Netflix: Integrating Spark At Petabyte Scale

Ashwin Shankar
Cheolsoo Park

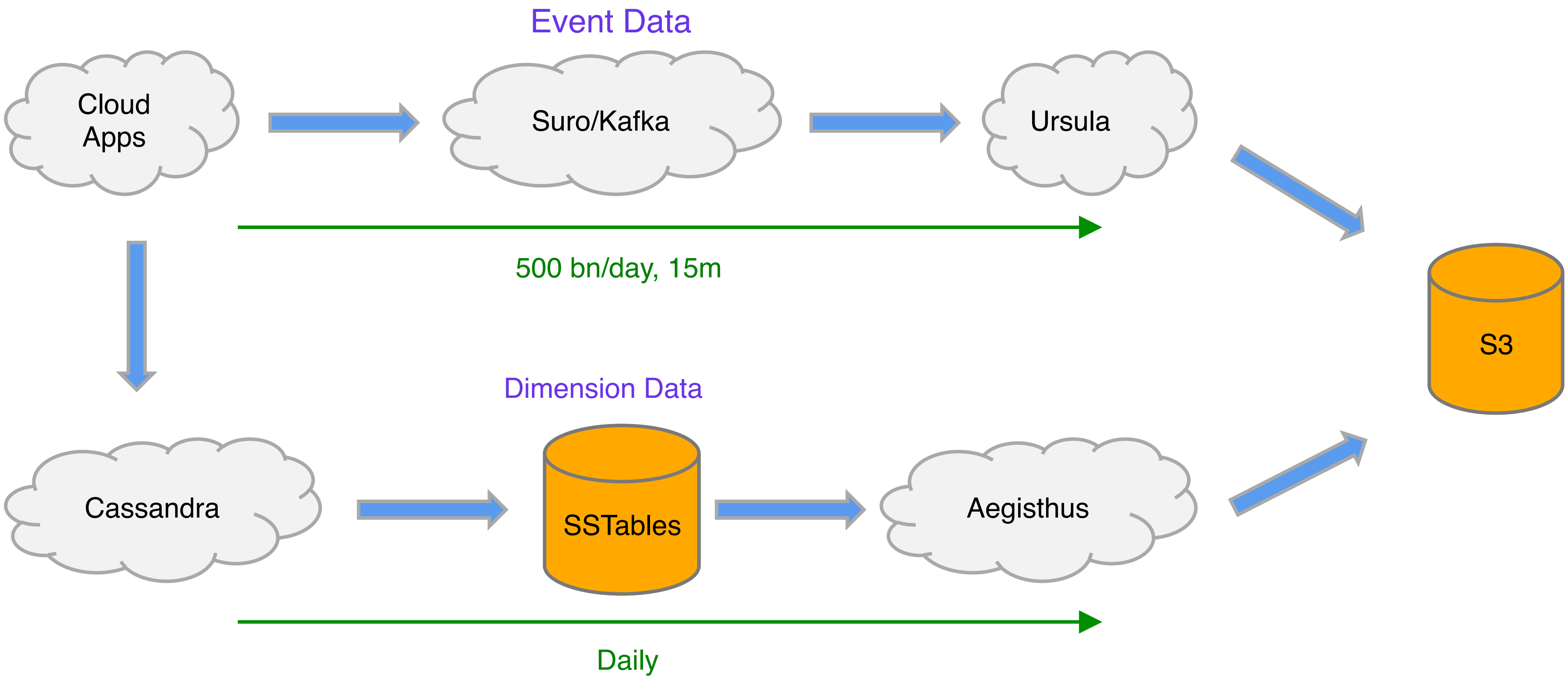
Outline

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	Gateways
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

Mix of deployments

- Spark on Mesos
  - Self-serving AMI
  - Full BDAS (**B**erkeley **D**ata **A**nalytics **S**tack)
  - 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

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

GENIE

```
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

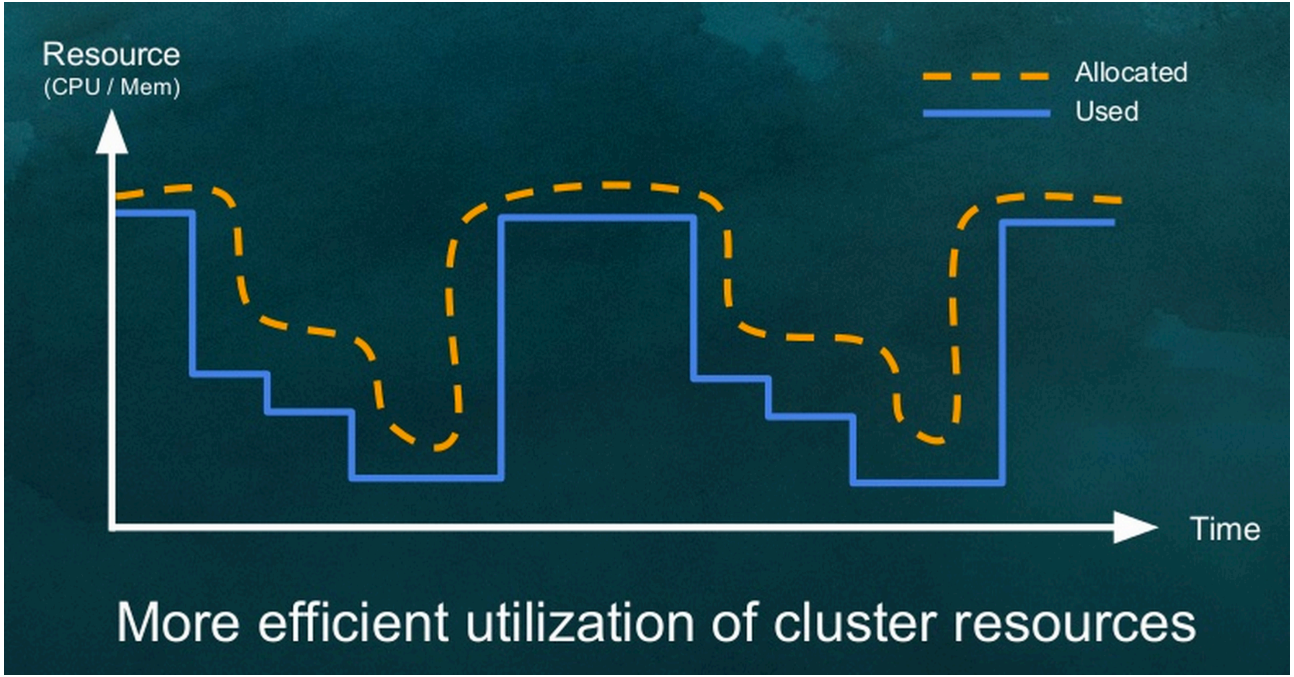
Advantages

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

APACHE: BIG_DATA EUROPE

Dynamic allocation



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

Dynamic allocation

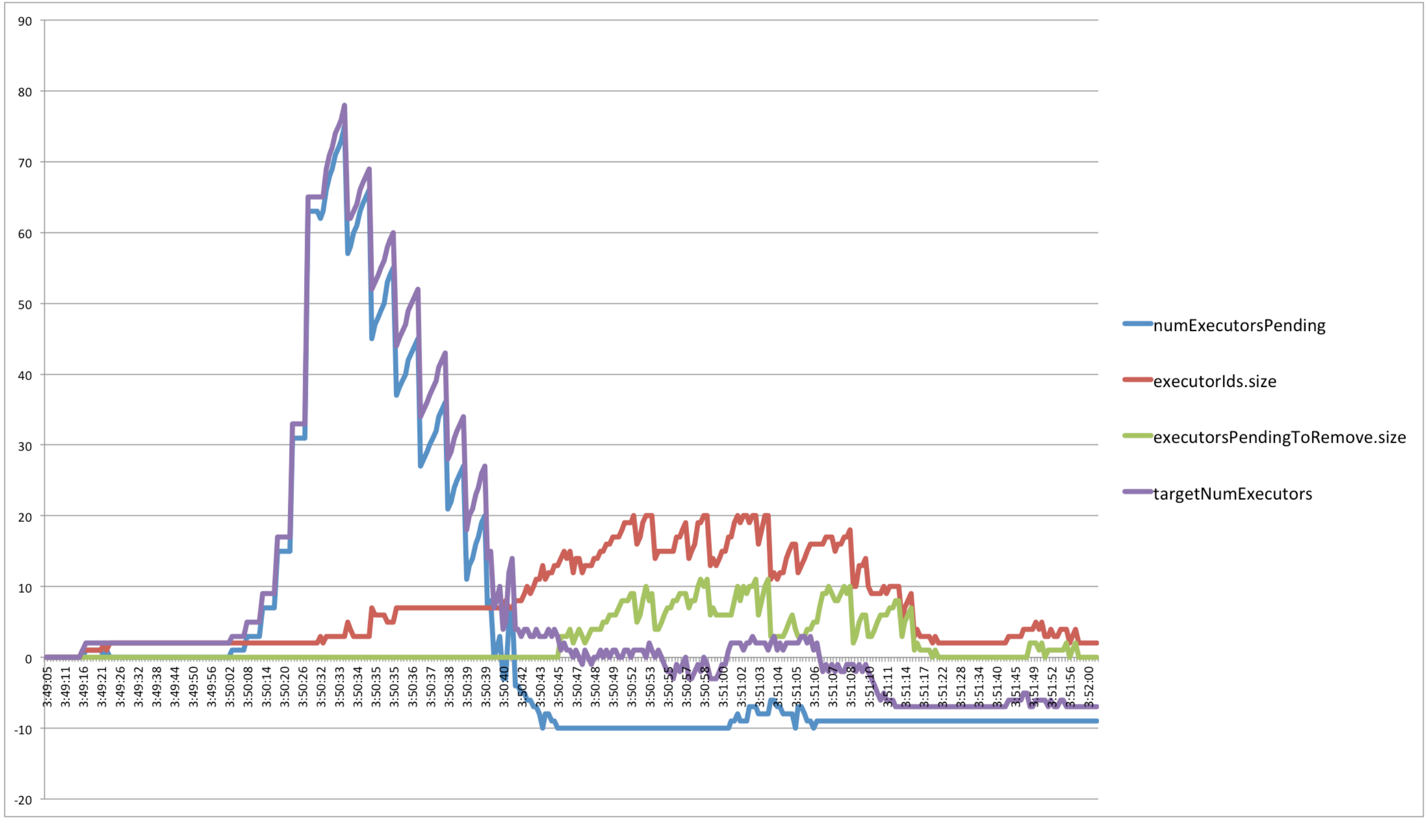
```
// spark-defaults.conf
spark.dynamicAllocation.enabled                          true
spark.dynamicAllocation.executorIdleTimeout              5
spark.dynamicAllocation.initialExecutors                 3
spark.dynamicAllocation.maxExecutors                     500
spark.dynamicAllocation.minExecutors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

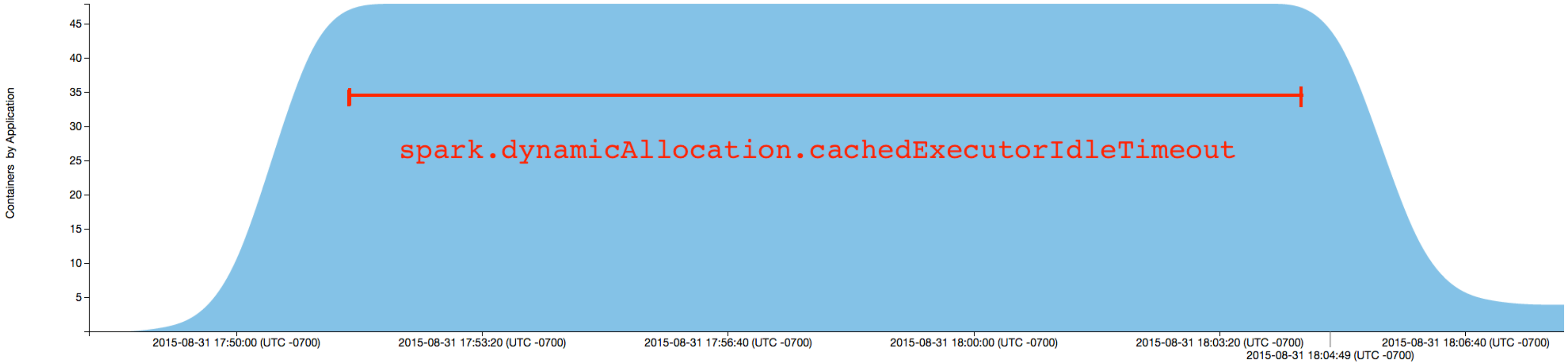


numExecutorsPending
executorIds.size
executorsPendingToRemove.size
targetNumExecutors
90
80
70
60
50
40
30
20
10
0
-10
-20

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”

YARN-2730

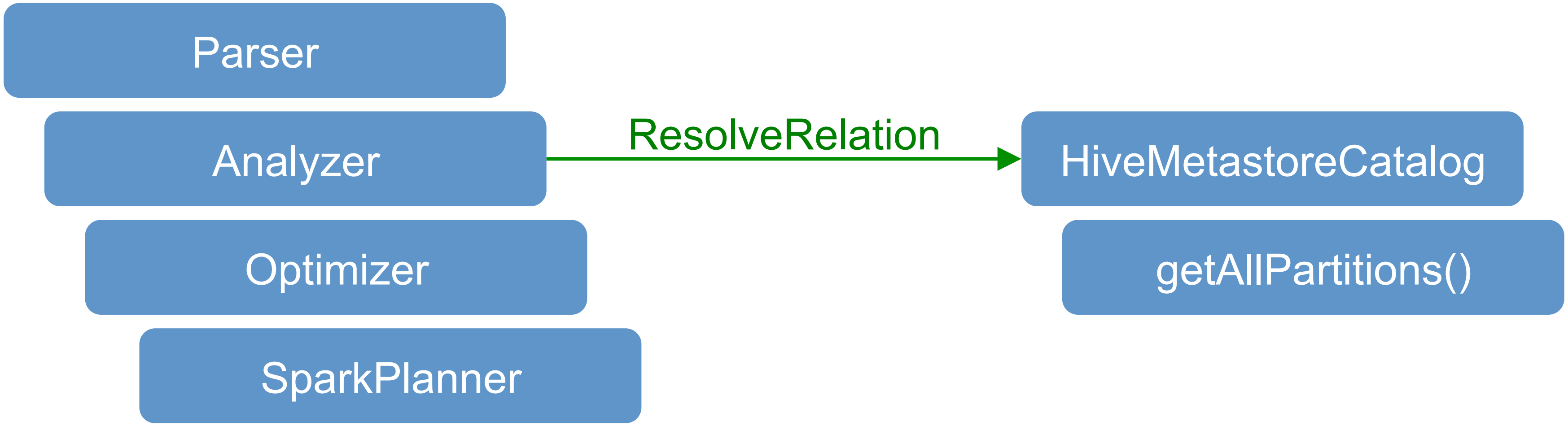
- Symptom
  - MR jobs get timed out during localization when running with Spark jobs on the same cluster.
- Cause
  - NM localizes one job at a time. Since Spark runtime jar is big, localizing Spark jobs may take long, blocking MR jobs.
- Solution
  - Stage Spark runtime jar on HDFS with high repliacation.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

Predicate Pushdown

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. sqlContext.table("nccp_log").take(10)	Full scan
No predicate on non-partitioned table	Single partition scan

Predicate pushdown for metadata

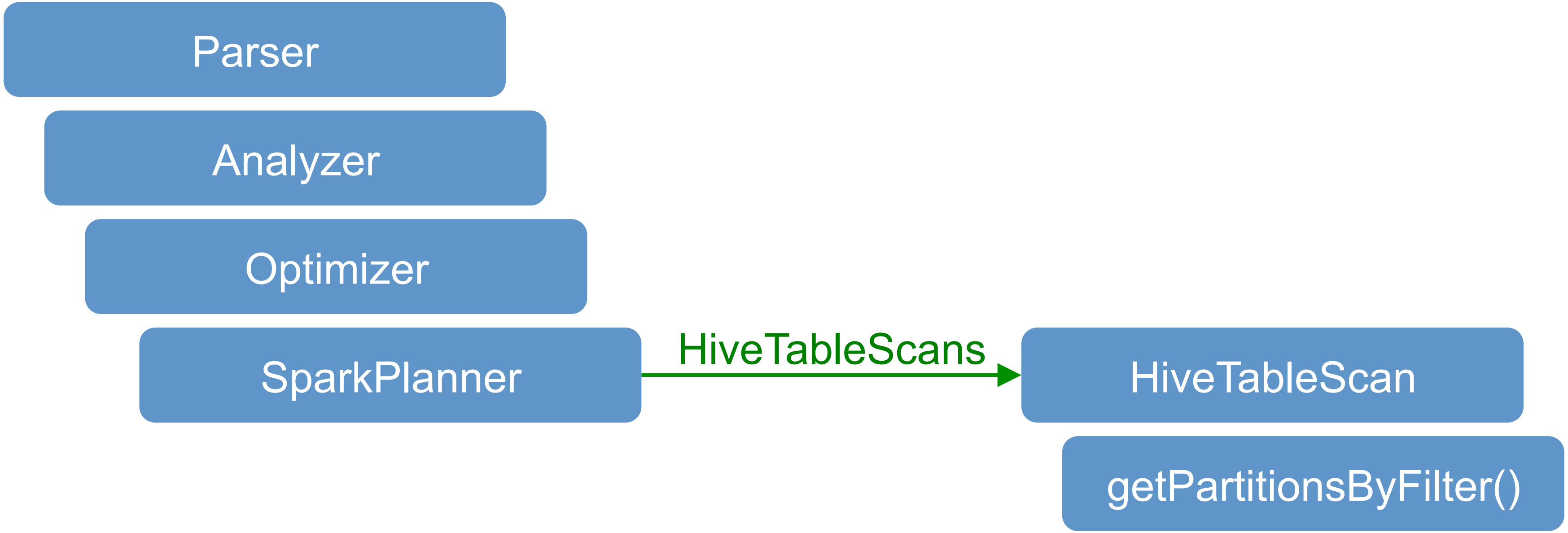


What if your table has 1.6M partitions?

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

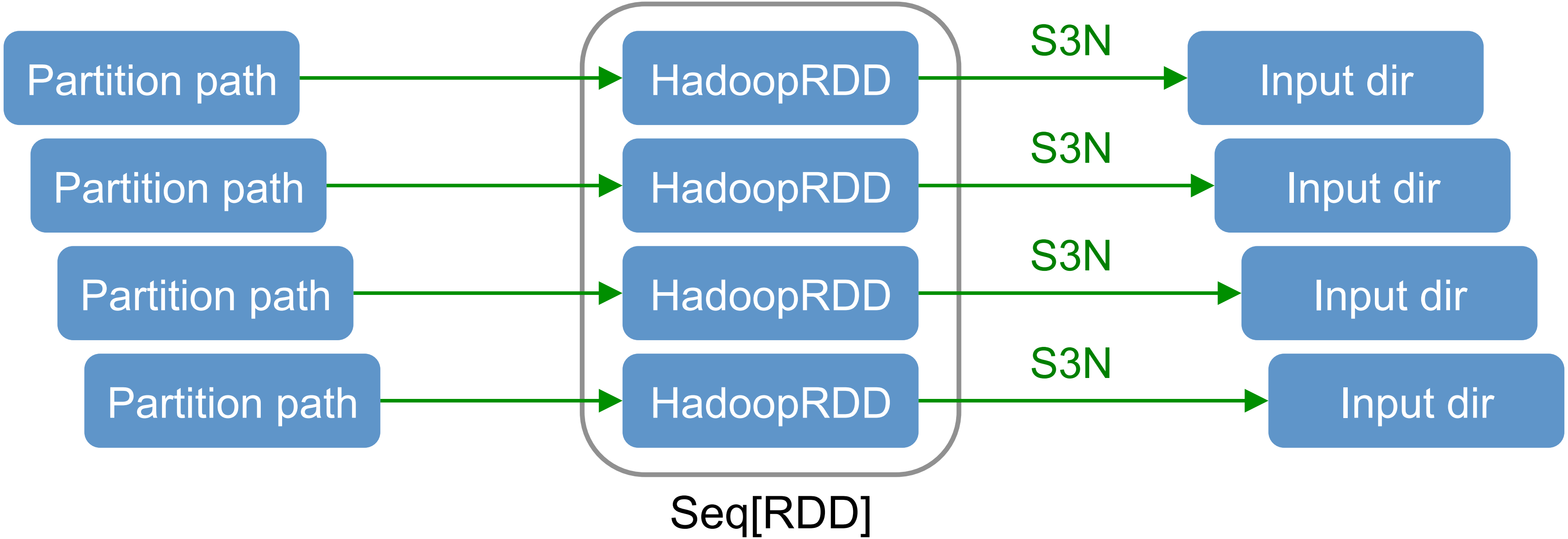


S3
File Listing

Input split computation

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

File listing for partitioned table

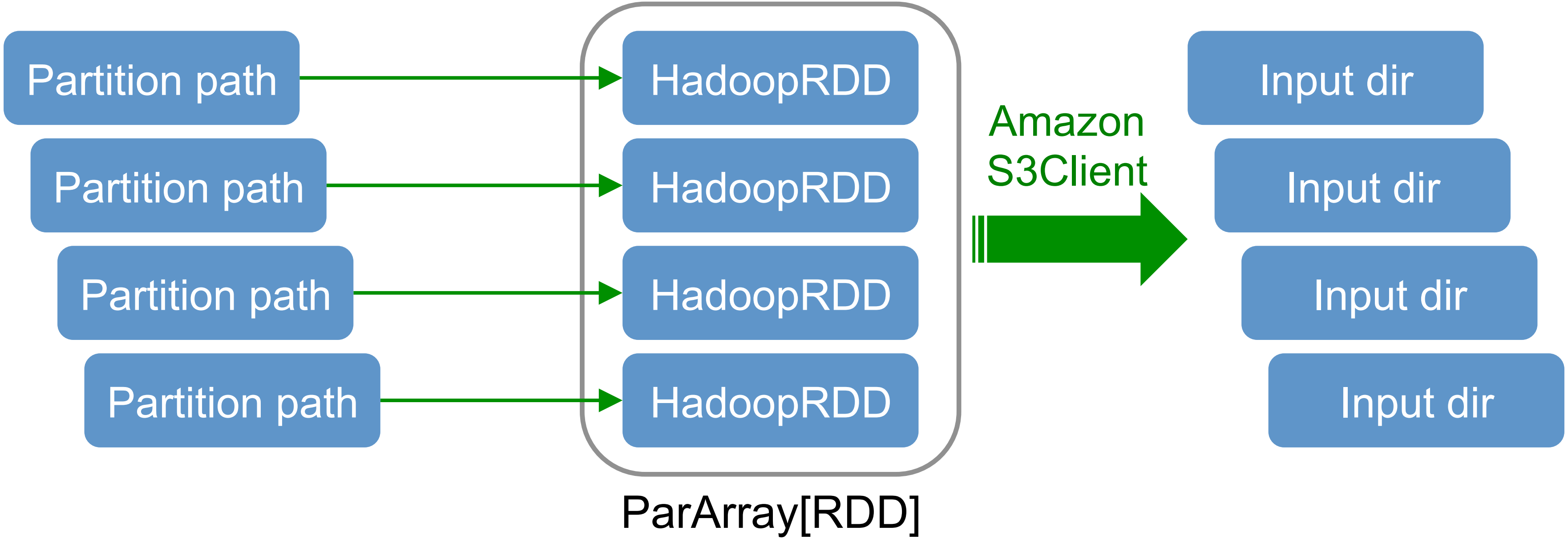


Sequentially listing input dirs via S3N file system.

SPARK-9926, SPARK-10340

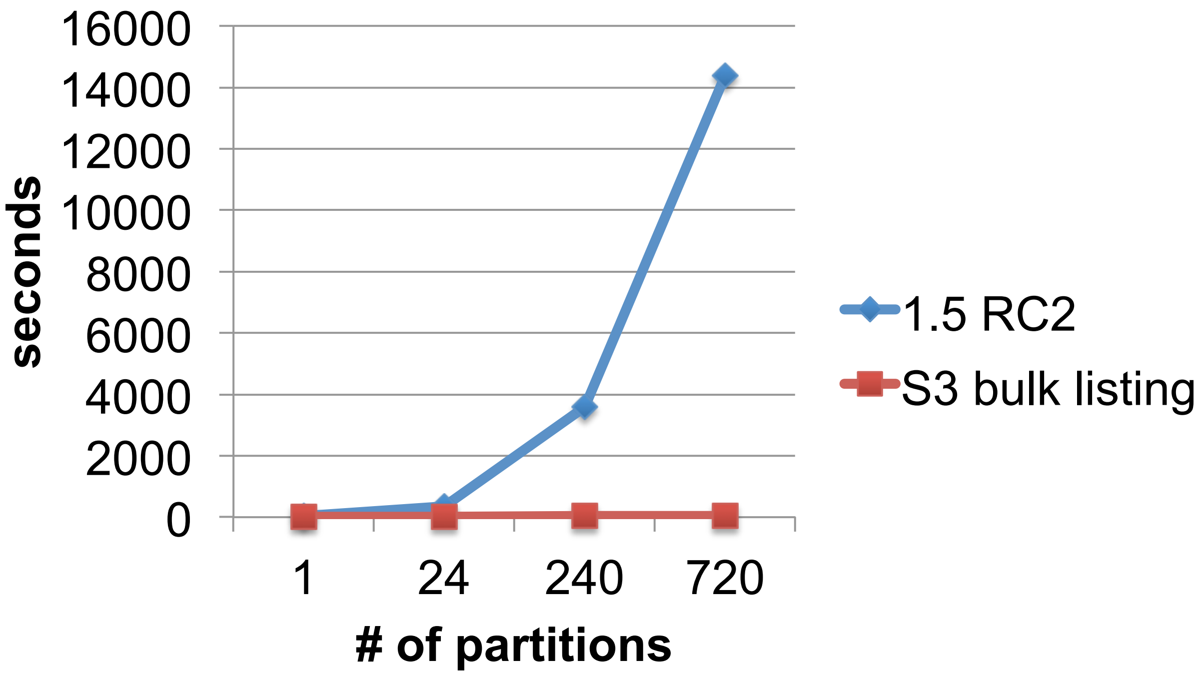
- Symptom
  - Input split computation for partitioned Hive table on S3 is slow.
- Cause
  - Listing files on a per partition basis is slow.
  - S3N file system computes data locality hints.
- Solution
  - Bulk list partitions in parallel using AmazonS3Client.
  - Bypass data locality computation for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

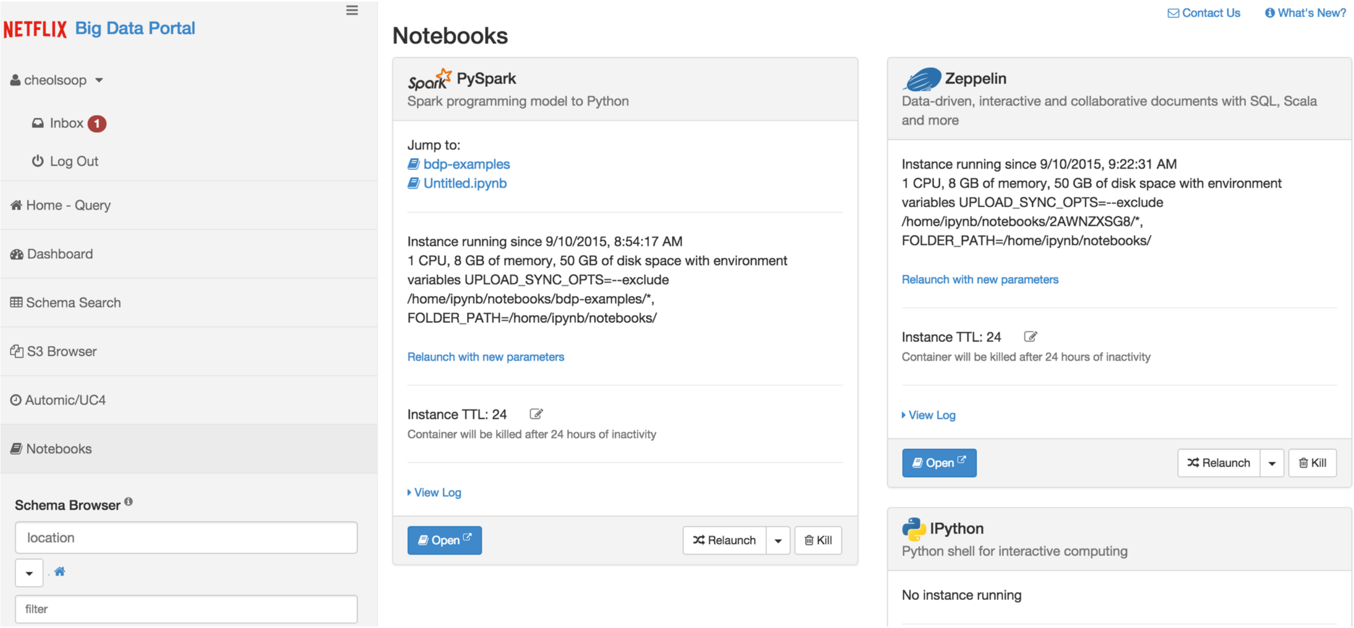
- How it works:
  - Never delete existing output in partitions.
  - Each job inserts a unique subpartition called “batchid.”
- Advantages:
  - Avoid eventual consistency.

Zeppelin
Ipython
Notebooks

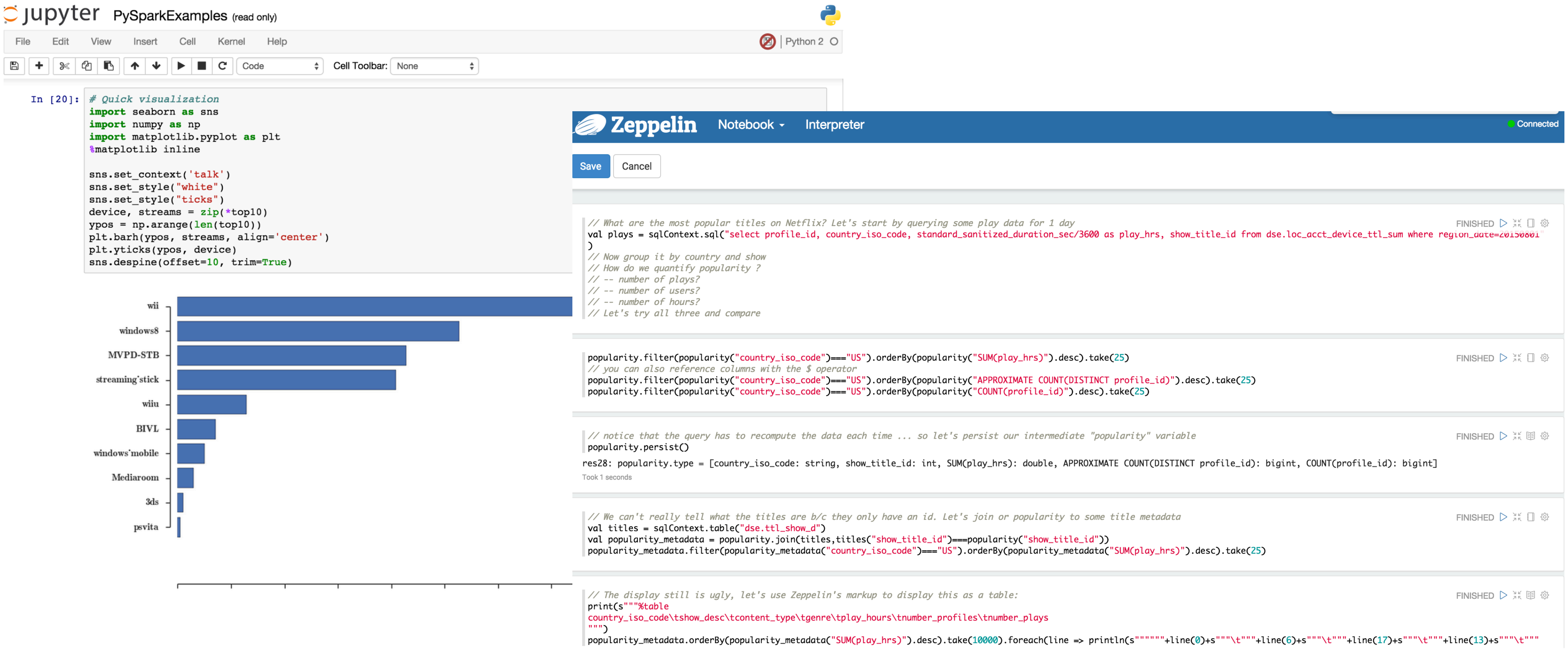
APACHE: BIG_DATA EUROPE

Big data portal

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



Out of box examples



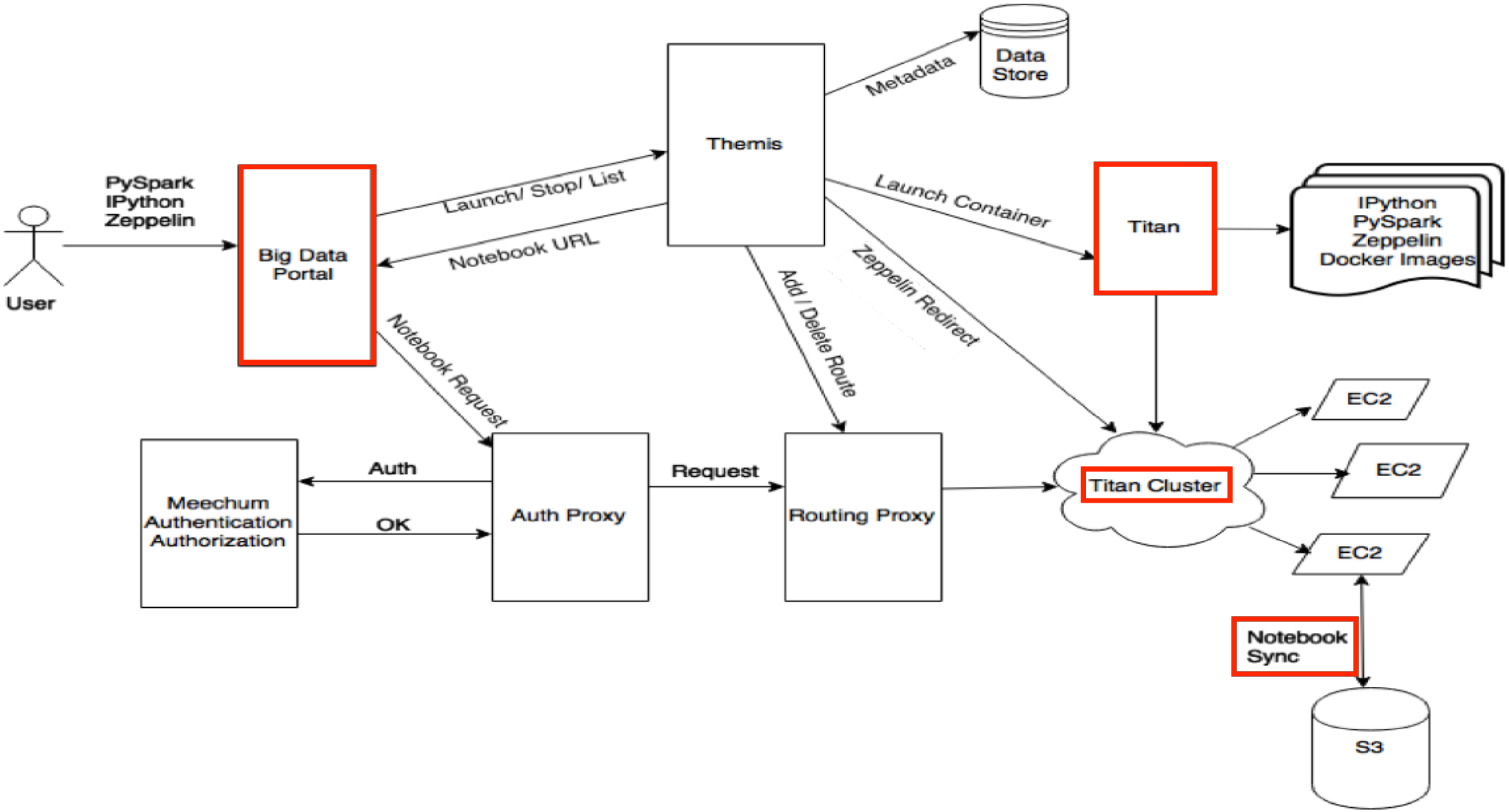
On demand notebooks

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

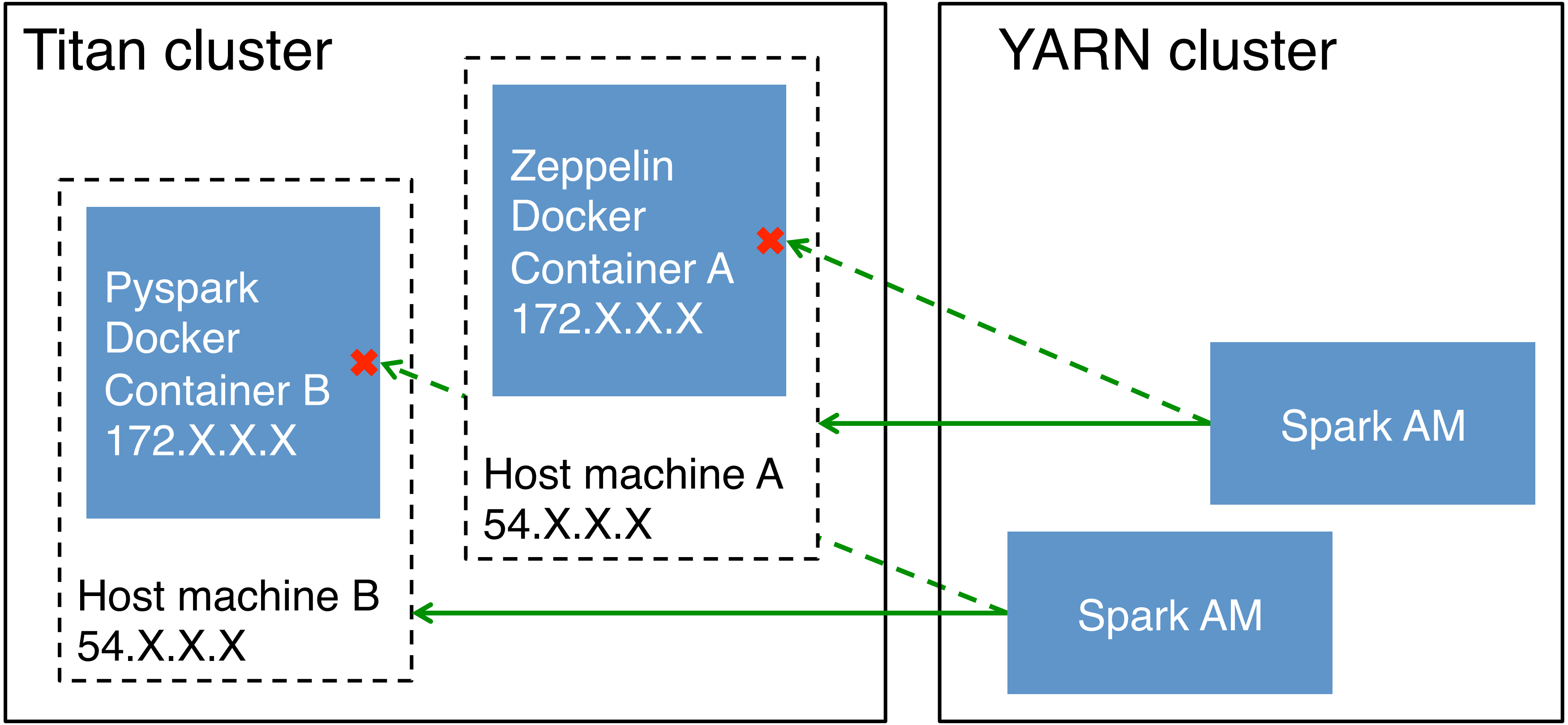
Quick facts about Titan

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

Notebook Infrastructure

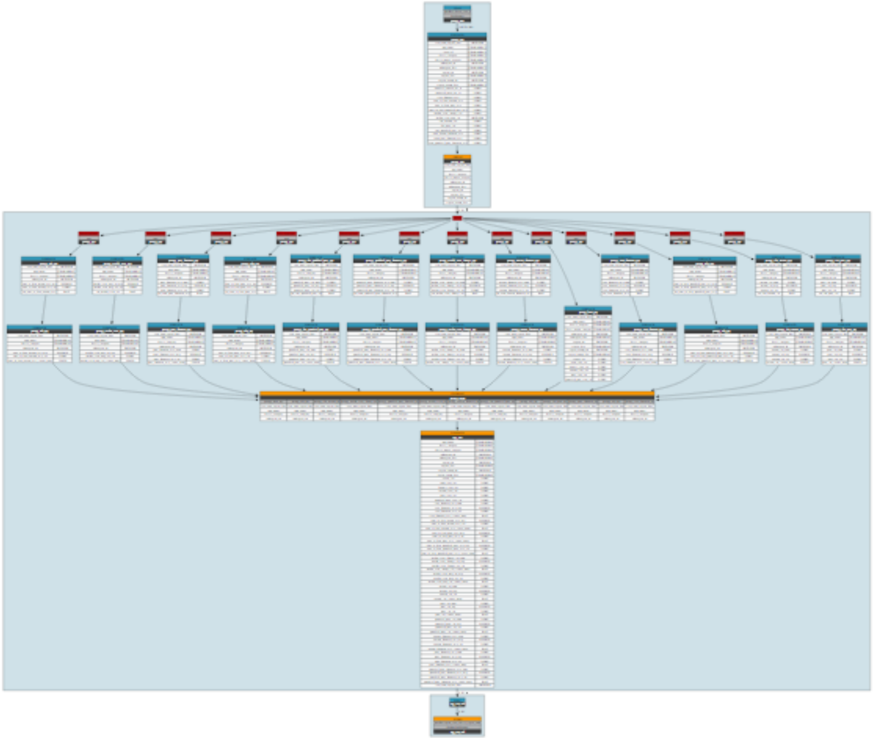


Ephemeral ports / --net=host mode

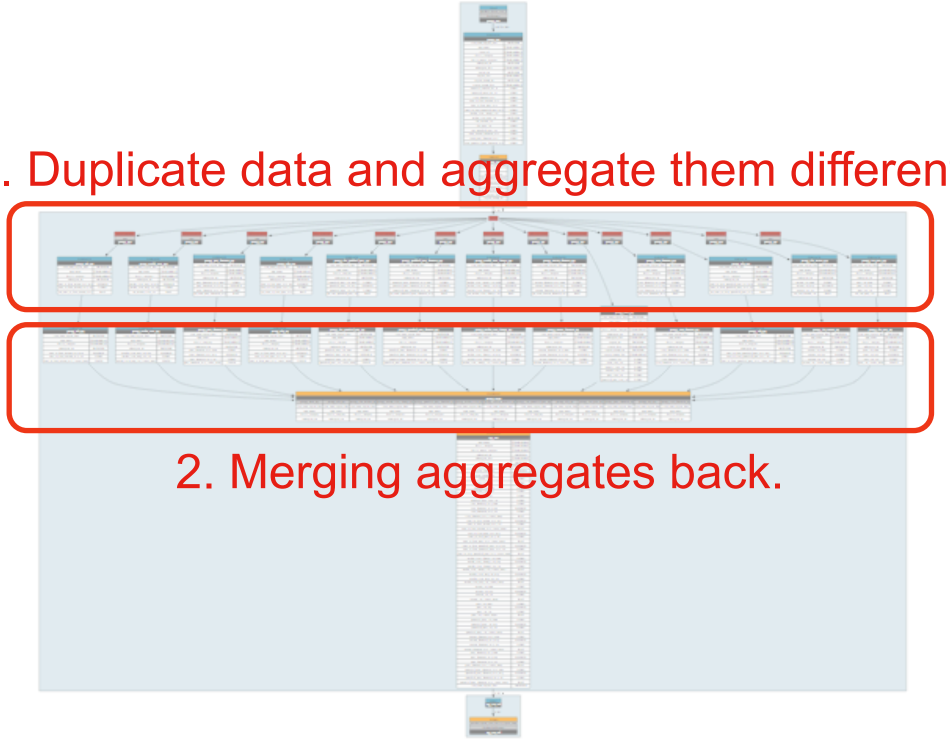


Use Case
Pig vs. Spark

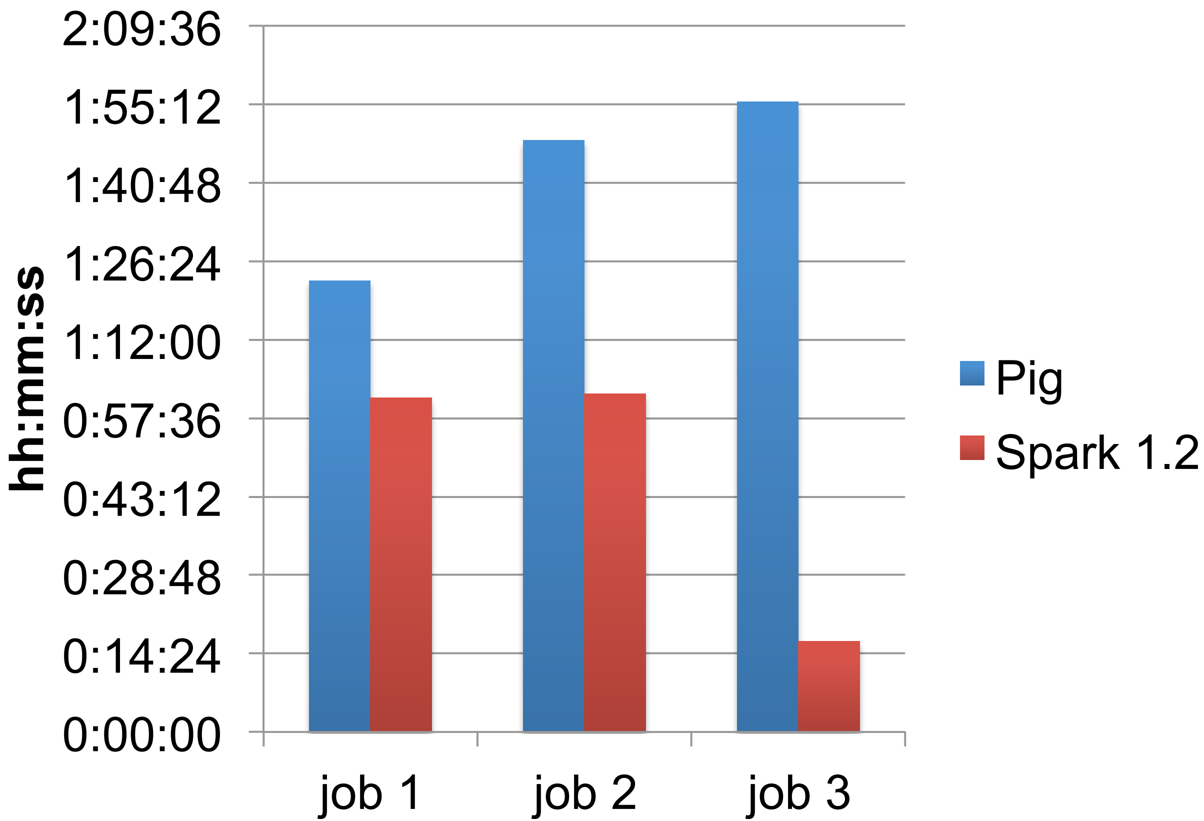
Iterative job



Iterative job



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