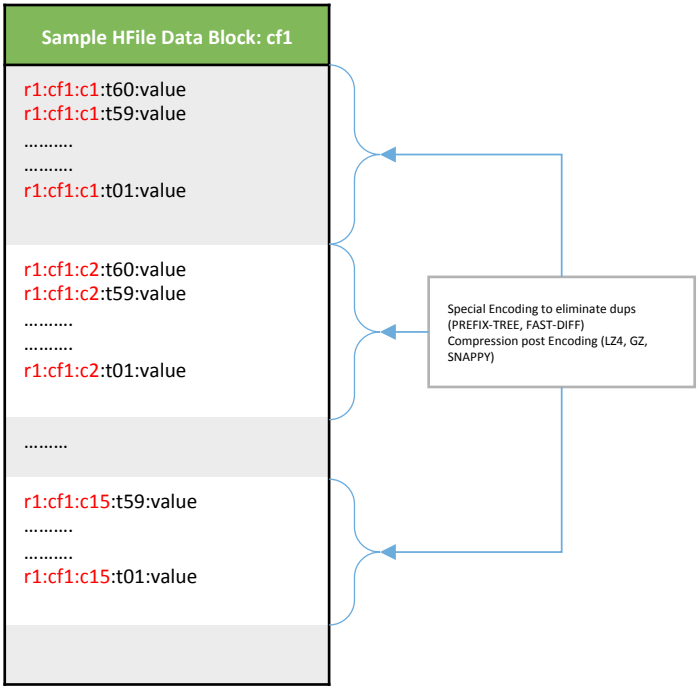
Will also use guide posts (if available)

Performance Optimizations

Performance Improvement Variables

Block Size, Block Encoding and Block Compression





Performance Optimization Variables

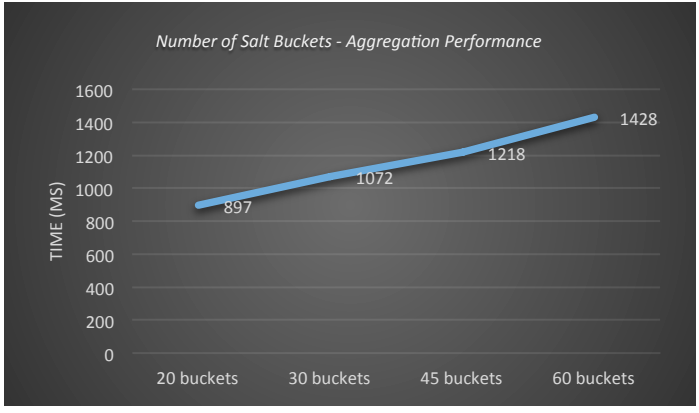
Phoenix

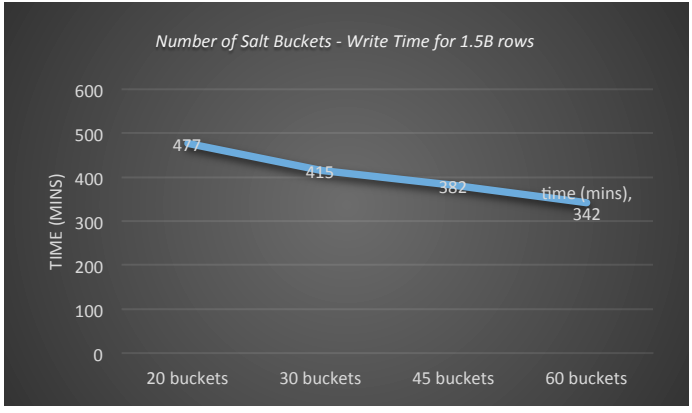
- Number of Salt Buckets
- Width of Guide Posts
- Selecting Primary keys
  - proper filter keys, proper group-by keys
  - light weight primary key
- Query Plan
  - Usage of skip scans
  - Parallel scan using guide posts
  - Utilizations of Secondary indexes
- Phoenix Query
  - Memory Utilization settings

HBase

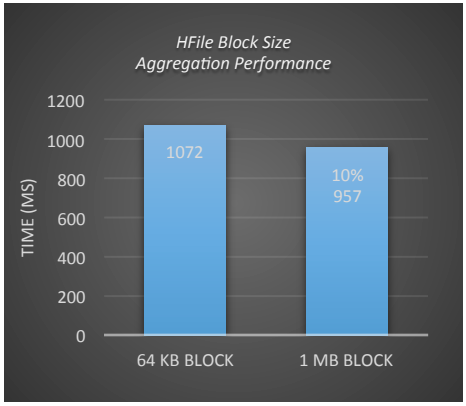
- Compression
  - Data Block compression
  - GZ, LZ4, SNAPPY
- Encoding
  - Data Block Encoding
  - FAST DIFF and PREFIX TREE
- HBase Block Size
  - Data block is minimum data read into region server
  - Data block is cached in Block Cache

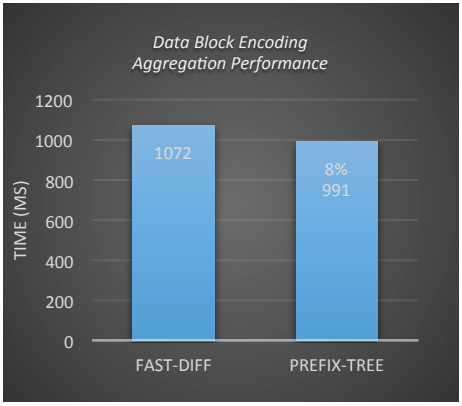
Performance Optimizations – Encoding, Compression, Buckets/Regions

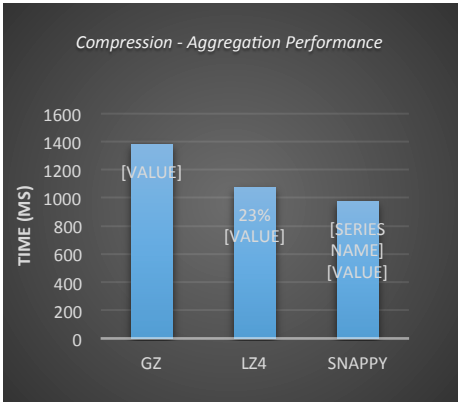




* These numbers could change based on data size from each node



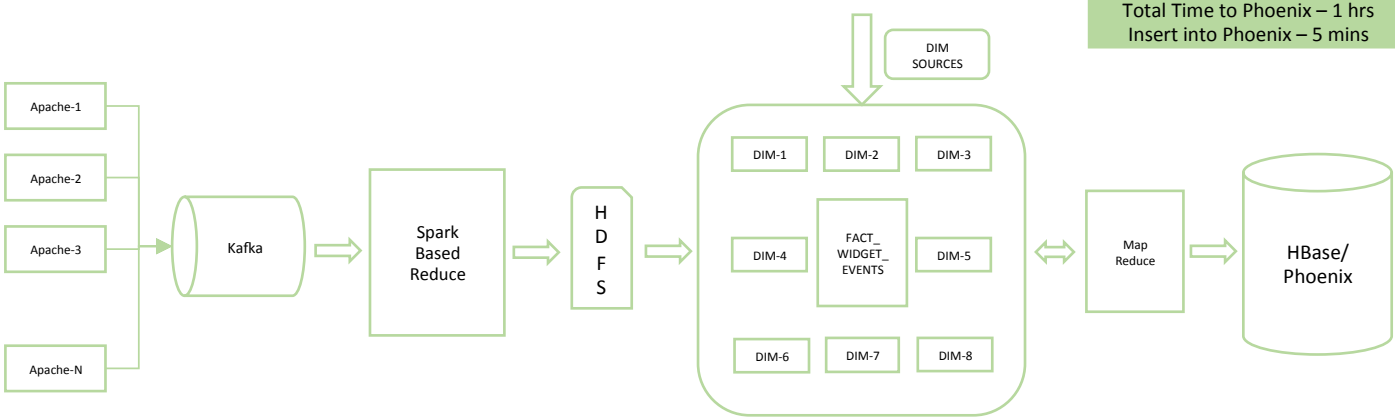




Appendix – A

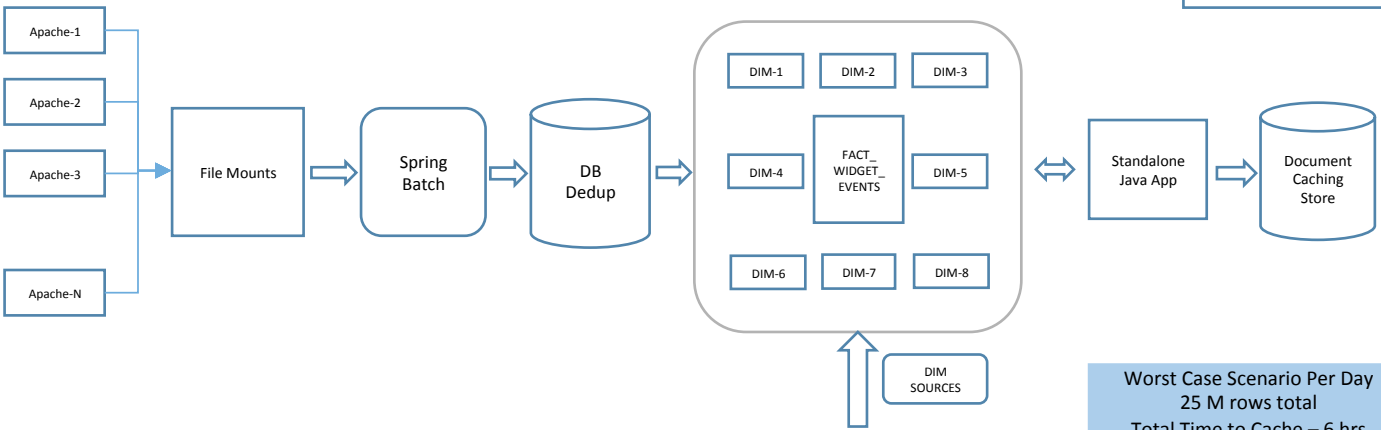
Evolution of Clickstream ETL Pipeline

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



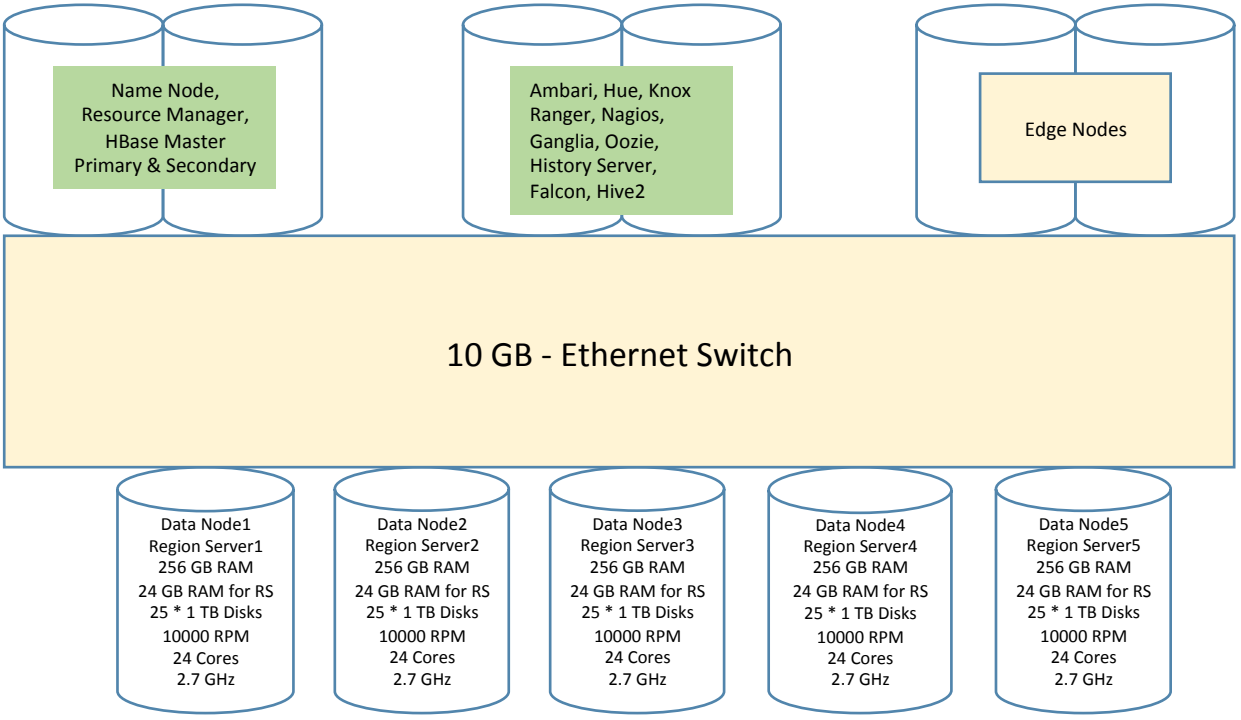
Present

Past



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

Appendix-B Cluster Hardware



Questions?