PayPal Merchant ecosystem using Spark, Hive, Druid, HBase & Elasticsearch

Deepika Khera & Kasiviswanathan Natarajan

June 19 2018

Who we are?

Deepika Khera

- Big Data Technologist for over a decade
- Focused on building scalable platforms with Hadoop ecosystem – Map Reduce, HBase, Spark, Elasticsearch, Druid
- Senior Engineering Manager - Merchant Analytics at PayPal
- Contributed to Druid for the Spark Streaming integration

Kasi Natarajan

- 15+ years of industry experience
- Spark Engineer @PayPal Merchant Analytics
- Building solutions using Apache Spark, Scala, Hive, HBase, Druid and Spark ML.
- Passionate about providing Analytics at scale from Big Data platforms

Agenda

- PayPal Data & Scale
- Merchant Use Case Review
- Data Pipeline
- Learnings - Spark & HBase
- Tools & Utilities
  - Behavioral Driven Development
  - Data Quality Tool using Spark
  - BI with Druid & Tableau

PayPal Data & Scale

PayPal is more than a button

CBT

Mobile

In-Store

Online

Loyalty

Credit

APV Lift

Offers

Faster Conversion

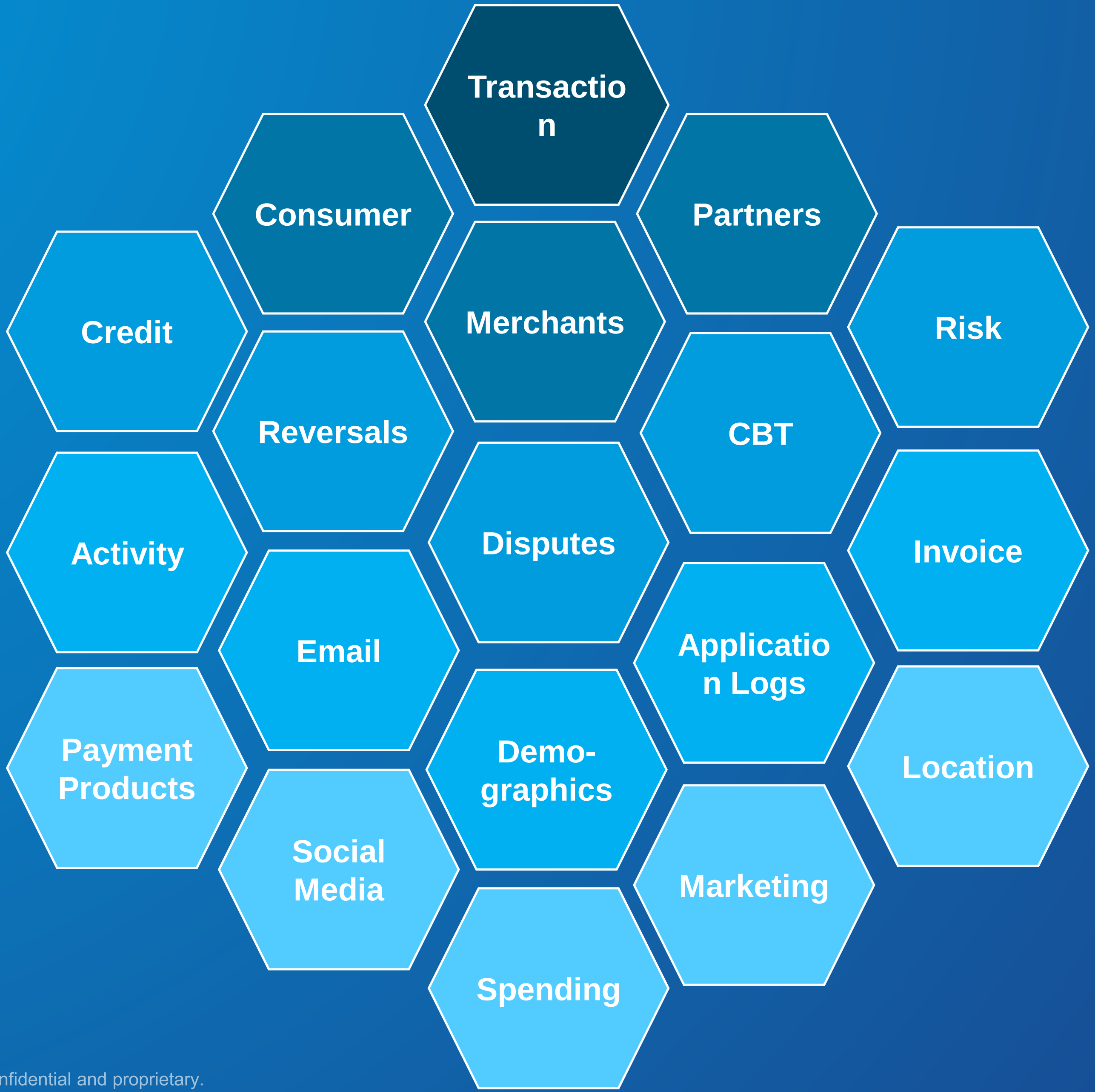
Reduction in Cart Abandonment

Customer Acquisition

Invoicing

Check out with PayPal

PayPal Datasets



The power of
our platform

PayPal operates one of the largest
PRIVATE CLOUDS
in the world*

7.6
BILLION
payments in 2017**

~600
payments/ second at peak*

19M
merchants

200+
markets

237M
active customer accounts**

42
petabytes of data*

Dedicated to with a customer focused, strong performance, highly scalable, continuously available
PLATFORM.

PayPal operates one of the largest Hadoop deployments in the world.
A 1600 Node Hadoop Cluster
with 230TB of Memory, 78PB of Storage
Running 50,000 Jobs Per day

PayPal has one of the top five Kafka deployments in the world, handling over 200 billion messages per day

Merchant Use Case Review

Use Case Overview

INSIGHTS PayPal.com

- Revenue & transaction trends
- Cross-Border Insights
- Customer Shopping Segments

MARKETING SOLUTIONS

- Help Merchants engage their customers by personalized shopping experience
- Offers & Campaigns
- Shoppers Insights

PAYPAL ANALYTICS

- Products performance
- Checkout Funnel
- Behavior analysis
- Measuring effectiveness

Merchant Data Platform

1. Fast Processing platform crunching multi-terabytes of data
2. Scalable, Highly available, Low latency Serving platform

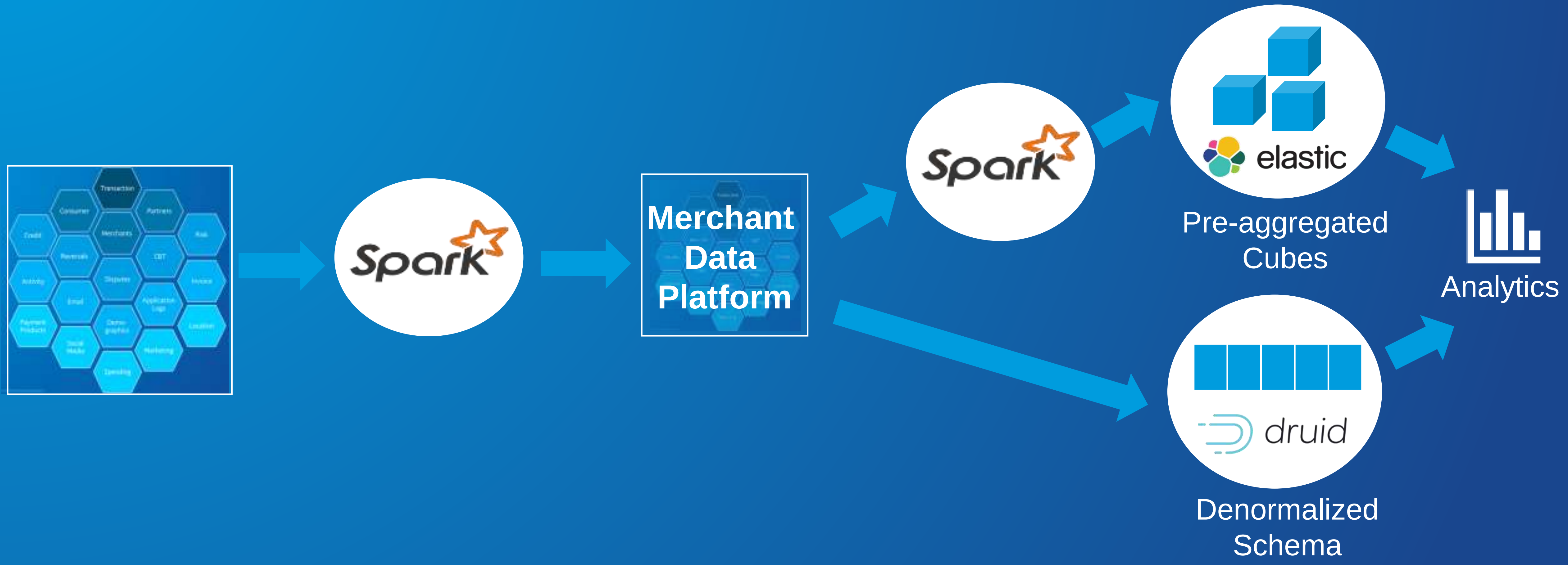
Technologies

Spark — Processing

elastic druid — Serving

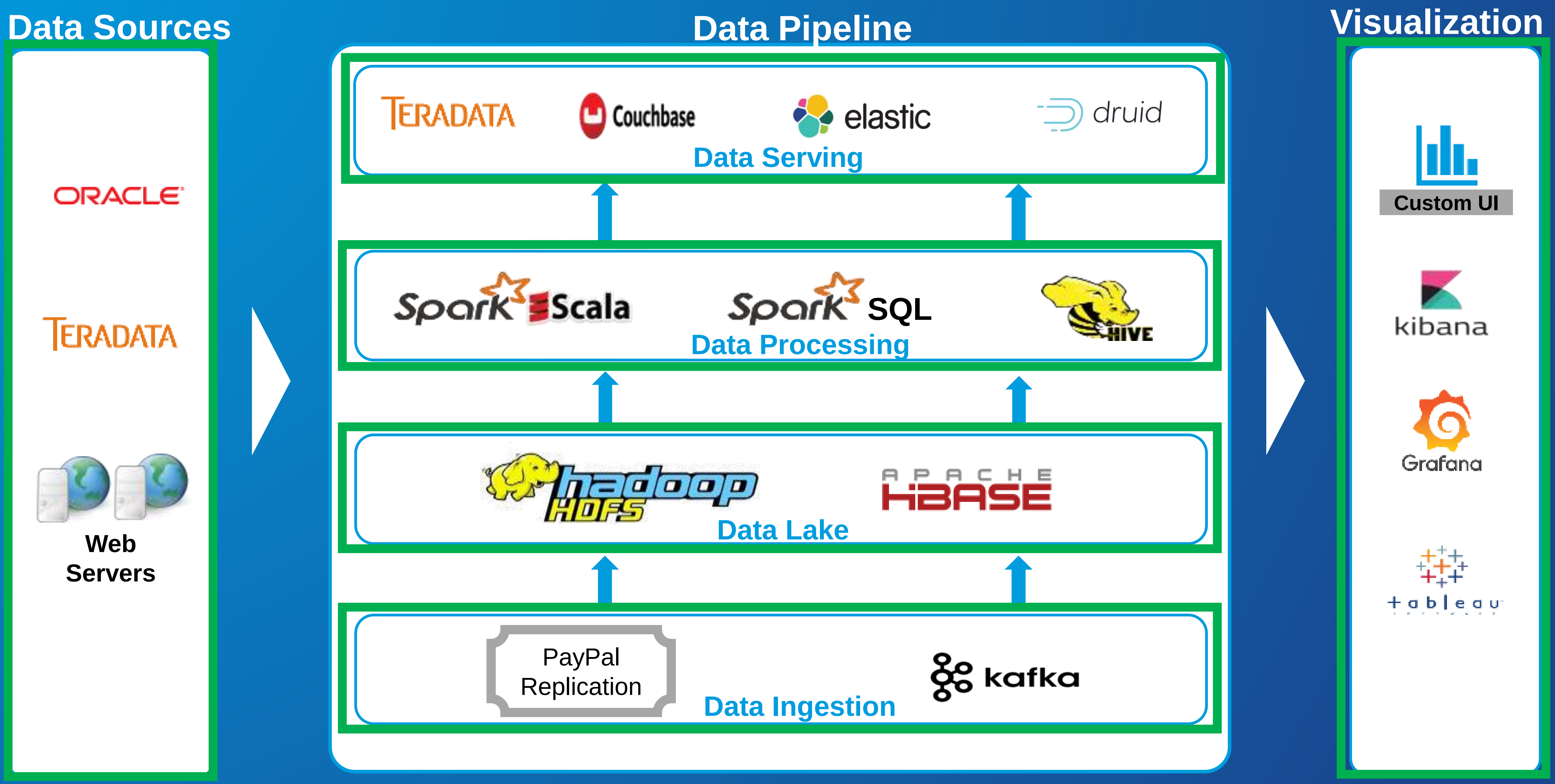
kafka — Movement

Merchant Analytics



Data Pipeline

Data Pipeline Architecture



Learnings – Spark & HBase

Spark Best Practices Checklist

Data Serialization	✓ Use Kyro Serializer with SparkConf, which is faster and compact ✓ Tune kyroserializer buffer to hold large objects
Garbage Collection	✓ Clean up cached/persisted collections when they are no longer needed ✓ Tuned concurrent abortable preclean time from 10sec to 30sec to push out stop the world GC
Memory Management	✓ Avoided using executors with too much memory
Parallelism	✓ Optimize number of cores & partitions*
Action-Transformation	✓ Minimize shuffles on join() by broadcasting the smaller collection ✓ Optimize wider transformations as much as possible*
Caching & Persisting	✓ Used MEMORY_AND_DISK storage level for caching large ✓ Repartition data before persisting to HDFS for better performance in downstream jobs

*Specific examples later

Learnings

Spark job failures with Fetch Exceptions

Observations

- Executor spends long time on shuffle reads. Then times out , terminates and results in job failure
- Resource constraints on executor nodes causing delay in executor node

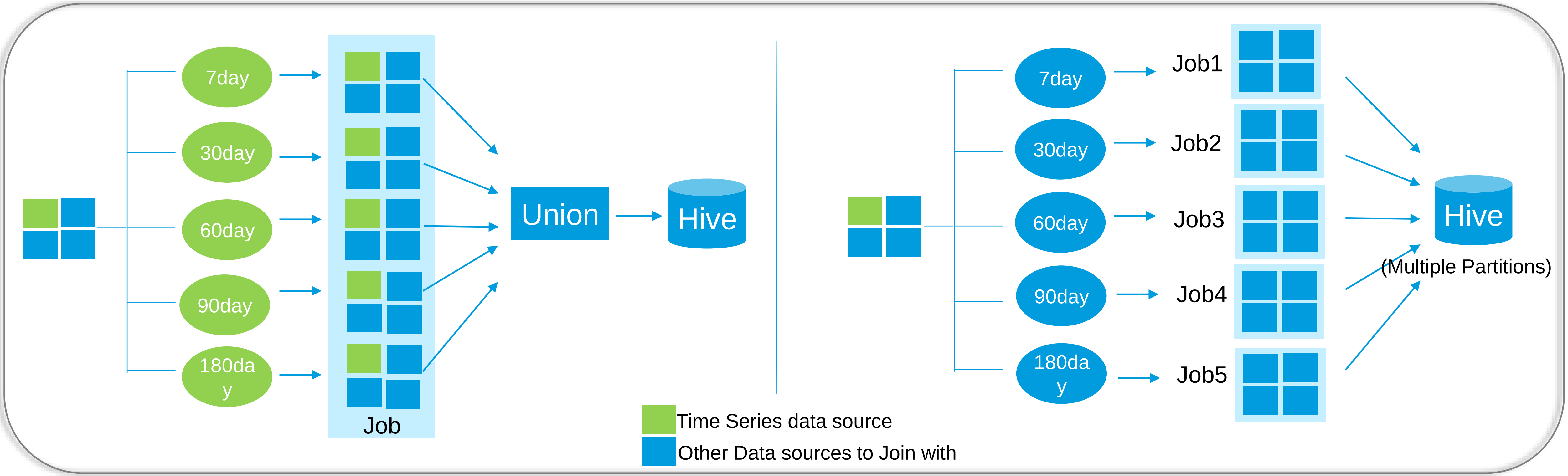
Resolution

To address memory constraints, tuned

1. config from 200 executor * 4 cores to 400 executor * 2 cores
2. executor memory allocation (reduced)

Learnings

Parallelism for long running jobs



Observations

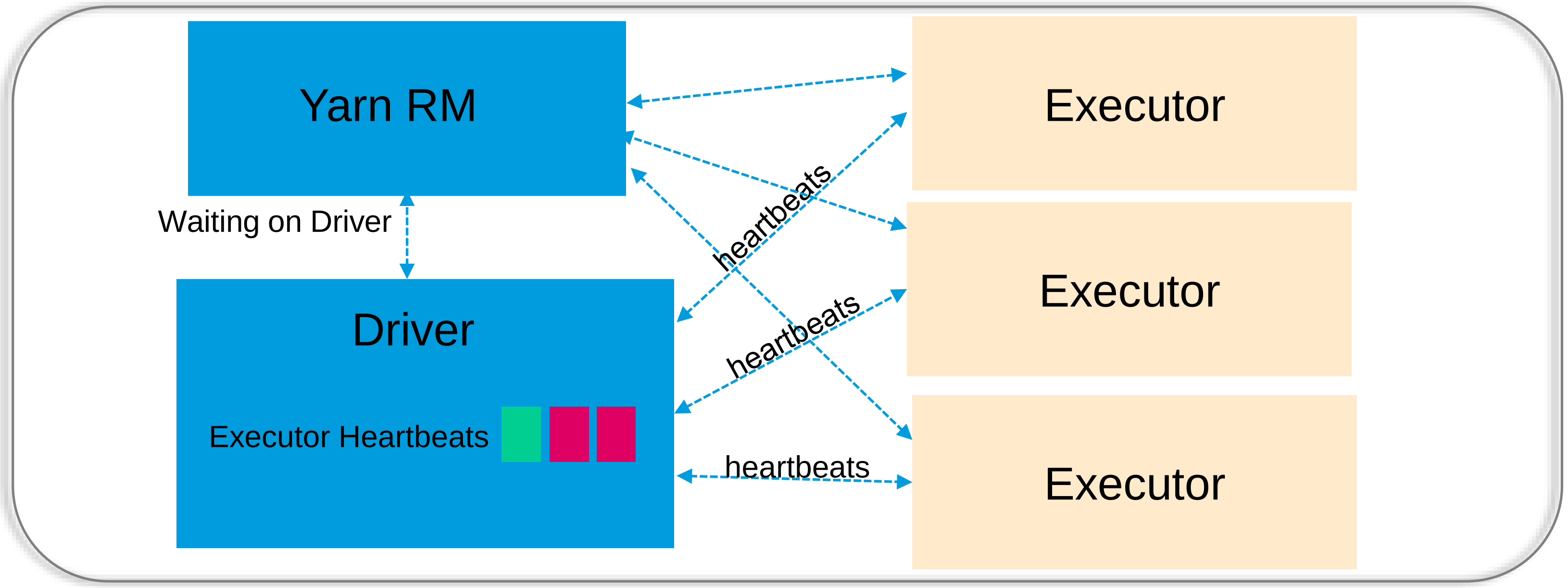
- Series of left joins on large datasets cause shuffle exceptions

Resolution

1. Split into Small jobs and run them in parallel
2. Faster reprocessing and fail fast jobs

Learnings

Tuning between Spark driver and executors



Observations

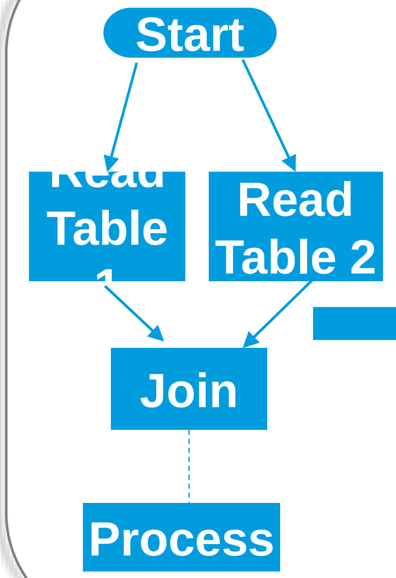
- Spark Driver was left with too many heartbeat requests to process even after the job was complete
- Yarn kills the Spark job after waiting on the Driver to complete processing the Heartbeats

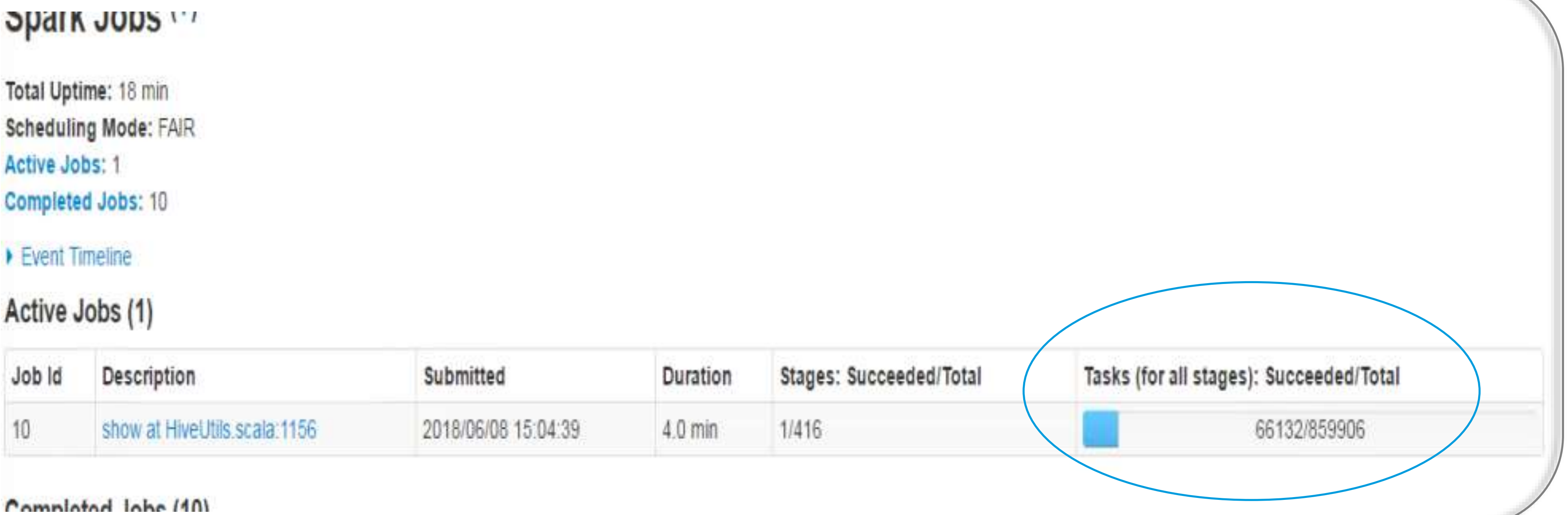
Resolution

- The setting “spark.executor.heartbeatInterval” was set too low. Increasing it to 50s fixed the issue
- Allocate more memory to Driver to handle overheads other than typical Driver processes

Learnings

Optimize joins for efficient use of cluster resources (Memory, CPU etc..,)





Spark Jobs

Total Uptime: 18 min
Scheduling Mode: FAIR
Active Jobs: 1
Completed Jobs: 10

Event Timeline

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
10	show at HiveUtils.scala:1156	2018/06/08 15:04:39	4.0 min	1/416	66132/859906

Observation

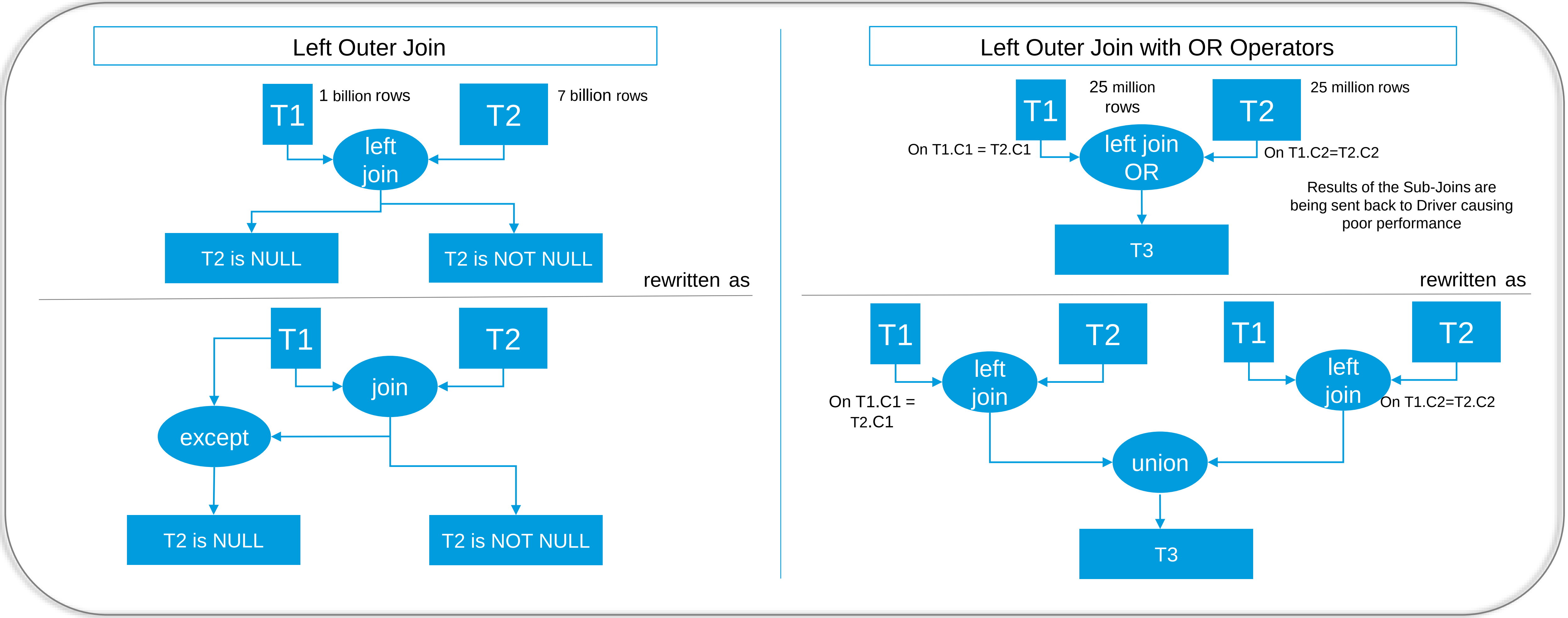
With the default shuffle partitions of 200, the Join Stage was running with too many tasks causing performance overhead

Resolution

Reduce the spark.sql.shuffle.partitions settings to a lower threshold

Learnings

Optimize wide transformations



Resolution

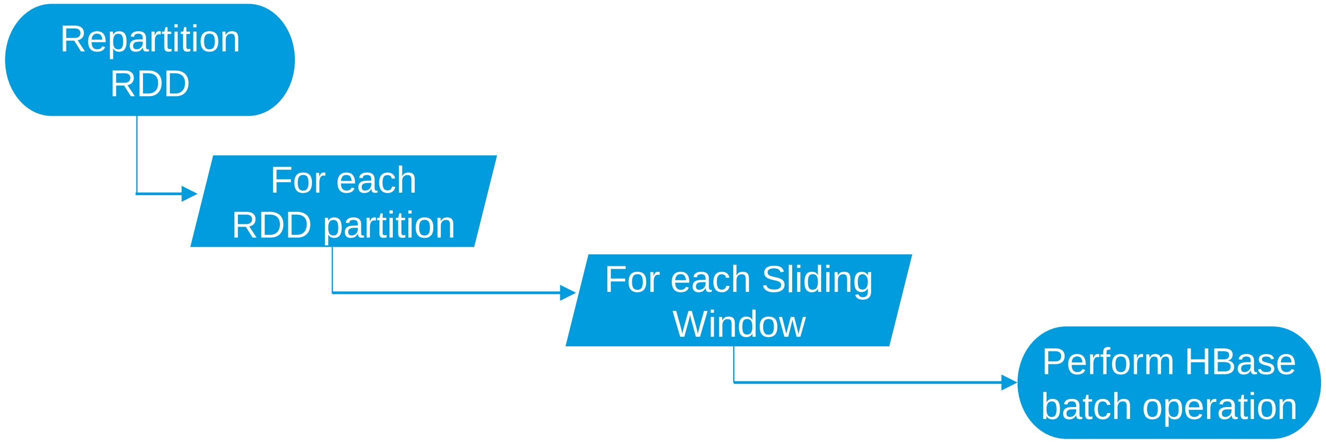
- Convert expensive left joins to combination of light weight join and except/union etc..,

Learnings

Optimize throughput for HBase Spark Connection

Observations

- Batch puts and gets slow due to HBase overloaded connections
- Since our HBase row was wide, HBase operations for partitions containing larger groups were slow



Example

```
Val rePartitionedRDD: RDD[Event] =
filledRDD.repartition(2000)
…..
…..
groupedEventRDD.mapPartitions( p =>
p.sliding(2000,2000)..foreach(
Create Hbase Connection
Batch Hbase Read or Write
Close Hbase Connection
)
)
```

Resolution

- Implemented sliding window for HBase Operations to reduce HBase connection overload

Tools & Utilities

Behavioral Driven Development

- While Unit tests are more about the implementation, BDD emphasizes more on the behavior of the code
- Writing “Specifications” in pseudo-English.
- Enables testing at external touch-points of your application

Feature : Identify the activity related to an event

Scenario: Should perform an iteration on events and join to activity table and identify the activity name

Given I have a set of events
cookie_id:String	page_id:String	last_actvty:String
263FHFBCBBCBV	login_provide	review_next_page
HFFDJFLUFBFNJL	home_page	provide_credent

And I have a Activity table
last_activity_id:String	activity_id:String	activity_name:String
review_next_page	1494300886856	Reviewing Next Page
provide_credent	2323232323232	Provide Credentials

When I implement Event Activity joins

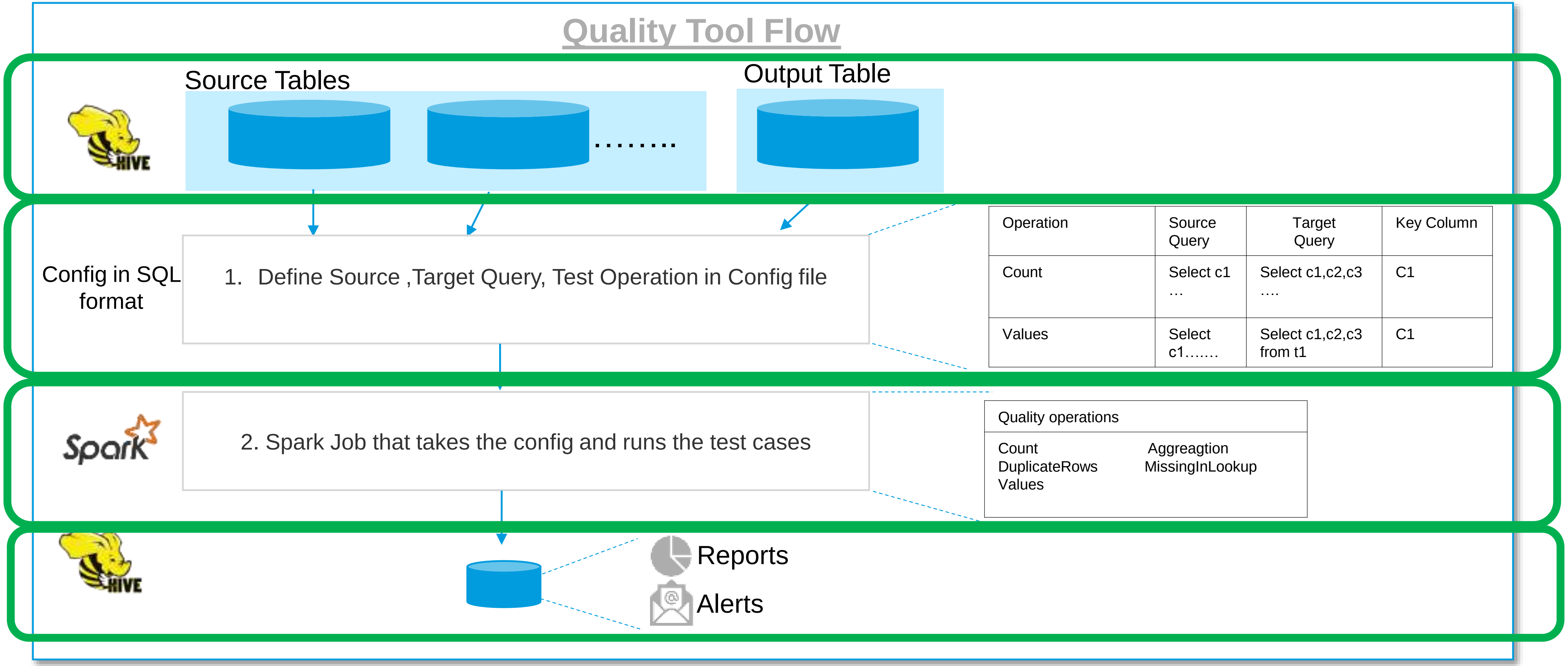
Then the final result is
cookie_id:String	activity_id:String	activity_name:String
last_activity_id:String	activity_id:String	activity_name:String
263FHFBCBBCBV	1494300886856	Reviewing Next Page
HFFDJFLUFBFNJL	2323232323232	Provide Credentials

pseudo code

```
import cucumber.api.scala.{EN, ScalaDsl}
import cucumber.api.DataTable
import org.scalatest.Matchers

Given("""^I have a set of events$""") { (data:DataTable) =>
   eventdataDF = dataTableToDataFrame(data)
 }

Given("""^I have a Activity table$""") { (data:DataTable) =>
   activityDataDF = dataTableToDataFrame(data)
 }

When("""^I implement Event Activity joins$"""){ () =>
   eventActivityDF = Activity.findAct(eventdataDF, activityDataDF) } }

Then("""^the final result is $"""){ (expectedData:DataTable) =>
   val expectedDf = dataTableToDataFrame(expectedData)
   val resultDF = eventActivityDF
   resultDF.except(expectedDF).count
```

Data Quality Tool

- Config driven automated tool written in Spark for Quality Control
- Used extensively during functional testing of the application and once live, used as quality check for our data pipeline
- Feature to compare tables (schema agnostic and at Scale) for data validation and helping engineers troubleshoot effectively



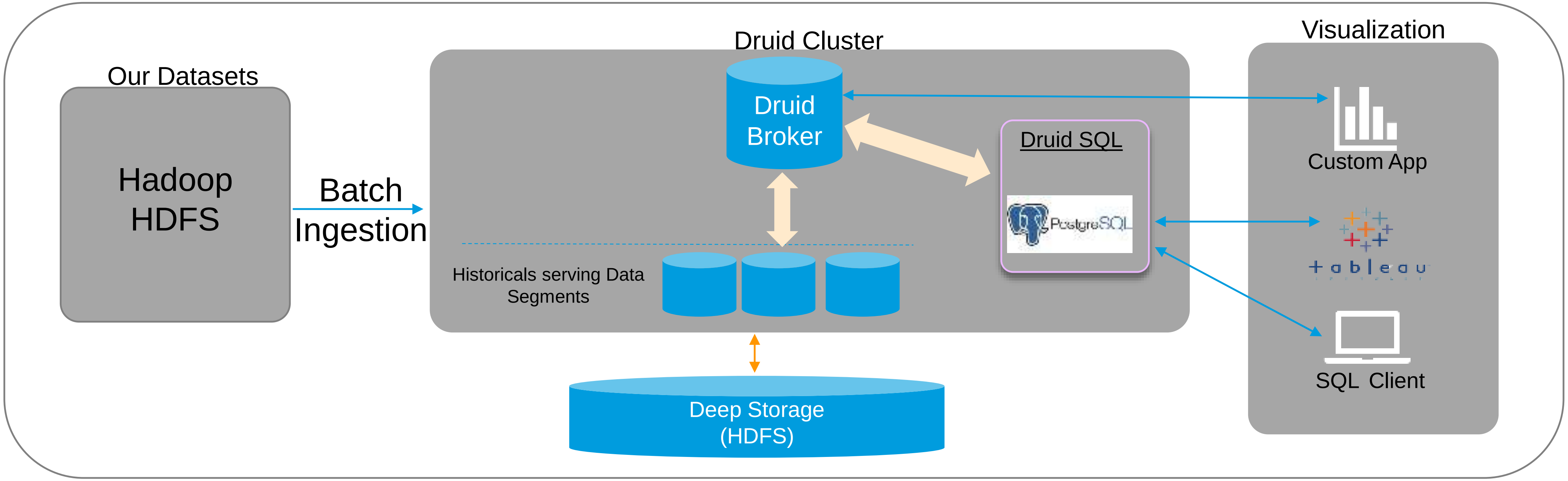
Operation	Source Query	Target Query	Key Column
Count	Select c1 …	Select c1,c2,c3 ….	C1
Values	Select c1…….	Select c1,c2,c3 from t1	C1

Quality operations	
Count DuplicateRows Values	Aggreagtion MissingInLookup

Druid Integration with BI

- Druid is an open-source time series data store designed for sub-second queries on real-time and historical data. It is primarily used for business intelligence queries on event data*
- Traditional Databases did not scale and perform with Tableau dashboards (for many use cases)
- Enable Tableau dashboards with Druid as the serving platform
- Live connection from tableau to druid avoids getting limited by storage at any layer.

Visualization at scale



*.from http://Druid.io

Conclusion

- Spark Applications on Yarn (Hortonworks distribution).
- Spark jobs were easy to write and had excellent performance (though little hard to troubleshoot)
- Spark-HBase optimization improved performance
- Pre-aggregated datasets to Elasticsearch
- Denormalized datasets to Druid
- Pushed lowest-granularity denormalized datasets to Druid
- Behavior Driven Development a great add-on for Product-backed applications

QUESTIONS?