

Netflix: Integrating Spark At Petabyte Scale

Ashwin Shankar
Cheolsoo Park



Outline

1. Netflix big data platform
2. Spark @ Netflix
3. Multi-tenancy problems
4. Predicate pushdown
5. S3 file listing
6. S3 insert overwrite
7. Zeppelin, Ipython notebooks
8. Use case (Pig vs. Spark)

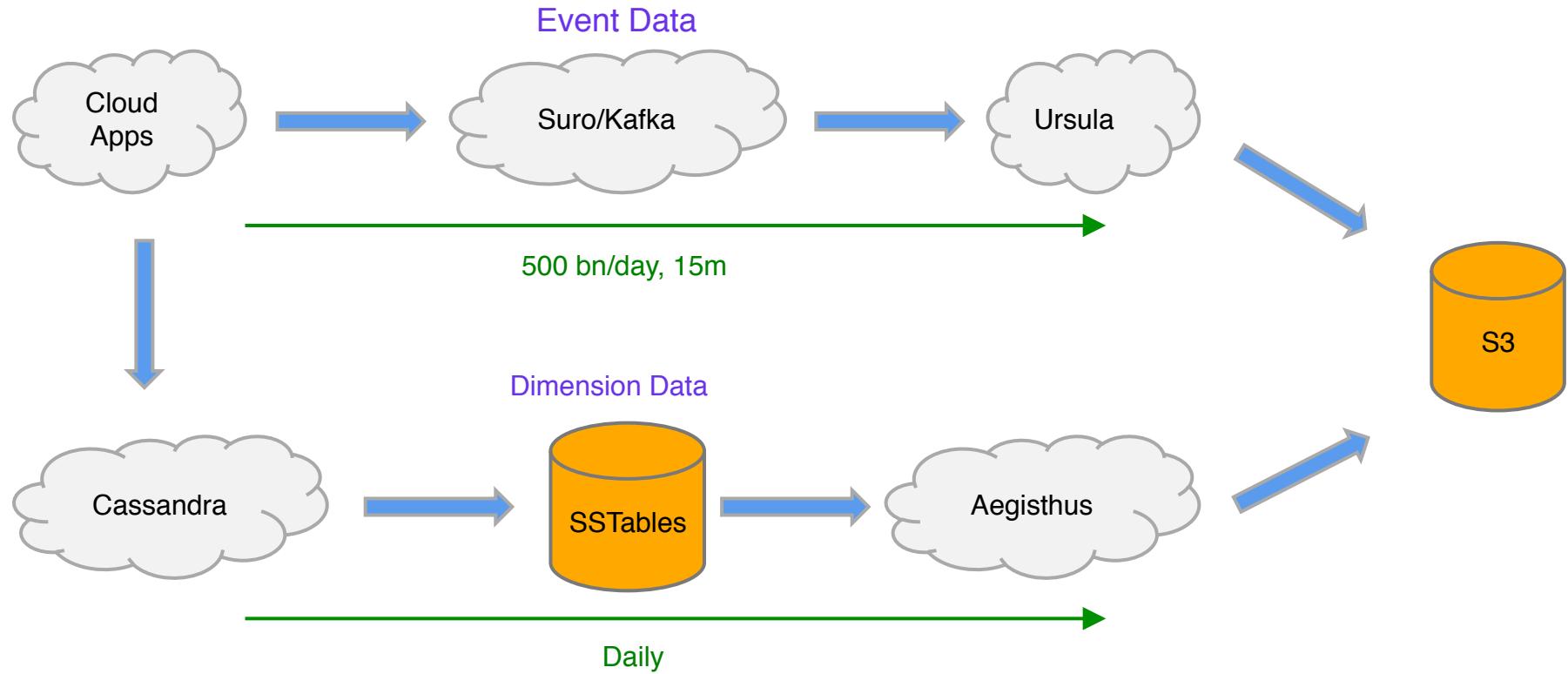
Netflix

Big Data Platform

APACHE:
BIG DATA
EUROPE



Netflix data pipeline



Netflix big data platform

APACHE:
BIG DATA
EUROPE

Tools

Big Data API/Portal



Service

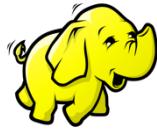


Metacat

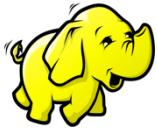
Clients



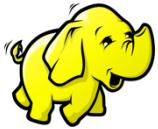
Clusters



Prod



Test



Adhoc



Prod



Test

Data
Warehouse



Our use cases

- Batch jobs (Pig, Hive)
 - ETL jobs
 - Reporting and other analysis
- Interactive jobs (Presto)
- Iterative ML jobs (Spark)

Spark @ Netflix

APACHE:
BIG DATA
EUROPE



Mix of deployments

- Spark on Mesos
 - Self-serving AMI
 - Full BDAS (Berkeley Data Analytics Stack)
 - Online streaming analytics
- **Spark on YARN**
 - Spark as a service
 - YARN application on EMR Hadoop
 - Offline batch analytics

Spark on YARN

- Multi-tenant cluster in AWS cloud
 - Hosting MR, Spark, Druid
- EMR Hadoop 2.4 (AMI 3.9.0)
- D2.4xlarge ec2 instance type
- 1000+ nodes (100TB+ total memory)

Deployment



s3://bucket/spark/1.4/spark-1.4.tgz, spark-defaults.conf (spark.yarn.jar=1440304023)

s3://bucket/spark/1.5/spark-1.5.tgz, spark-defaults.conf (spark.yarn.jar=1440443677)

Timestamp
1440443677



/spark/1.4/1440304023/spark-assembly.jar
/spark/1.4/1440989711/spark-assembly.jar

/spark/1.5/1440443677/spark-assembly.jar
/spark/1.5/1440720326/spark-assembly.jar



```
name: spark
version: 1.5
tags: ['type:spark', 'ver:1.5']
jars:
  - 's3://bucket/spark/1.5/spark-1.5.tgz'
```



Download latest tarball
From S3 via Genie

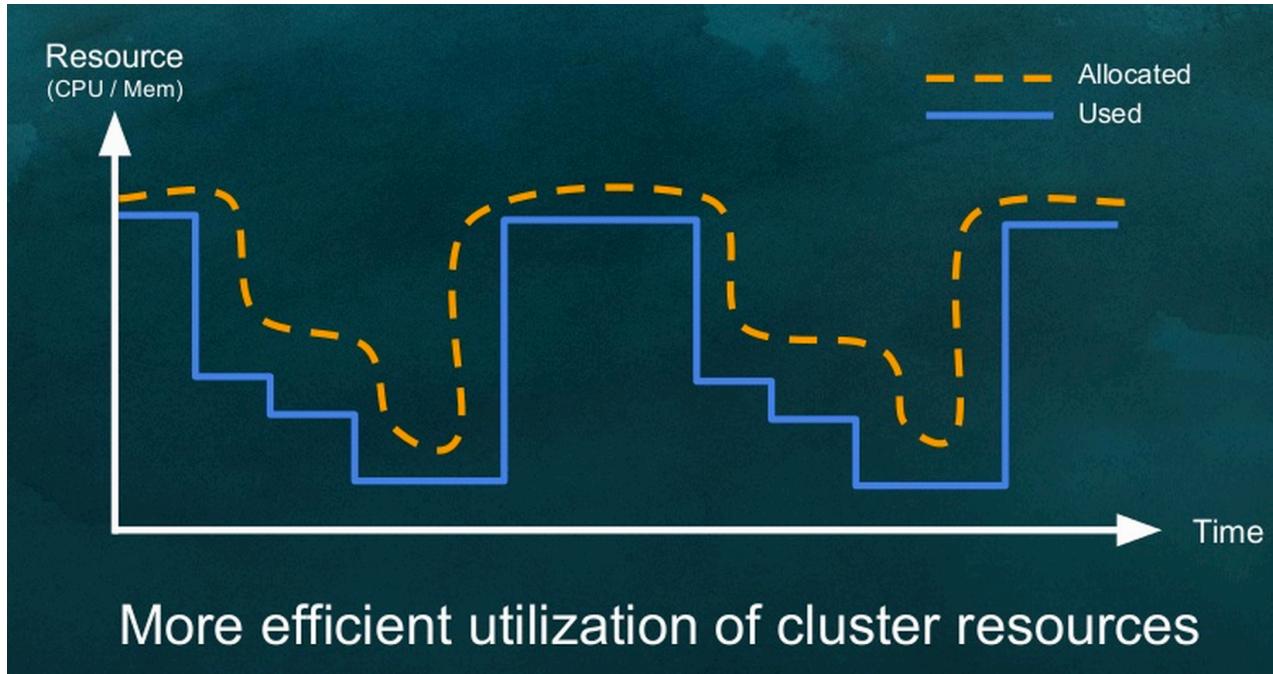
Advantages

1. Automate deployment.
2. Support multiple versions.
3. Deploy new code in 15 minutes.
4. Roll back bad code in less than a minute.

Multi-tenancy Problems



Dynamic allocation



Courtesy of “*Dynamic allocate cluster resources to your Spark application*” at Hadoop Summit 2015

Dynamic allocation

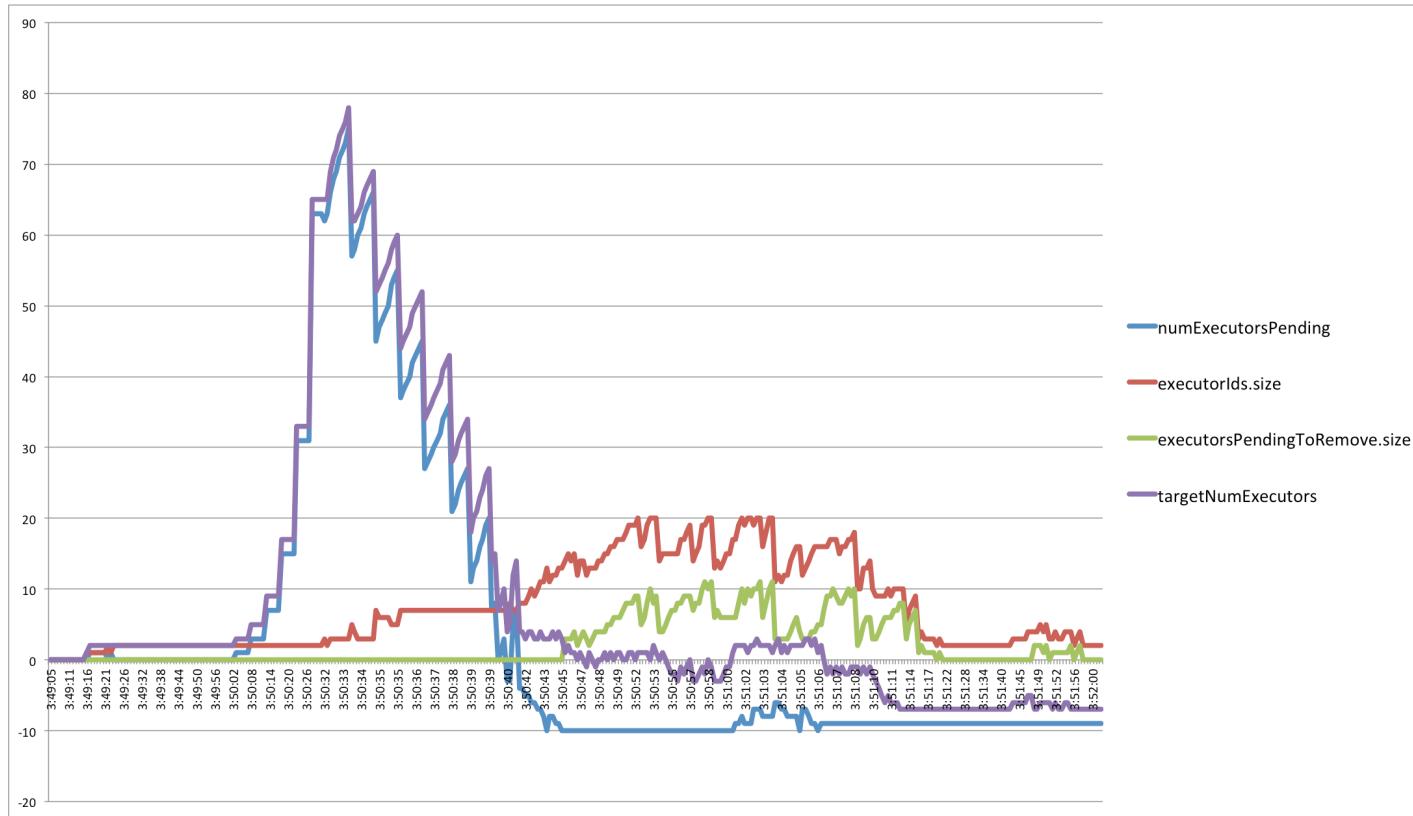
```
// spark-defaults.conf
spark.dynamicAllocation.enabled                      true
spark.dynamicAllocation.executorIdleTimeout          5
spark.dynamicAllocation.initialExecutors            3
spark.dynamicAllocation.maxExecutors                500
spark.dynamicAllocation.minExecutors                3
spark.dynamicAllocation.schedulerBacklogTimeout     5
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout 5
spark.dynamicAllocation.cachedExecutorIdleTimeout   900
```

```
// yarn-site.xml
yarn.nodemanager.aux-services
  • spark_shuffle, mapreduce_shuffle
yarn.nodemanager.aux-services.spark_shuffle.class
  • org.apache.spark.network.yarn.YarnShuffleService
```

Problem 1: SPARK-6954

“Attempt to request a negative number of executors”

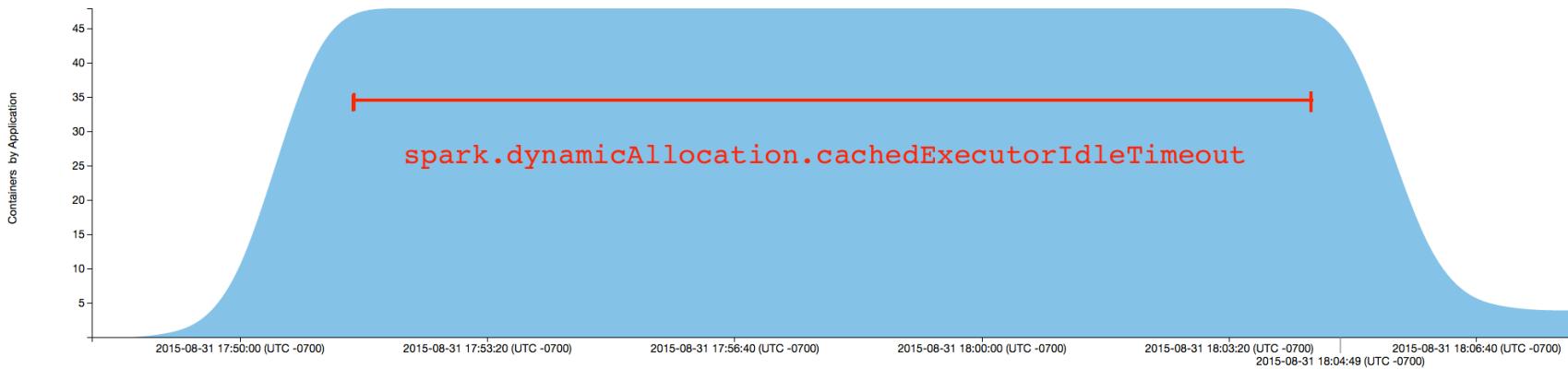
SPARK-6954



Problem 2: SPARK-7955

“Cached data lost”

SPARK-7955



```
val data = sqlContext
    .table("dse.admin_genie_job_d")
    .filter($"dateint">>=20150601 and $"dateint"<=20150830)
data.persist
data.count
```

Problem 3: SPARK-7451, SPARK-8167

“Job failed due to preemption”

SPARK-7451, SPARK-8167

- Symptom
 - Spark executors/tasks randomly fail causing job failures.
- Cause
 - Preempted executors/tasks are counted as failures.
- Solution
 - Preempted executors/tasks should be considered as killed.

Problem 4: YARN-2730

“Spark causes MapReduce jobs to get stuck”

- Symptom
 - MR jobs get timed out during localization when running with Spark jobs on the same cluster.
- Cause
 - NM localizes one job at a time. Since Spark runtime jar is big, localizing Spark jobs may take long, blocking MR jobs.
- Solution
 - Stage Spark runtime jar on HDFS with high replication.
 - Make NM localize multiple jobs concurrently.

Predicate Pushdown

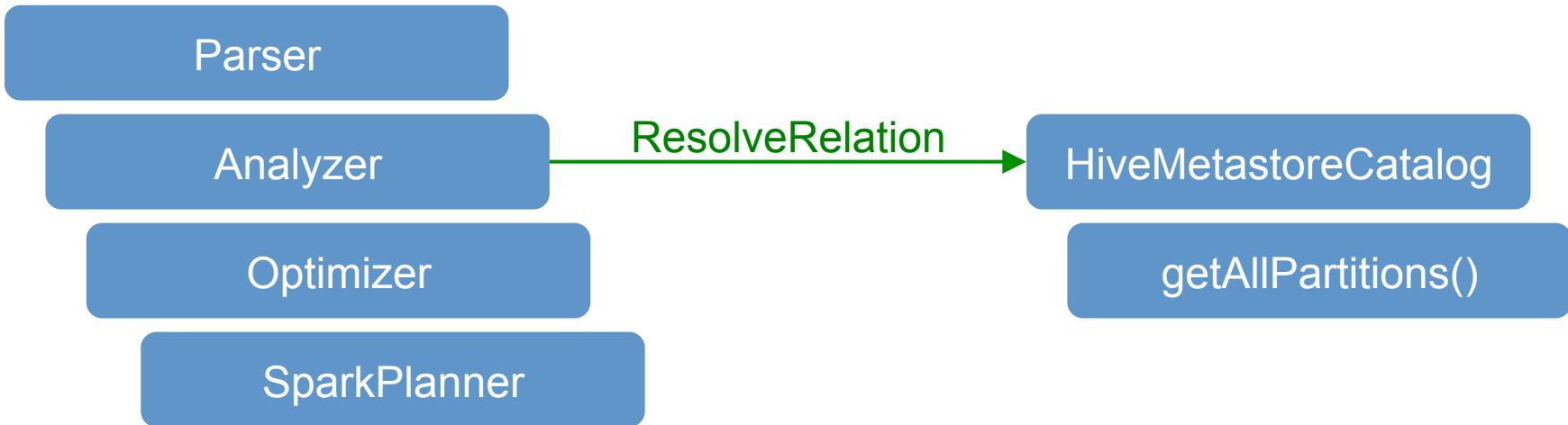
APACHE:
BIG DATA
EUROPE



Predicate pushdown

Case	Behavior
Predicates with partition cols on partitioned table	Single partition scan
Predicates with partition and non-partition cols on partitioned table	Single partition scan
No predicate on partitioned table e.g. <code>sqlContext.table("nccp_log").take(10)</code>	Full scan
No predicate on non-partitioned table	Single partition scan

Predicate pushdown for metadata

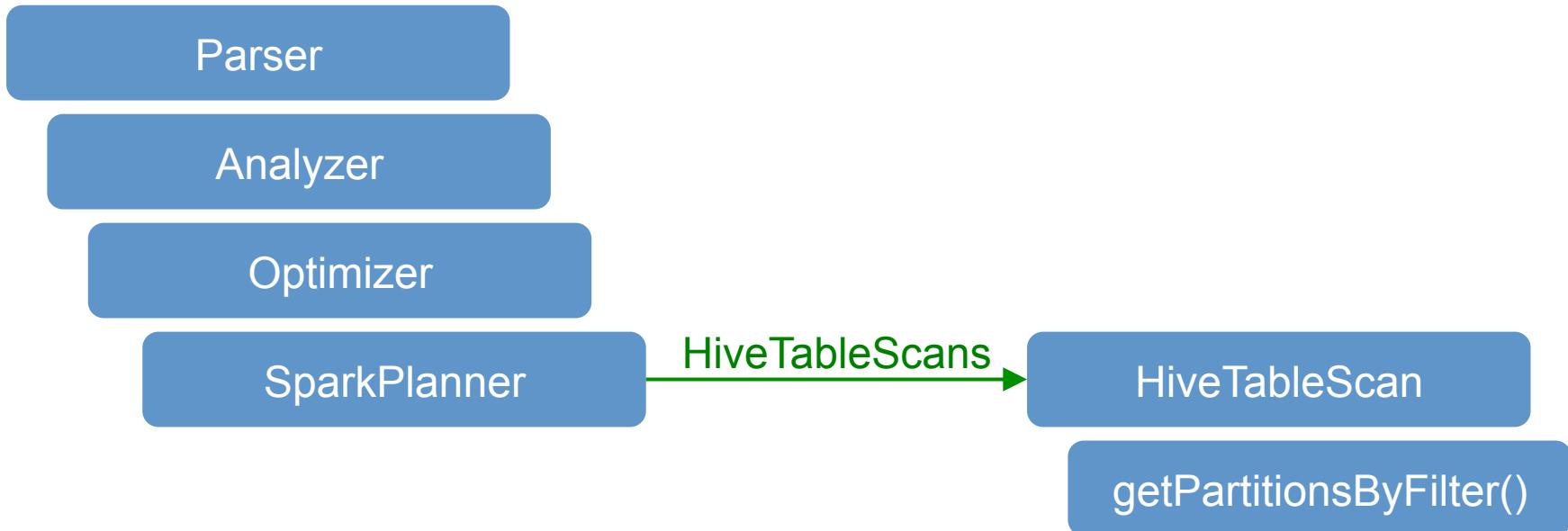


What if your table has 1.6M partitions?

SPARK-6910

- Symptom
 - Querying against heavily partitioned Hive table is slow.
- Cause
 - Predicates are not pushed down into Hive metastore, so Spark does full scan for table metadata.
- Solution
 - Push down binary comparison expressions via `getPartitionsByfilter()` in to Hive metastore.

Predicate pushdown for metadata



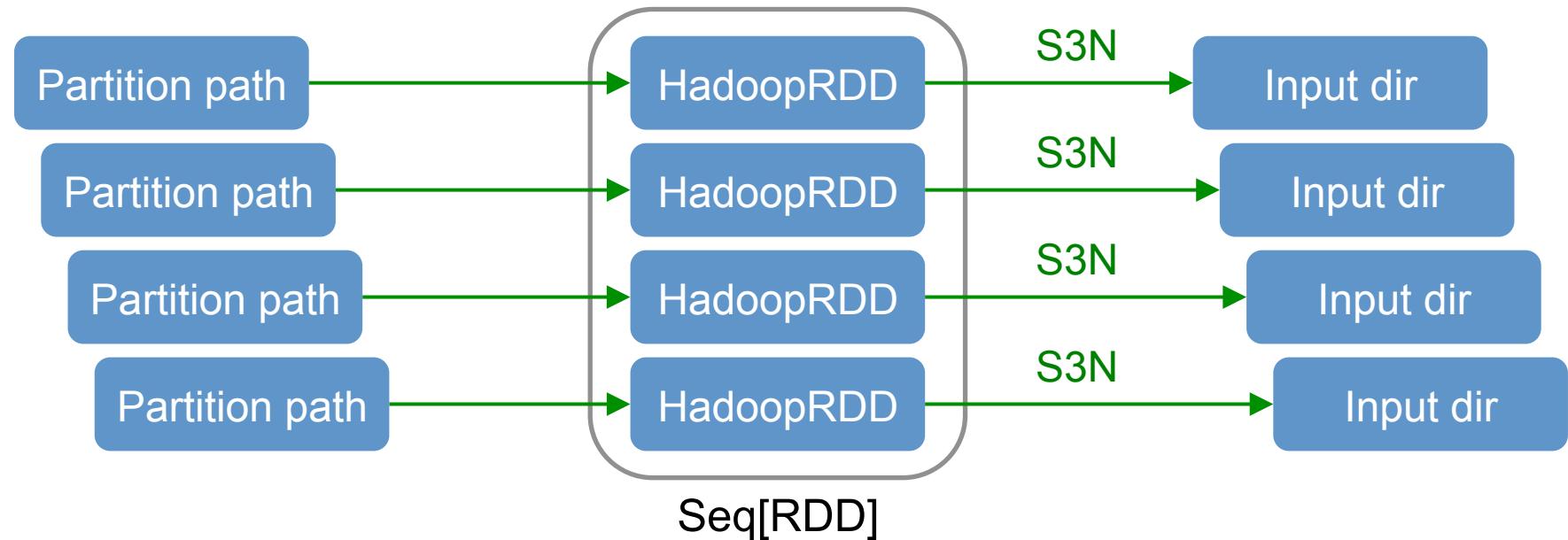
S3 File Listing



Input split computation

- `mapreduce.input.fileinputformat.list-status.num-threads`
 - The number of threads to use list and fetch block locations for the specified input paths.
- Setting this property in Spark jobs doesn't help.

File listing for partitioned table

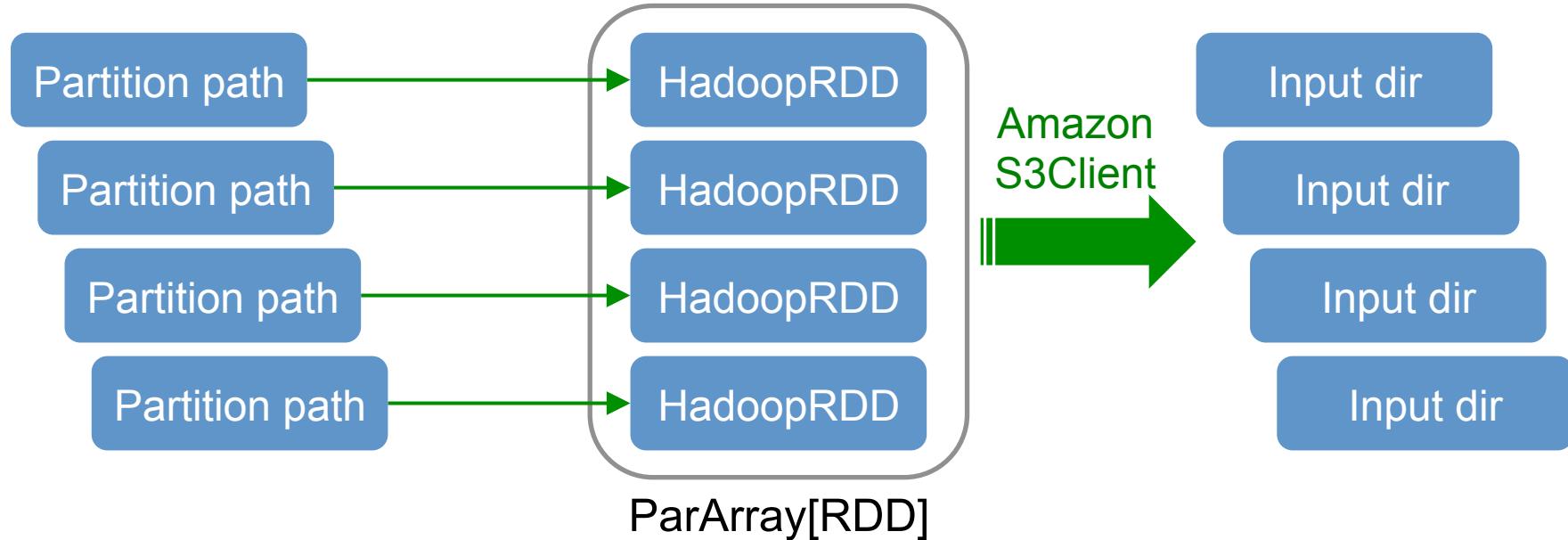


Sequentially listing input dirs via S3N file system.

SPARK-9926, SPARK-10340

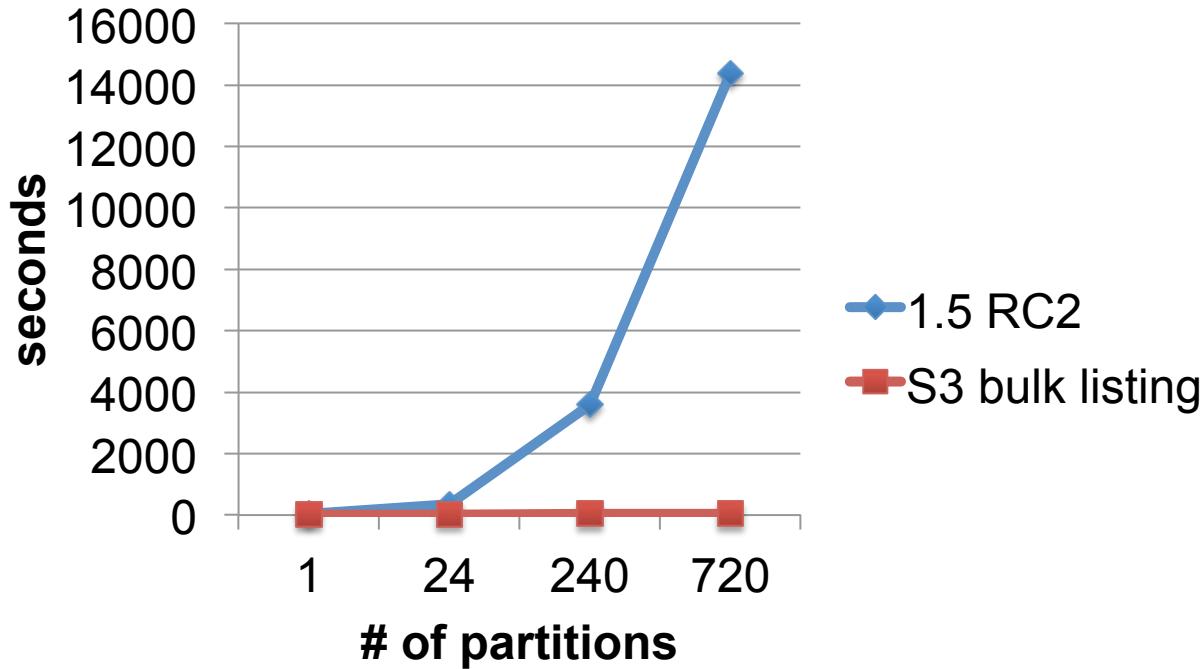
- Symptom
 - Input split computation for partitioned Hive table on S3 is slow.
- Cause
 - Listing files on a per partition basis is slow.
 - S3N file system computes data locality hints.
- Solution
 - Bulk list partitions in parallel using AmazonS3Client.
 - Bypass data locality computation for S3 objects.

S3 bulk listing



Bulk listing input dirs in parallel via AmazonS3Client.

Performance improvement



```
SELECT * FROM nccp_log WHERE dateint=20150801 and hour=0 LIMIT 10;
```

S3

Insert Overwrite



Problem 1: Hadoop output committer

- How it works:
 - Each task writes output to a temp dir.
 - Output committer renames first successful task's temp dir to final destination.
- Problems with S3:
 - S3 rename is copy and delete.
 - S3 is eventual consistent.
 - FileNotFoundException during “rename.”

S3 output committer

- How it works:
 - Each task writes output to local disk.
 - Output committer copies first successful task's output to S3.
- Advantages:
 - Avoid redundant S3 copy.
 - Avoid eventual consistency.

Problem 2: Hive insert overwrite

- How it works:
 - Delete and rewrite existing output in partitions.
- Problems with S3:
 - S3 is eventual consistent.
 - FileAlreadyExistException during “rewrite.”

Batchid pattern

- How it works:
 - Never delete existing output in partitions.
 - Each job inserts a unique subpartition called “batchid.”
- Advantages:
 - Avoid eventual consistency.

Zeppelin Ipython Notebooks

APACHE:
BIG DATA
EUROPE



Big data portal

- One stop shop for all big data related tools and services.
- Built on top of Big Data API.

The screenshot shows the Netflix Big Data Portal interface. On the left, there's a sidebar with navigation links: cheolsoop (selected), Inbox (1 notification), Log Out, Home - Query, Dashboard, Schema Search, S3 Browser, Automic/UC4, Notebooks (selected), and Schema Browser. The main area is titled "Notebooks". It displays three instances:

- PySpark**: Spark programming model to Python. Status: Instance running since 9/10/2015, 8:54:17 AM. 1 CPU, 8 GB of memory, 50 GB of disk space with environment variables UPLOAD_SYNC_OPTS="--exclude /home/ipynb/notebooks/bdp-examples/", FOLDER_PATH=/home/ipynb/notebooks/. Buttons: Relaunch with new parameters, View Log, Open, Relaunch, Kill.
- Zeppelin**: Data-driven, interactive and collaborative documents with SQL, Scala and more. Status: Instance running since 9/10/2015, 9:22:31 AM. 1 CPU, 8 GB of memory, 50 GB of disk space with environment variables UPLOAD_SYNC_OPTS="--exclude /home/ipynb/notebooks/2AWNZXSG8/", FOLDER_PATH=/home/ipynb/notebooks/. Buttons: Relaunch with new parameters, View Log, Open, Relaunch, Kill.
- IPython**: Python shell for interactive computing. Status: No instance running.

Out of box examples

Jupyter PySparkExamples (read only)

In [20]:

```
# Quick visualization
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

sns.set_context('talk')
sns.set_style('white')
sns.set_ticks()
device_streams = zip(top10)
ypos = np.arange(len(device_streams))
plt.barh(ypos, streams, align='center')
plt.yticks(ypos, device)
sns.despine(offset=10, trim=True)
```

Zeppelin Notebook - Interpreter

Connected

Save Cancel

```
// What are the most popular titles on Netflix? Let's start by querying some play data for 1 day
val plays = sqlContext.sql("select profile_id, country_iso_code, standard_sanitized_duration_sec/3600 as play_hrs, show_title_id from dse.loc_acct_device_ttl_sum where region_auto='US'")
// Now group it by country and show
// How do we quantify popularity?
// -- number of plays?
// -- number of users?
// -- number of hours?
// Let's try all three and compare
```

```
popularity.filter(popularity("country_iso_code")=="US").orderBy(popularity("SUM(play_hrs)").desc).take(25)
// you can also reference columns with the $ operator
popularity.filter(popularity("country_iso_code")=="US").orderBy(popularity("APPROXIMATE COUNT(DISTINCT profile_id)").desc).take(25)
popularity.filter(popularity("country_iso_code")=="US").orderBy(popularity("COUNT(profile_id)").desc).take(25)
```

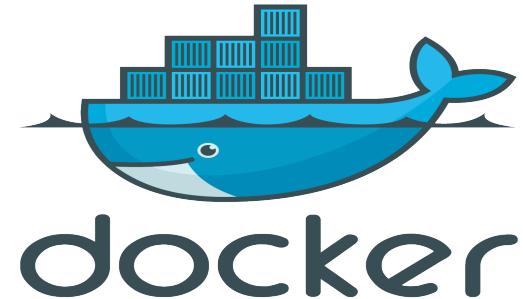
```
// notice that the query has to recompute the data each time ... so let's persist our intermediate "popularity" variable
popularity.persist()
res28: popularity.type = [country_iso_code: string, show_title_id: int, SUM(play_hrs): double, APPROXIMATE COUNT(DISTINCT profile_id): bigint, COUNT(profile_id): bigint]
Took 1 seconds
```

```
// We can't really tell what the titles are b/c they only have an id. Let's join or popularity to some title metadata
val titles = sqlContext.table("dse.ttl_show_id")
val popularity_metadata = popularity.join(titles,titles("show_title_id")===popularity("show_title_id"))
popularity_metadata.filter(popularity_metadata("country_iso_code")=="US").orderBy(popularity_metadata("SUM(play_hrs)").desc).take(25)
```

```
// The display still is ugly, let's use Zeppelin's markup to display this as a table:
print(s"""
    +-----+
    |table
    |country_iso_code\show_descrip_content_type\tgenre\tplay_hours\number_profiles\number_plays
    |+---+
    """)
popularity_metadata.orderBy(popularity_metadata("SUM(play_hrs)").desc).take(10000).foreach(println(s"""
    +-----+
    |table
    |country_iso_code\show_descrip_content_type\tgenre\tplay_hours\number_profiles\number_plays
    |+---+
    """))
```

On demand notebooks

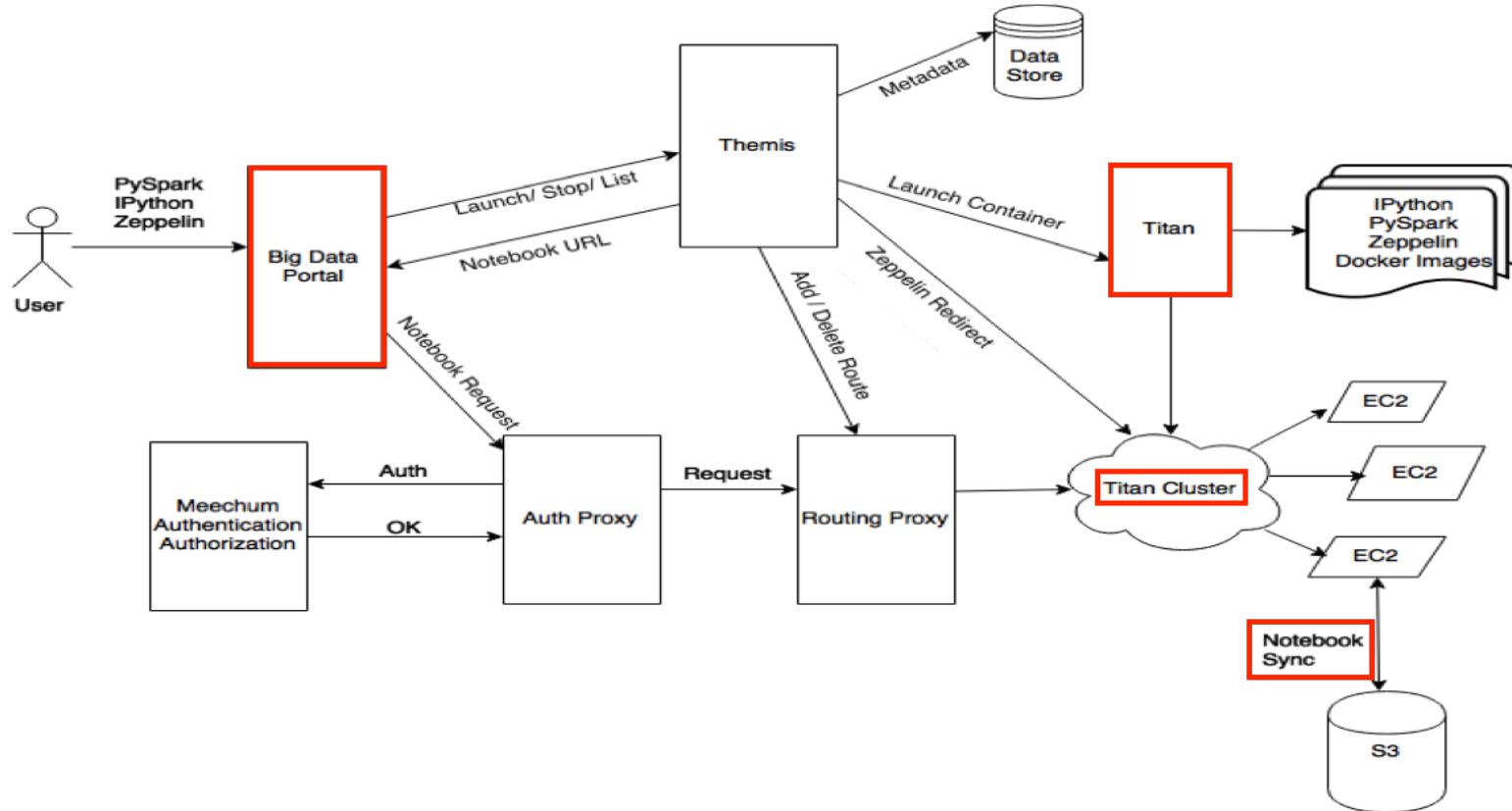
- Zero installation
 - Dependency management via Docker
- Notebook persistence
- Elastic resources



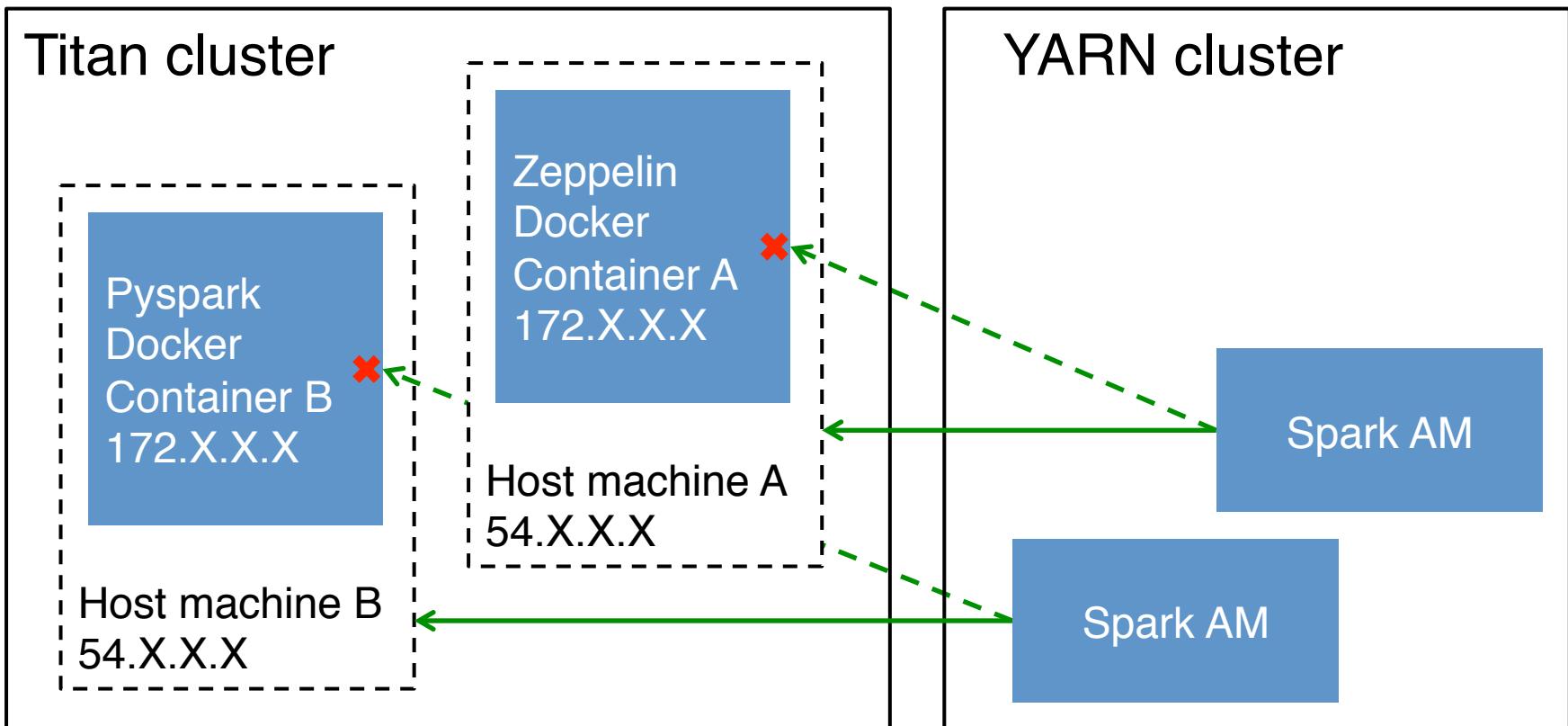
Quick facts about Titan

- Task execution platform leveraging Apache Mesos.
- Manages underlying EC2 instances.
- Process supervision and uptime in the face of failures.
- Auto scaling.

Notebook Infrastructure



Ephemeral ports / --net=host mode

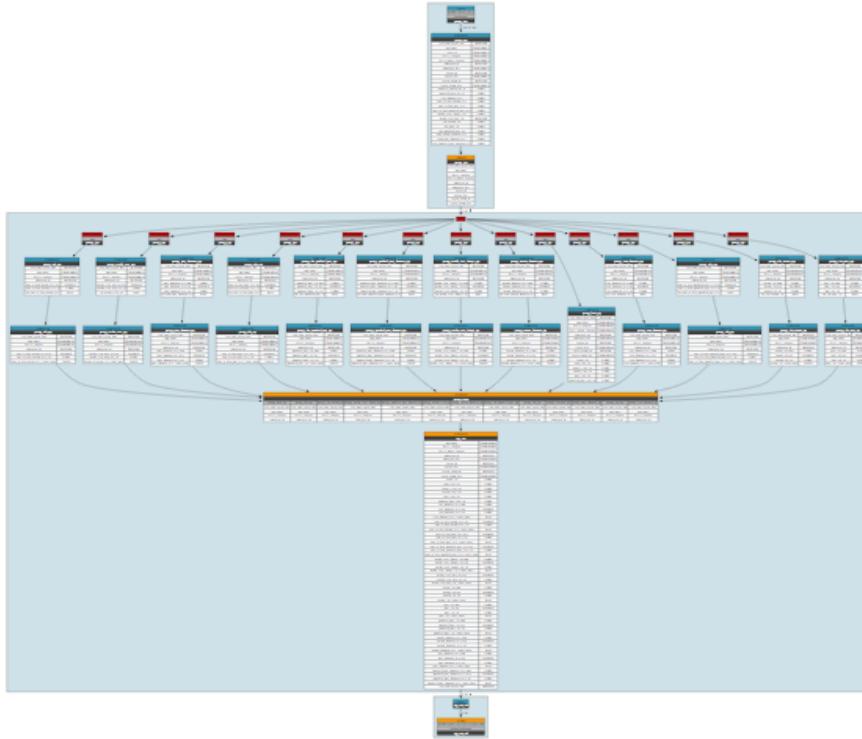


Use Case

Pig vs. Spark



Iterative job



Iterative job

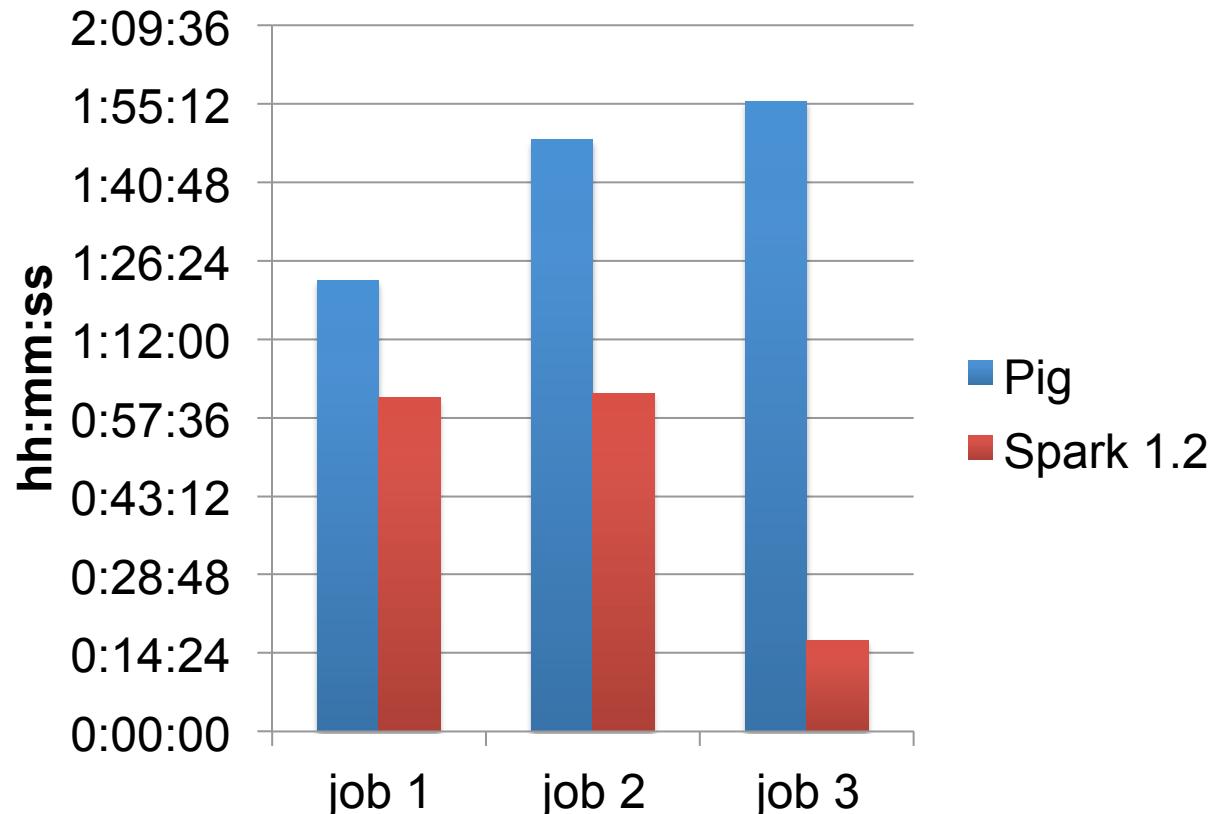
1. Duplicate data and aggregate them differently.



2. Merging aggregates back.



Performance improvement



Our contributions

SPARK-6018

SPARK-6662

SPARK-6909

SPARK-6910

SPARK-7037

SPARK-7451

SPARK-7850

SPARK-8355

SPARK-8572

SPARK-8908

SPARK-9270

SPARK-9926

SPARK-10001

SPARK-10340

Q&A

Thank You