넷플릭스 빅데이터 플랫폼
아파치 스팍
통합 경험기

DEVIEW
2015

이름: 박철수
소속: 넷플릭스

NETFLIX

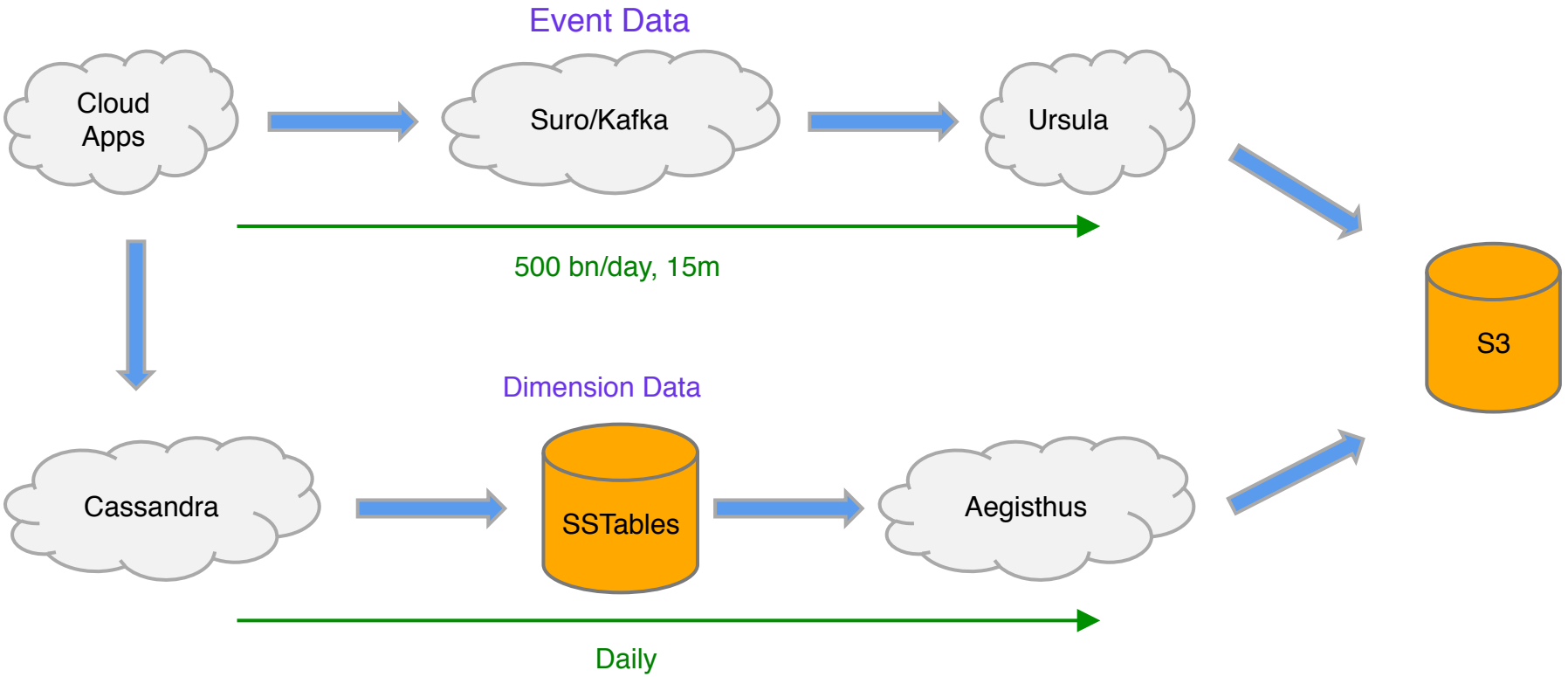
Outline

1. Netflix big data platform
2. Spark @ Netflix
3. Dynamic allocation
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

DEVIEW
2015

Netflix data pipeline



Netflix big data platform

DEVIEW 2015

Tools — **Big Data API/Portal** — 

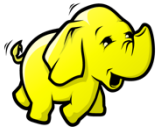
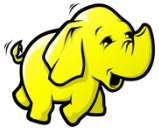
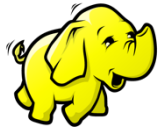
Service — 

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

DEVIEW
2015

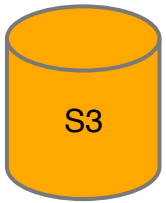
Mix of deployments

- 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



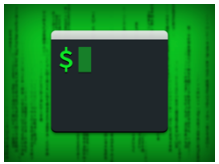
s3://bucket/spark/1.4/spark-1.4.tgz, spark-defaults.conf (spark.yarn.jar=1440304023)

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

/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

Goals

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

Dynamic Allocation

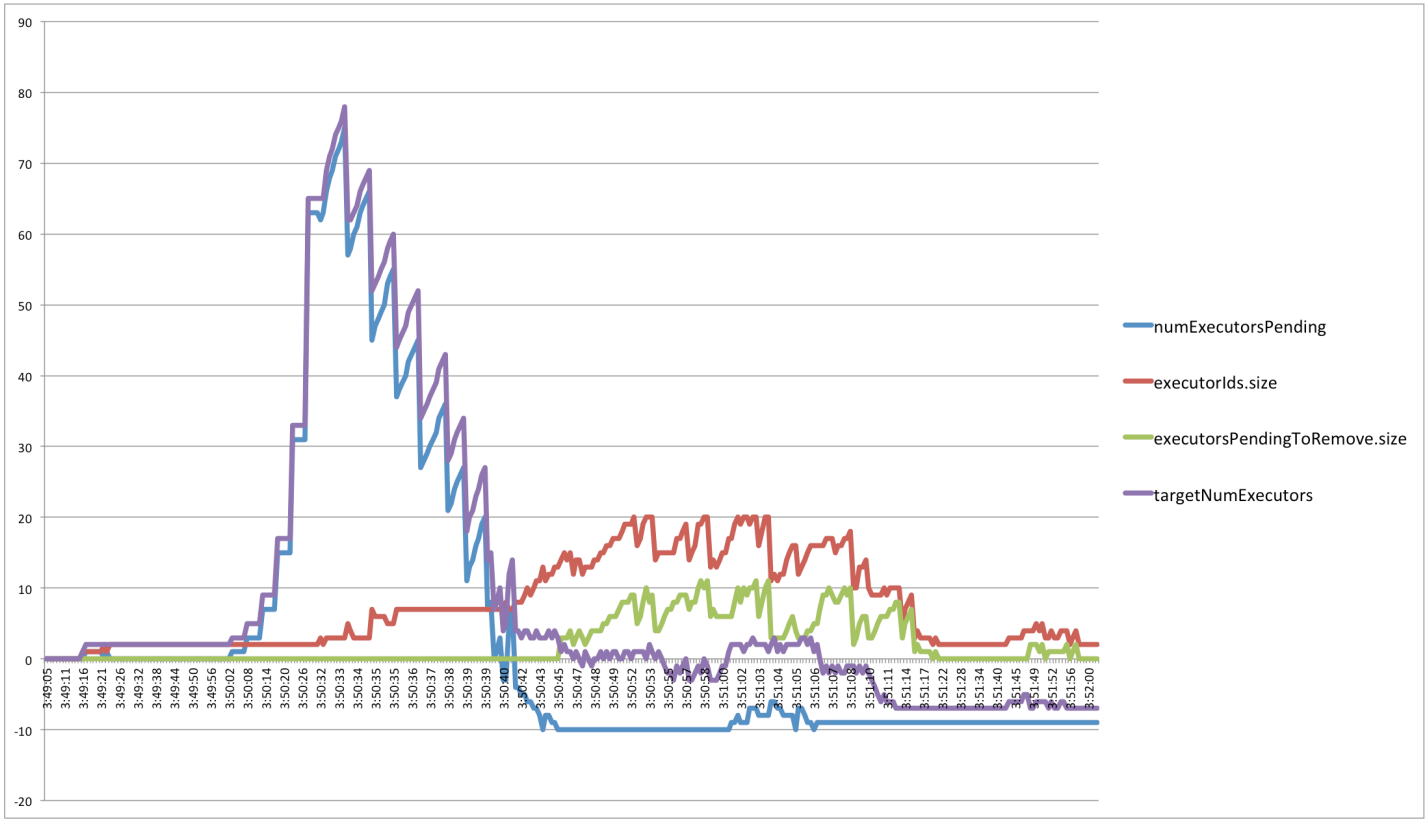
Dynamic allocation

DEVIEW 2015

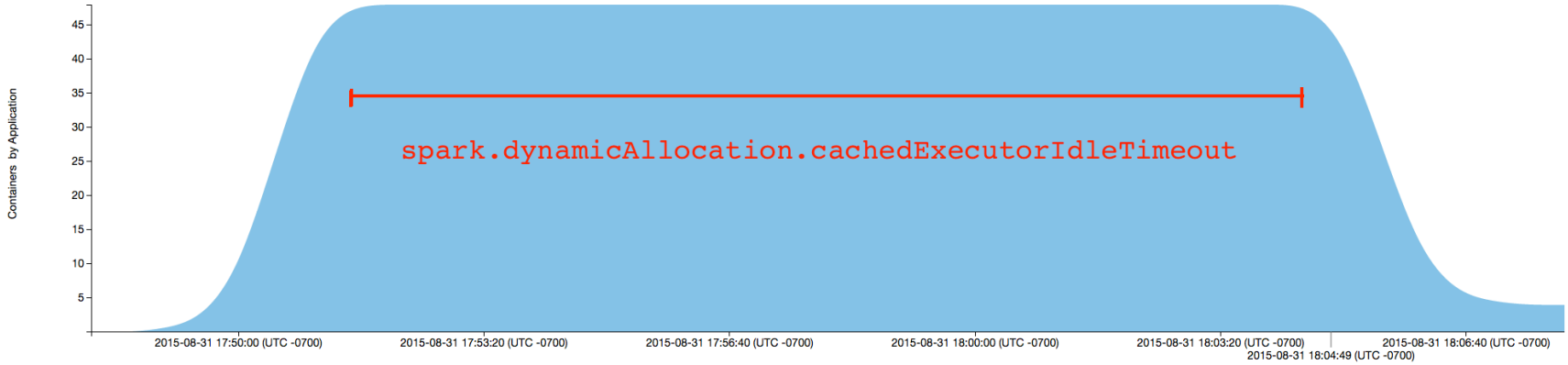
```
// spark-defaults.conf
spark.dynamicAllocation.enabled                          true
spark.dynamicAllocation.executorIdleTimeout              5
spark.dynamicAllocation.initialExecutors                 3
spark.dynamicAllocation.maxExecutors                     500
spark.dynamicAllocation.minExecutors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

SPARK-6954


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

SPARK-7955




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

SPARK-8167

- Symptom
  - Spark tasks randomly fail with “executor lost” error.
- Cause
  - YARN preemption is not graceful.
- Solution
  - Preempted tasks shouldn’t be counted as failures but should be retried.

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.

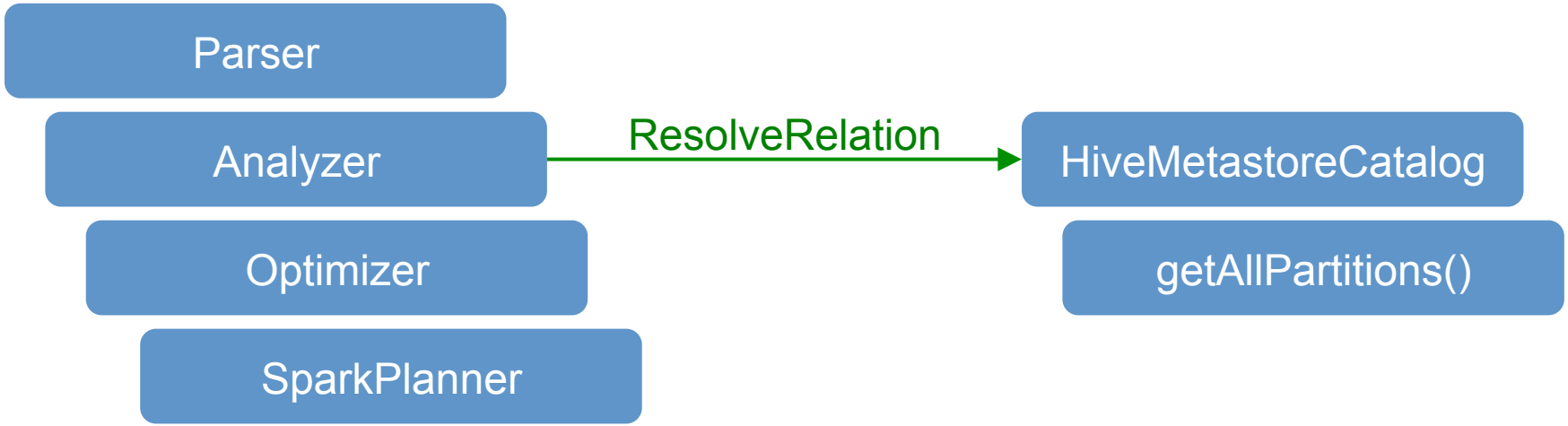
Predicate Pushdown

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

DEVIEW 2015

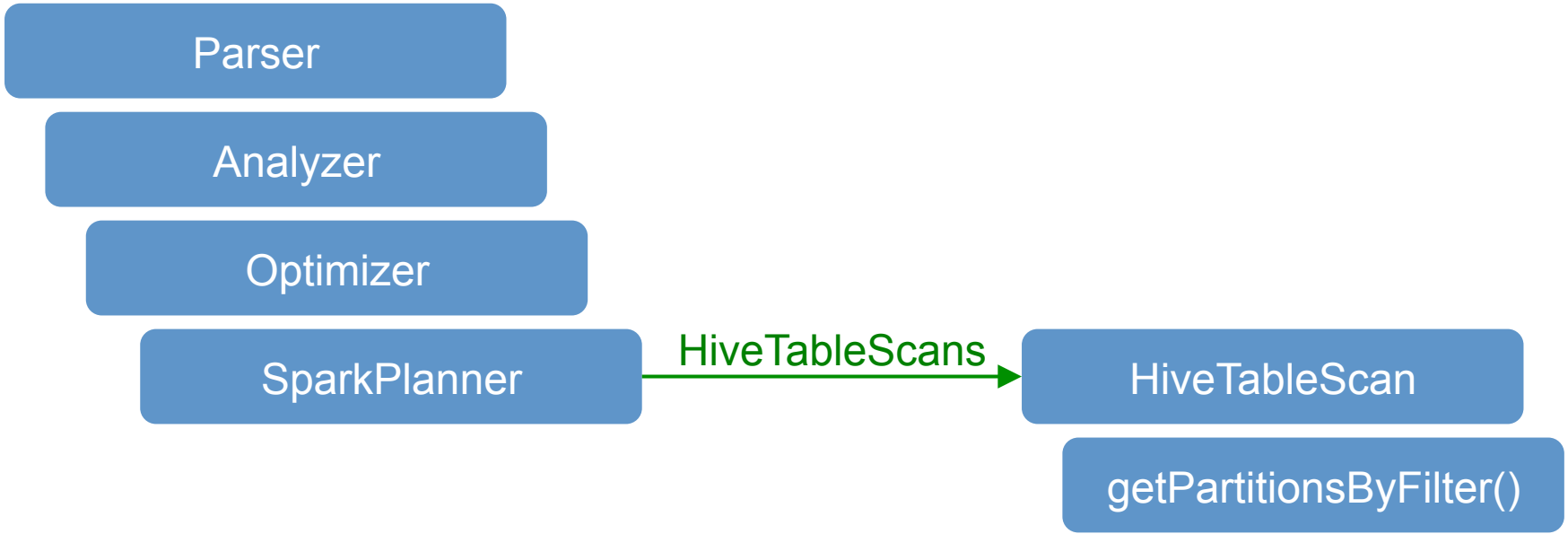


What if your table has 1.6M partitions?

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



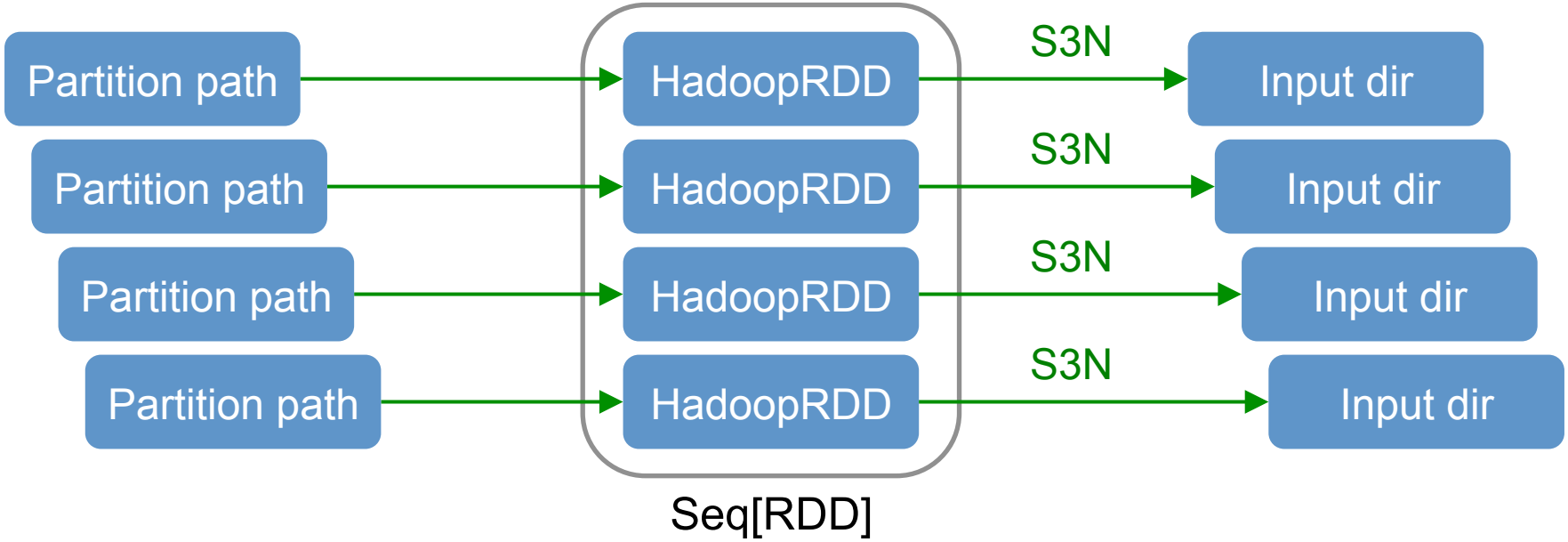
S3
File Listing

DEVIEW
2015

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

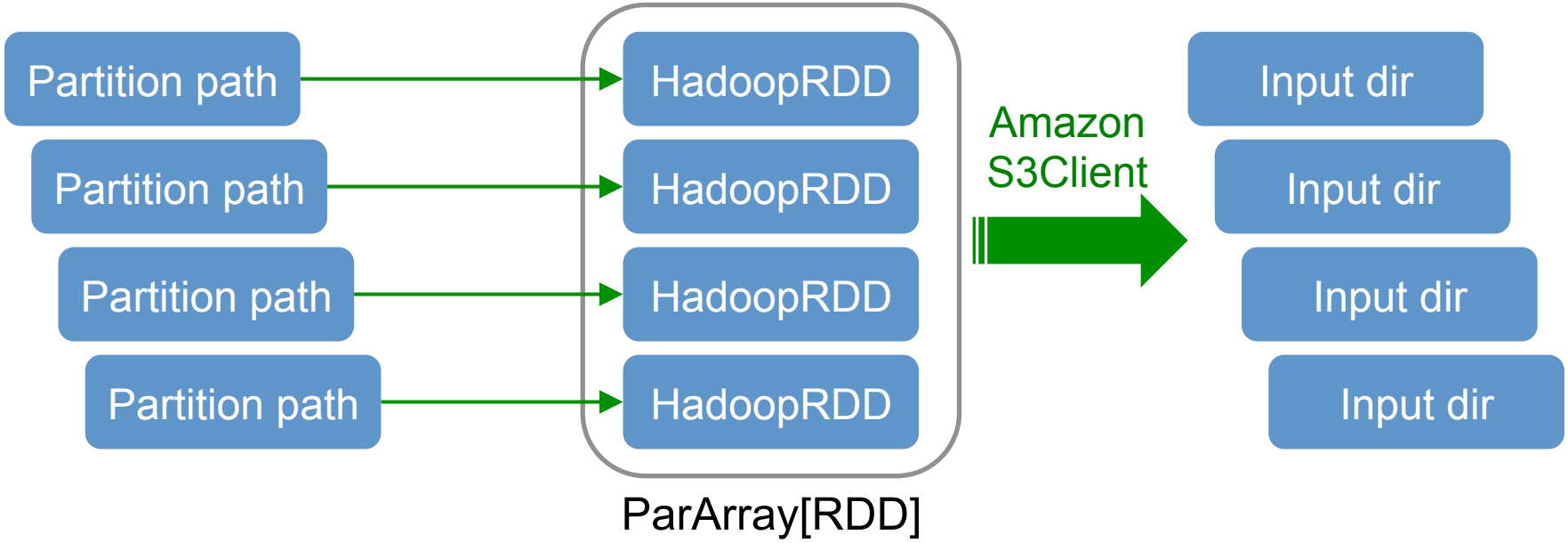


Sequentially listing input dirs via S3N file system.

SPARK-9926, SPARK-10340

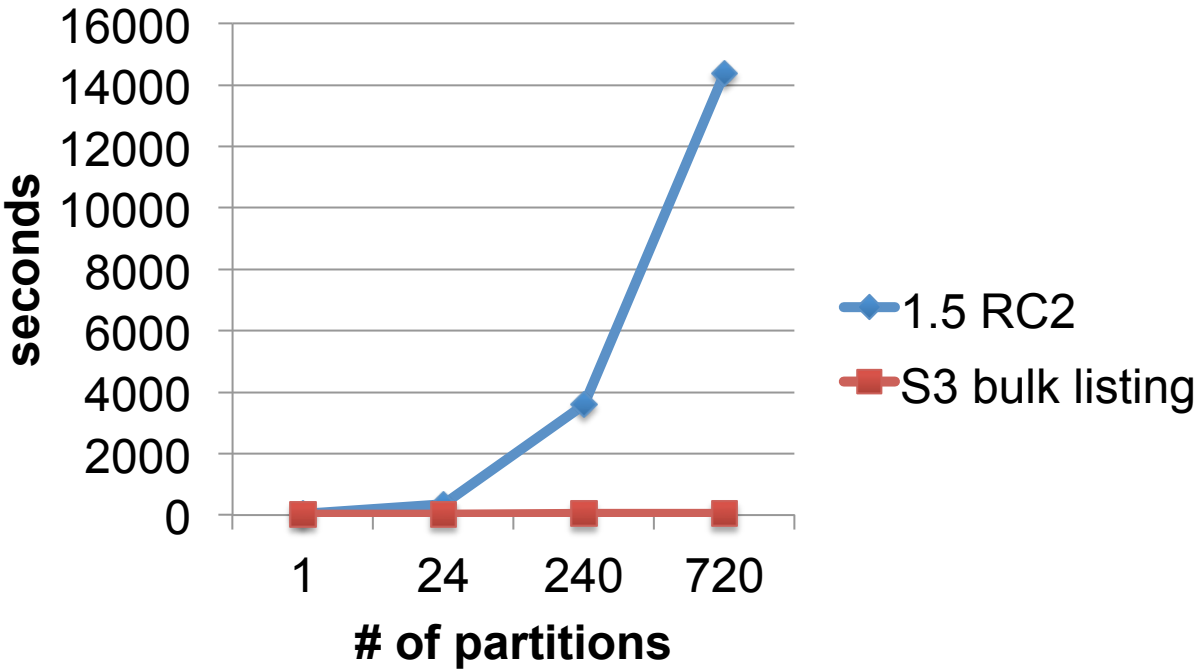
- Symptom
  - Input split computation for partitioned Hive table on S3 is slow.
- Cause
  - Listing files on a per partition basis is slow.
  - S3N file system computes data locality hints.
- Solution
  - Bulk list partitions in parallel using AmazonS3Client.
  - Bypass data locality computation for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

Hadoop output committer

- How it works:
  - Each task writes output to a temp dir.
  - Output committer renames first successful task's temp dir to fin al 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.

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

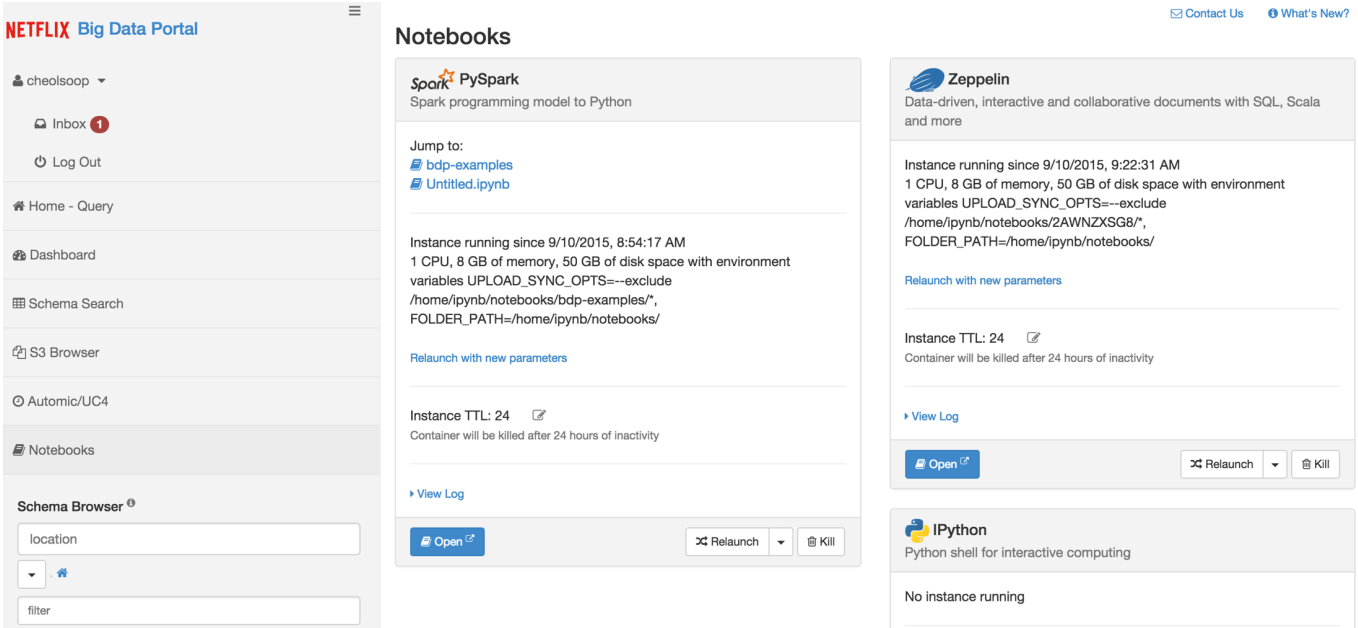
- How it works:
  - Never delete existing output in partitions.
  - Each job inserts a unique subpartition called “batchid.”
- Advantages:
  - Avoid eventual consistency.

Zeppelin Ipython Notebooks

DEVIEW
2015

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

DEVIEW 2015

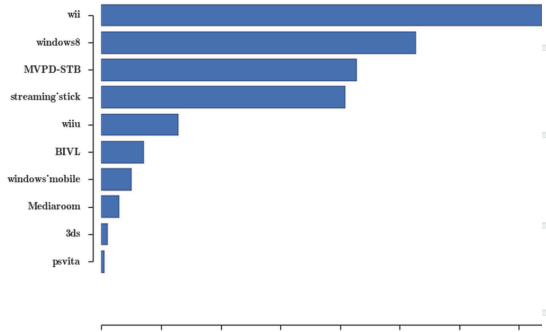
jupyter PySparkExamples (read only)

File Edit View Insert Cell Kernel Help | Python 2

Code | Cell Toolbar: None

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_style("ticks")
device, streams = zip(*top10)
ypos = np.arange(len(top10))
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_date=20150601
)
// 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
```

FINISHED

```
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)
```

FINISHED

```
// 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

FINISHED

```
// 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_d")
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)
```

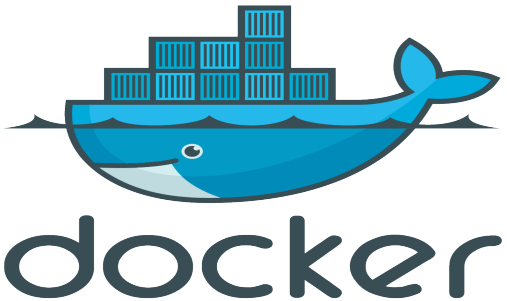
FINISHED

```
// The display still is ugly, let's use Zeppelin's markup to display this as a table:
print(s"""%table
country_iso_code\tshow_desc\tcontent_type\tgenre\tplay_hours\tnumber_profiles\tnumber_plays
""")
popularity_metadata.orderBy(popularity_metadata("SUM(play_hrs)").desc).take(10000).foreach(line => println(s"""""+line(0)+s"""\t"""+line(6)+s"""\t"""+line(17)+s"""\t"""+line(13)+s"""\t"""
```

FINISHED

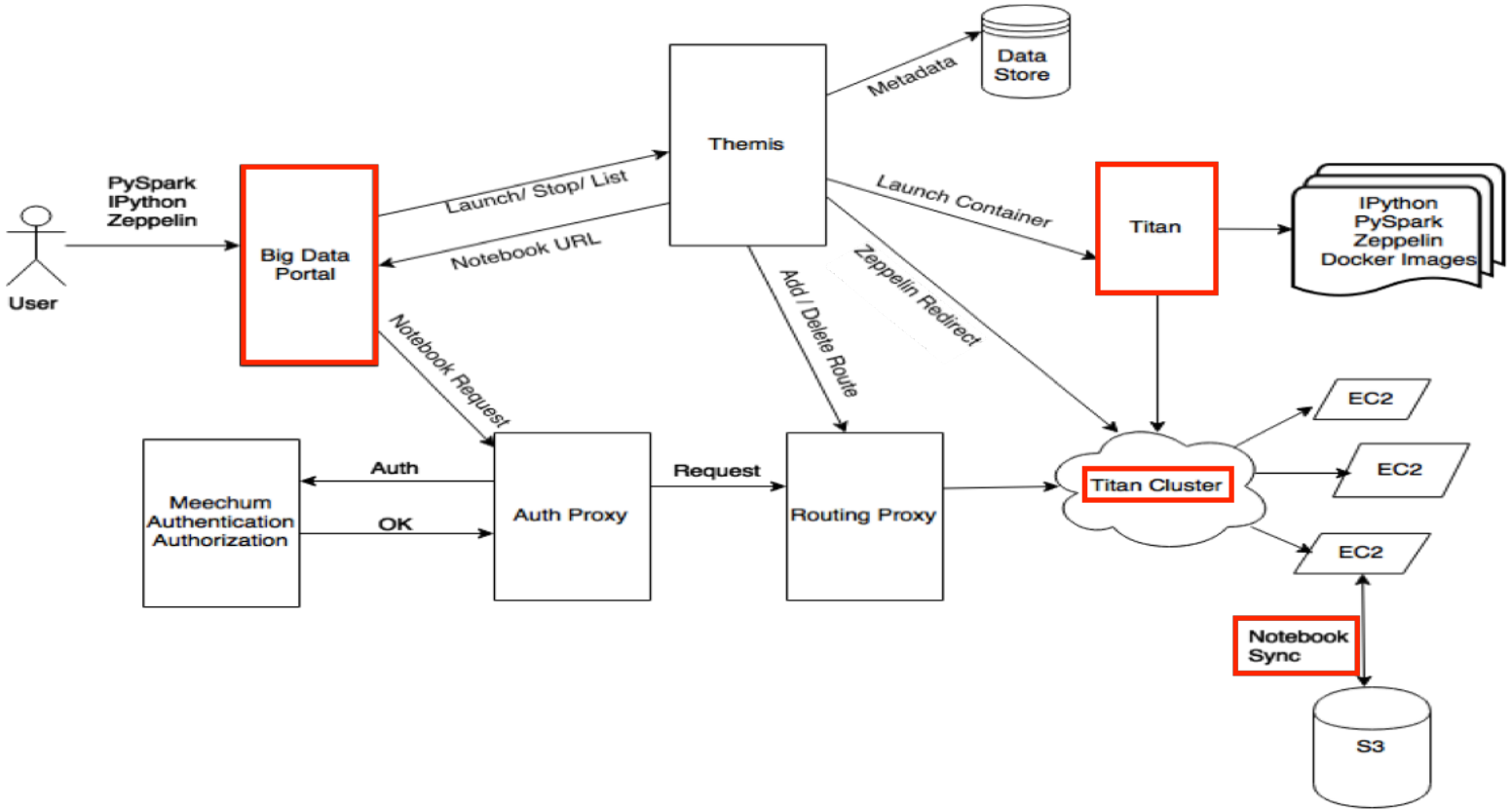
On demand notebooks

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



Notebook Infrastructure

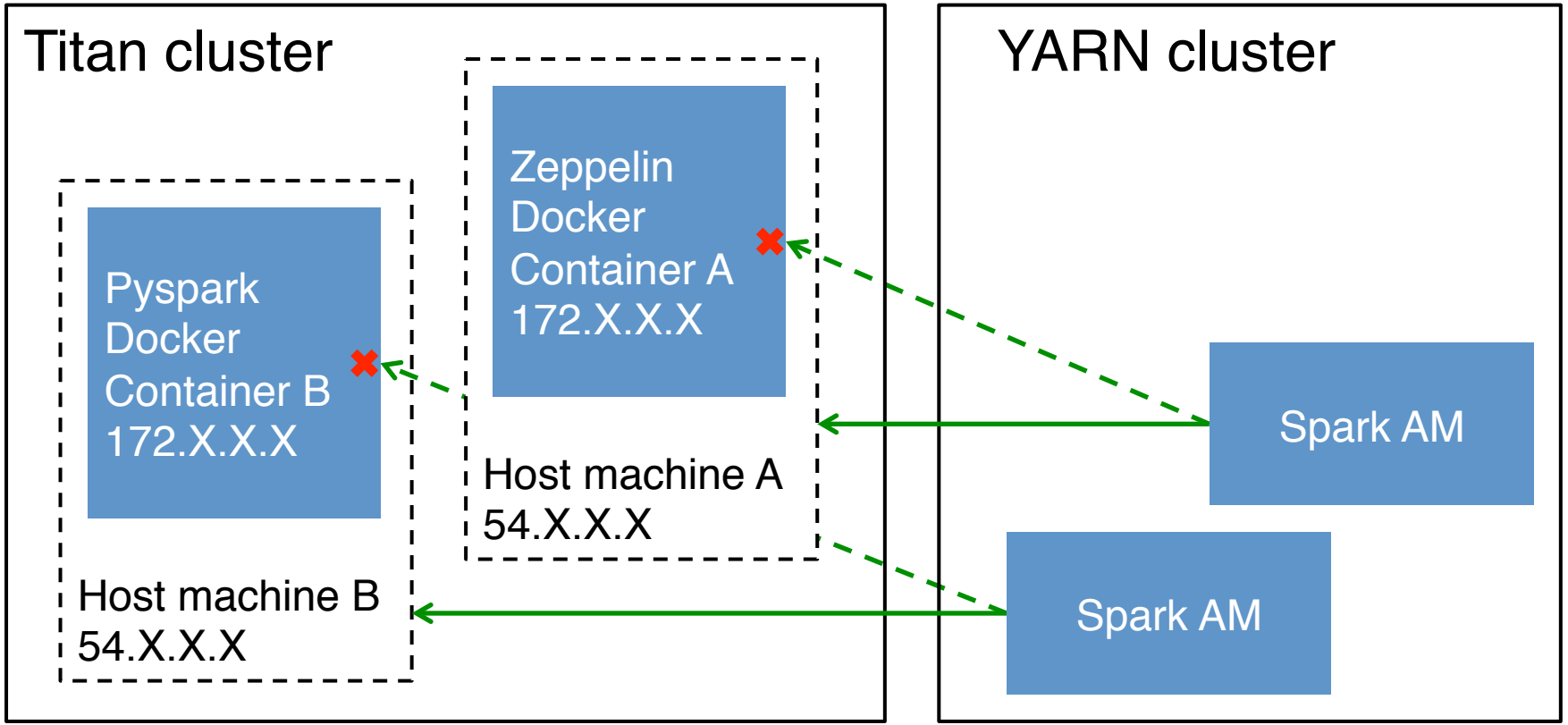
DEVIEW 2015



Quick facts about Titan

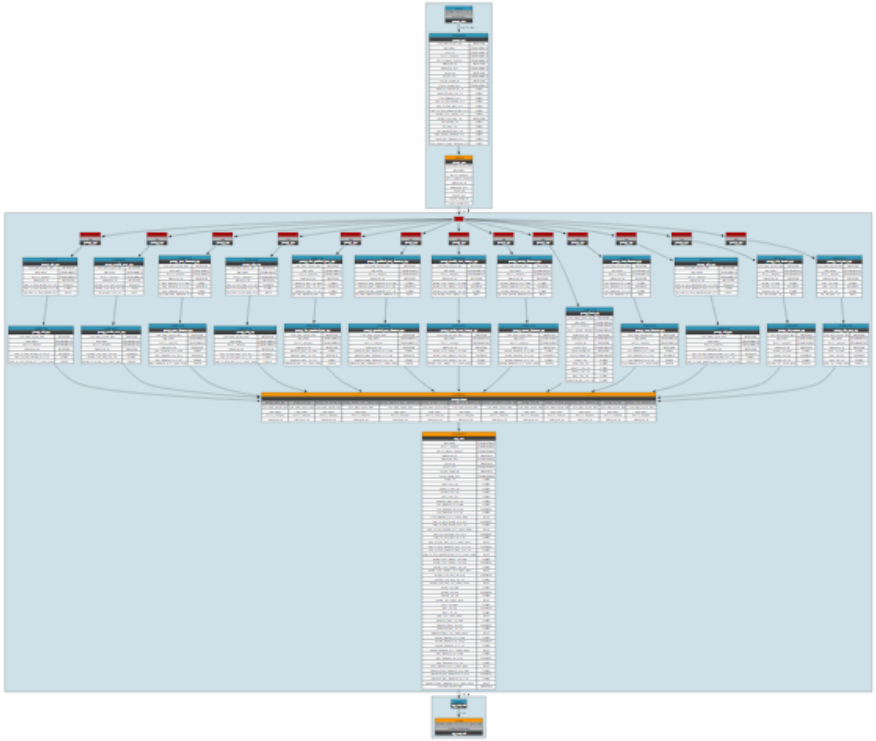
- Task execution platform leveraging Apache Mesos.
- Manages underlying EC2 instances.
- Process supervision and uptime in the face of failures.
- Auto scaling.

Ephemeral ports / --net=host mode

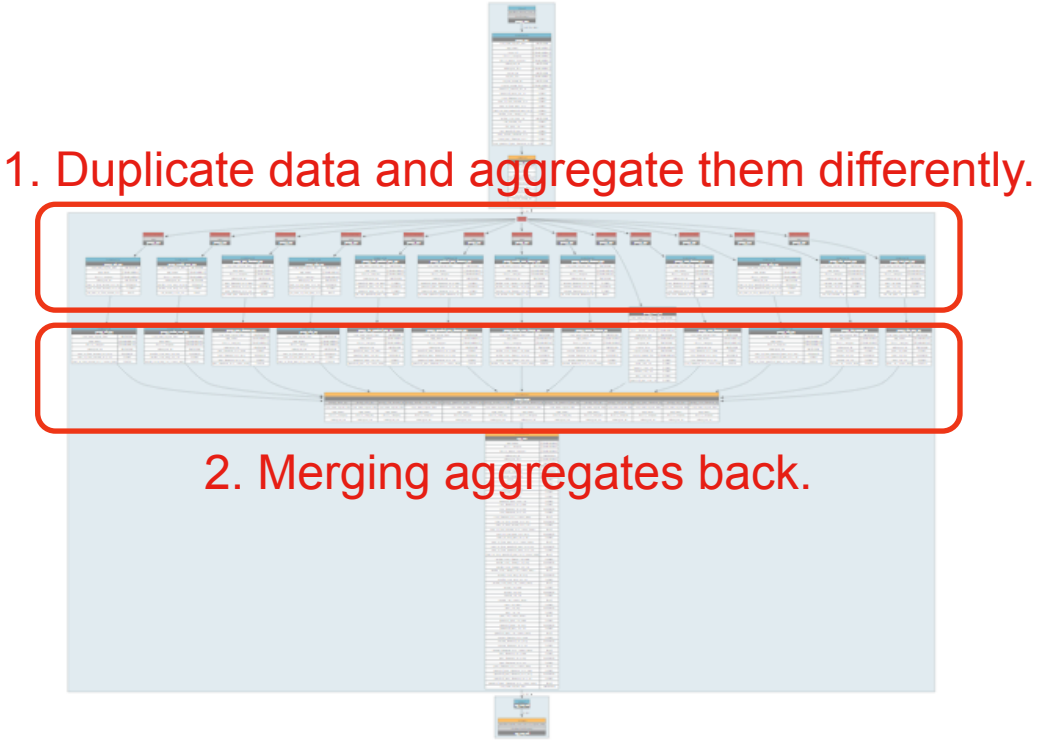


Use Case
Pig vs. Spark

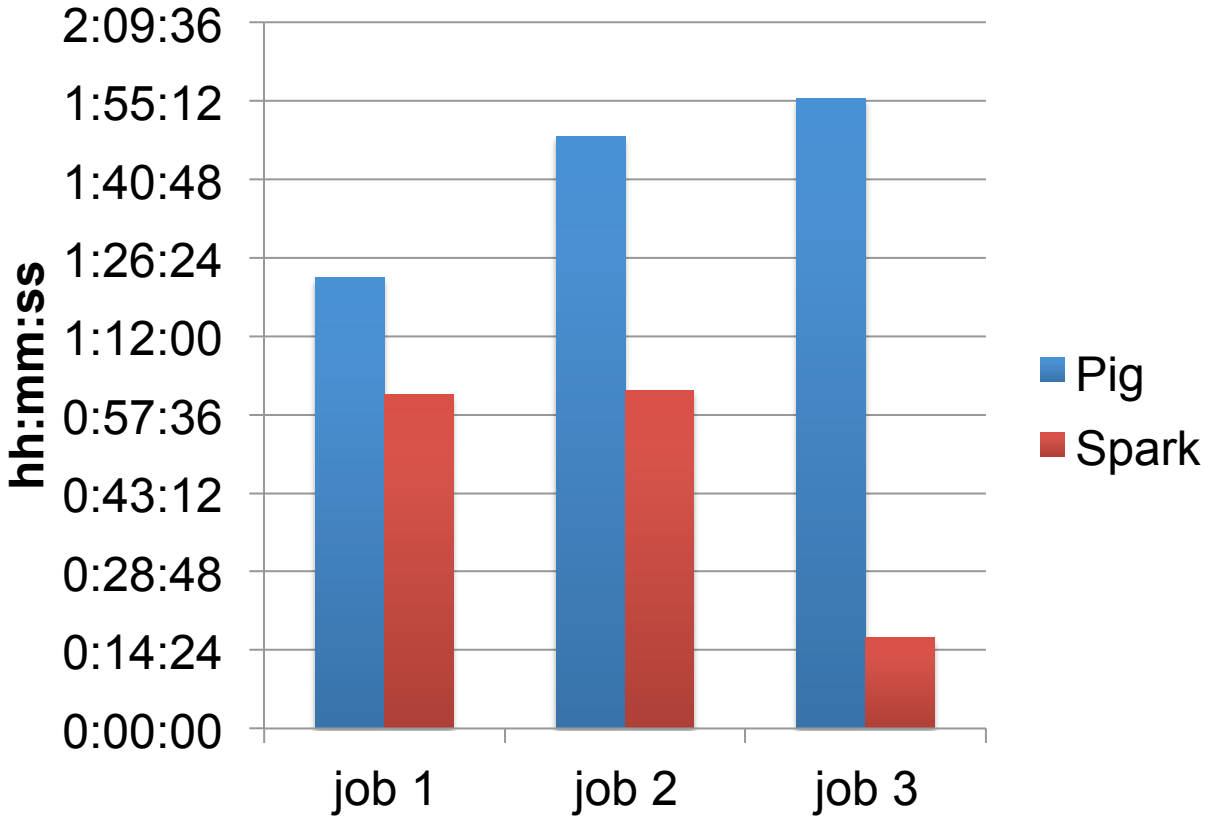
Iterative job



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

DEVIEW
2015

Thank You