AWS re:Invent

BDT303

Running Presto and Spark on the NETFLIX Big Data Platform

Eva Tse and Daniel Weeks, Netflix

October 2015

amazon web services

What to Expect from the Session

Our big data scale

Our architecture

For Presto and Spark

- Use cases
- Performance
- Contributions
- Deployment on Amazon EMR
- Integration with Netflix infrastructure


PLAN?
YES
NO
NETFLIX
RESULTS:
W.C.C
proceed to backup
STEPS: (TO STORAGE)
Payback

ignite

1234 Top Picks Test Case

Select Report

Retention & Streaming

Allocation Dates

07/06/2015 - 07/16/2015

Activity

Activity Window

35

Device

All

Is Original

All

Allocation

Completed Activity Window

All

Subregion

All

1234 Top Picks Test Case

Metrics current through 09/08/2015

Admin View | Present | Show Delta | Auto Apply

Report Type: retention | Custom Group: All | Activity Window: 35 | Is Original: All | Start Date: 07/06/2015 | End Date: 07/16/2015

Allocation Type: [illegible] | Device: All

Comparison Cell: 1 Merge Cells	1 - Control	2 - Secondary Control	3 - Aggressive	4 - Default	5 - Minimal	6 - [illegible]	7 - [illegible]
# of Allocations	527,278	527,166	263,962	263,518	263,723	263,667	263,648
% Accounts Completed Window	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Cumulative Retention	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Streaming Hours hide							
% Accounts with > 0 Hours	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
% Accounts with >= 1 Hour	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
% Accounts with >= 5 Hours	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

NETFLIX

HOW WE CALCULATE THE RANKINGS

Enter country

UNITED STATES

The Netflix ISP Speed Index is a measure of **prime time Netflix performance on particular ISPs** (internet service providers) around the globe, and not a measure of overall performance for other services/data that may travel across the specific ISP network.

ISP LEADERBOARD - AUGUST 2015

SHOW SMALLER ISPS

RANK	ISP	SPEED Mbps		PREVIOUS Mbps	RANK CHANGE	TYPE (Fiber / Cable / DSL / Satellite / Wireless)
1	Cox	3.67		3.62		Cable
2	Verizon - FiOS	3.64		3.54	+1	Fiber
3	Cablevision - Optimum	3.63		3.59	-1	Cable
4	Bright House	3.57		3.42	+2	Cable
5	Time Warner Cable	3.53		3.37	+3	Cable
6	Charter	3.52		3.46	-2	Cable
7	Comcast	3.51		3.45	-2	Cable
8	Suddenlink	3.46		3.42	-1	Cable
9	Mediacom	3.45		3.32		Cable
10	AT&T - U-verse	3.28		3.20		Fiber

NETFLIX Browse Kids DVD Search eva

Suspenseful TV Shows

Children & Family Movies

Social & Cultural Documentaries

Foreign Movies



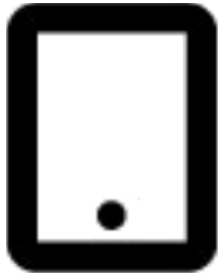


Our Biggest Challenge is Scale

Netflix Key Business Metrics

65+ million
members

50 countries



1000+ devices
supported

10 billion
hours / quarter

Our Big Data Scale

Total ~25 PB DW on Amazon S3

Read ~10% DW daily

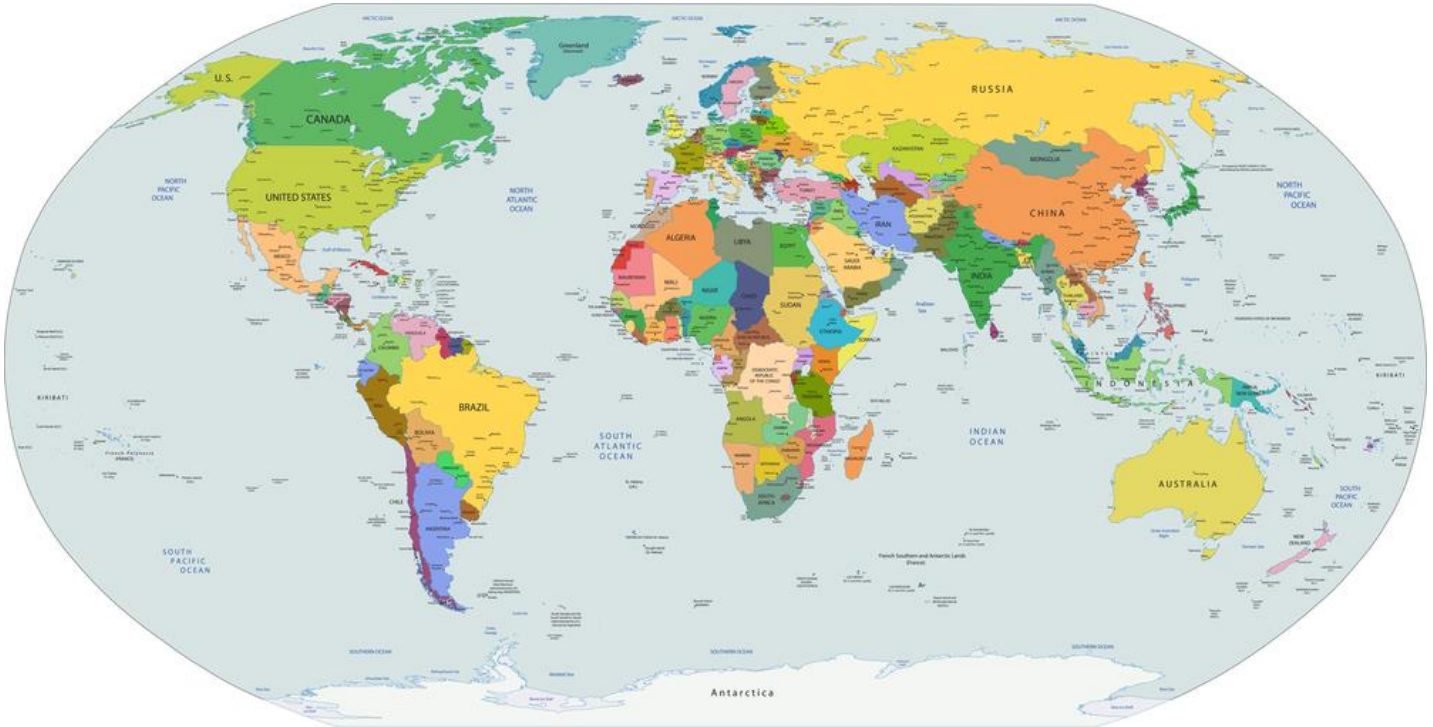
Write ~10% of read data daily

~ 550 billion events daily

~ 350 active platform users

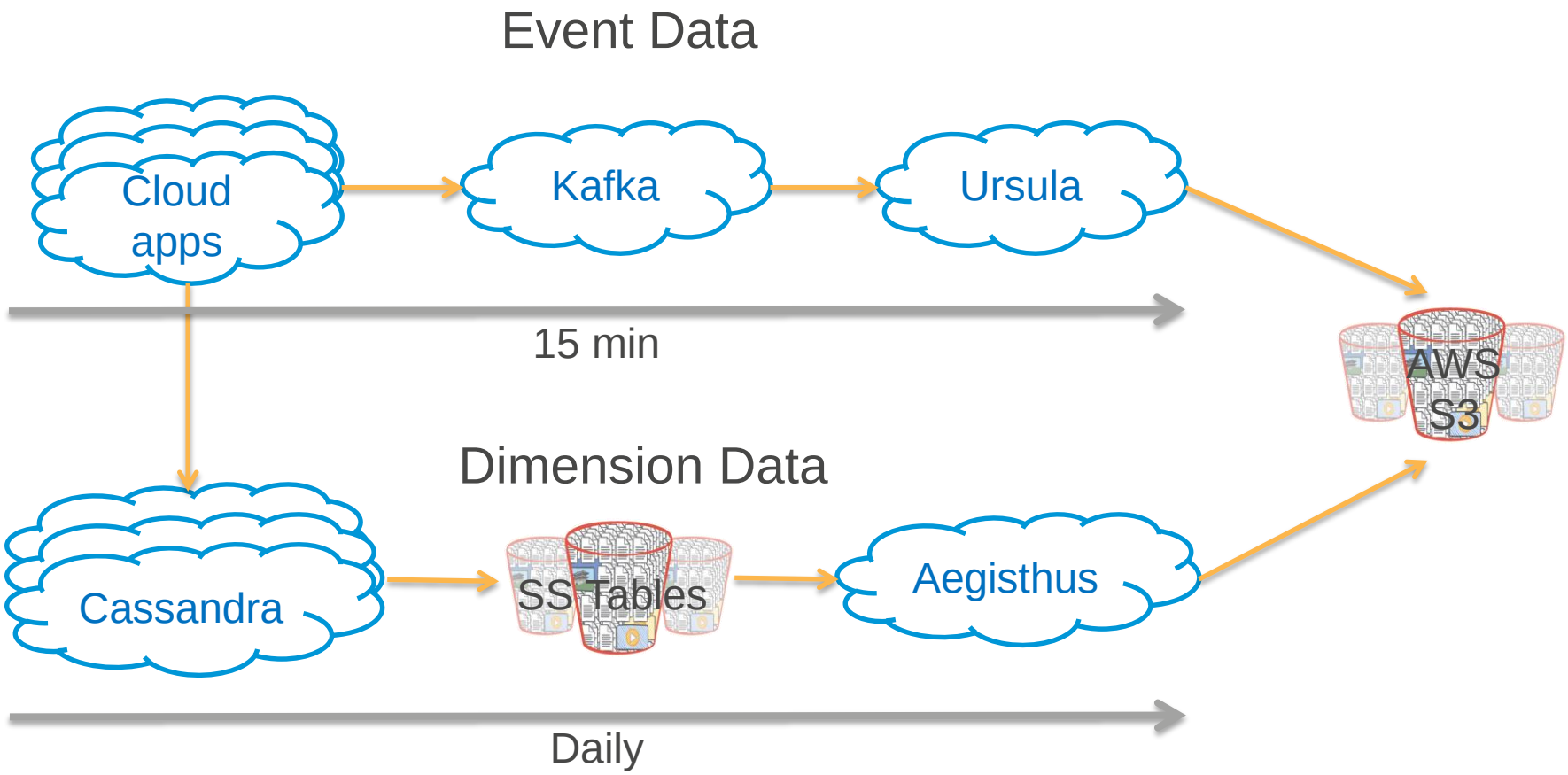
Global Expansion

200 countries by end of 2016



Architecture Overview

Data Pipelines



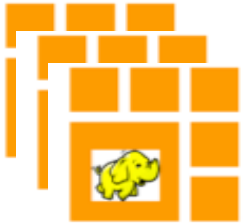
Storage

Compute

Service

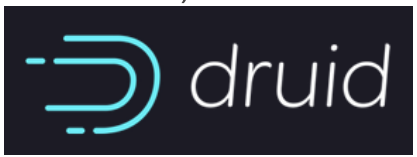
Tools

Parquet
FF



(Amazon
EMR
Hadoop
clusters)

(Federated
execution
service)

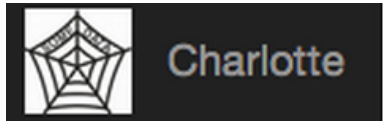


Metacat

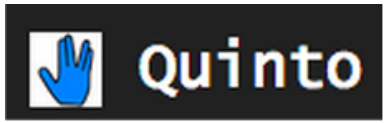
(Federated
metadata
service)

Data movement

Data visualization



Data lineage



Data quality

Pig workflow
visualization

Job/Cluster perf
visualization

Different Big Data Processing Needs

Analytics

ETL

Interactive data exploration

presto

Interactive slice & dice

druid

RT analytics & iterative/ML algo and more ...

Spark

Amazon S3 as our DW

Amazon S3 as Our DW Storage

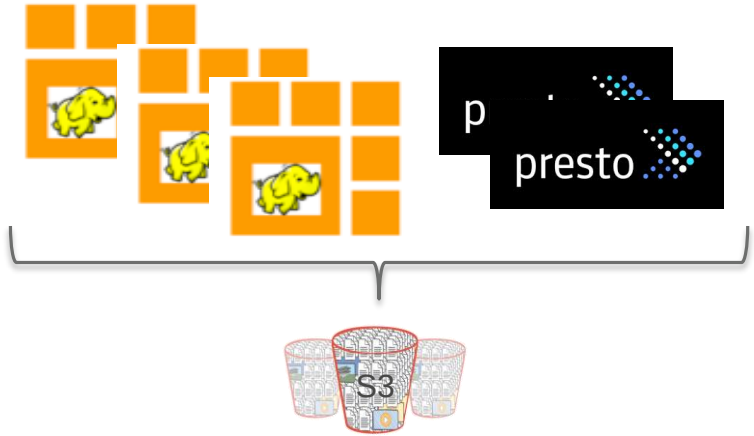
Amazon S3 as single source of truth (not HDFS)

Designed for 11 9's durability and 4 9's availability

Separate compute and storage

Key enablement to

- multiple heterogeneous clusters
- easy upgrade via r/b deployment



What About Performance?

Amazon S3 is a much bigger fleet than your cluster

Offload network load from cluster

Read performance

- Single stage read-only job has 5-10% impact
- Insignificant when amortized over a sequence of stages

Write performance

- Can be faster because Amazon S3 is eventually consistent w/ higher throughput

Presto

Presto is an **open-source, distributed SQL** query engine for running **interactive analytic** queries against data sources of all sizes ranging from gigabytes to **petabytes**

Why We Love Presto?

Hadoop friendly - integration with Hive metastore

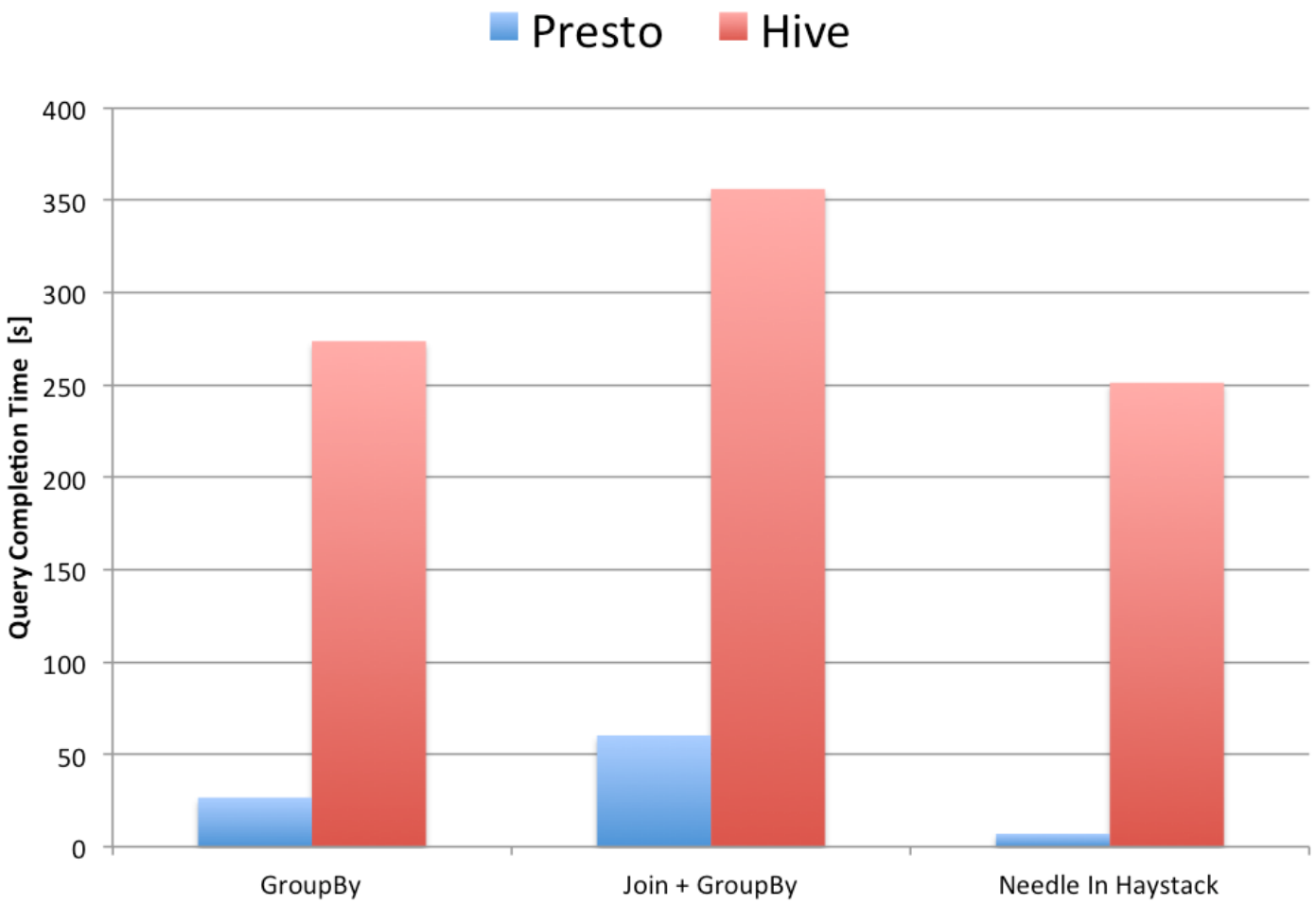
Works well on AWS - easy integration with Amazon S3

Scalable - works at petabyte scale

User friendly - ANSI SQL

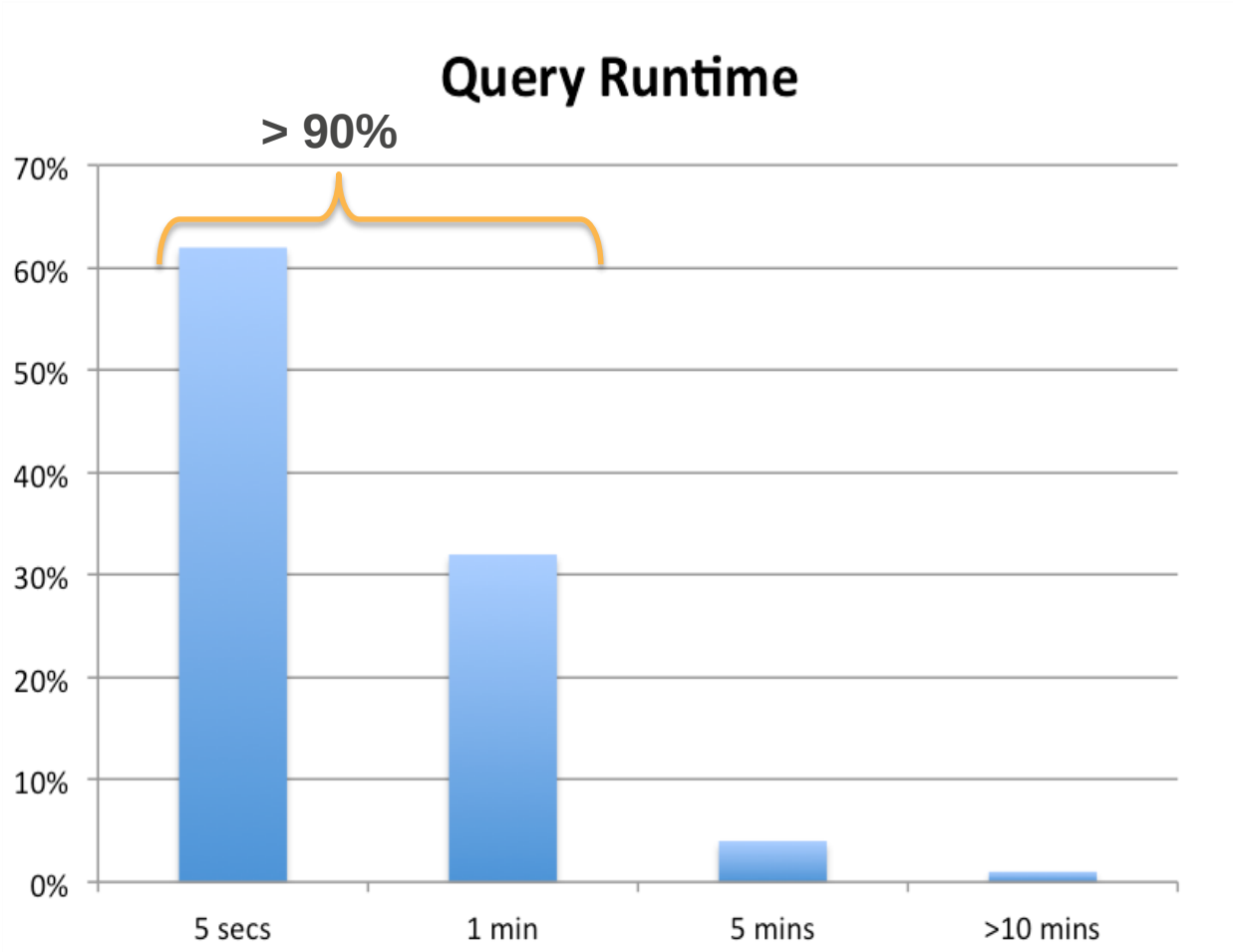
Open source - and in Java!

Fast


Presto
Hive
400
350
300
250
200
150
100
50
0
Query Completion Time [s]
GroupBy
Join + GroupBy
Needle In Haystack

Usage Stats

~3500 queries/day



Expanding Presto Use Cases

Data exploration and experimentation

Data validation

Backend for our interactive a/b test analytics application

Reporting

Not ETL (yet?)

Presto Deployment

Our Deployment

Version 0.114

+ patches
+ one non-public patch (Parquet vectorized read integration)

Deployed via bootstrap action on Amazon EMR

Separate clusters from our Hadoop YARN clusters

Not using Hadoop services

Leverage Amazon EMR for cluster management

Two Production Clusters

Resource isolation

Ad hoc cluster

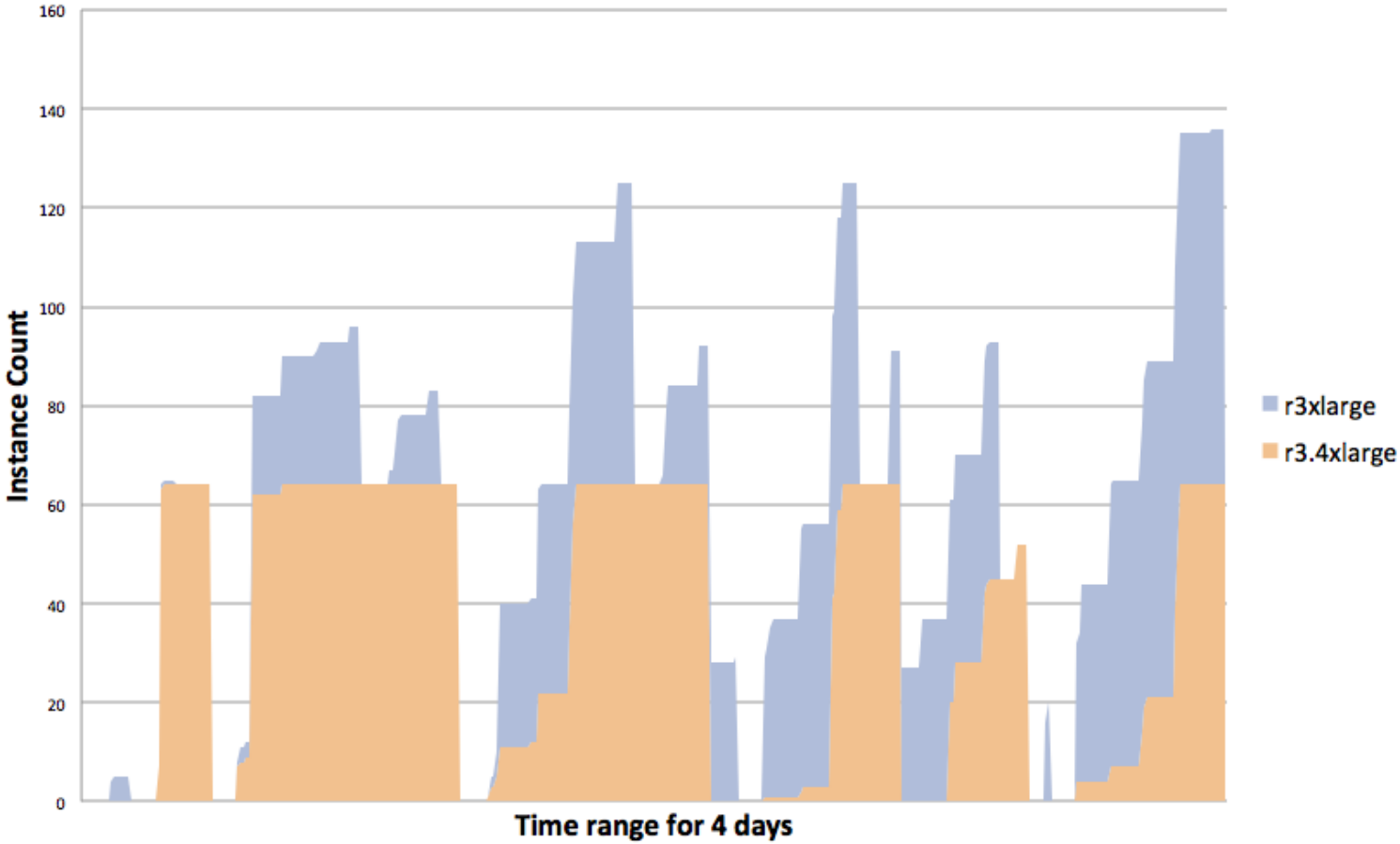
1 coordinator (r3.4xl) + 225 workers (r3.4xl)

Dedicated application cluster

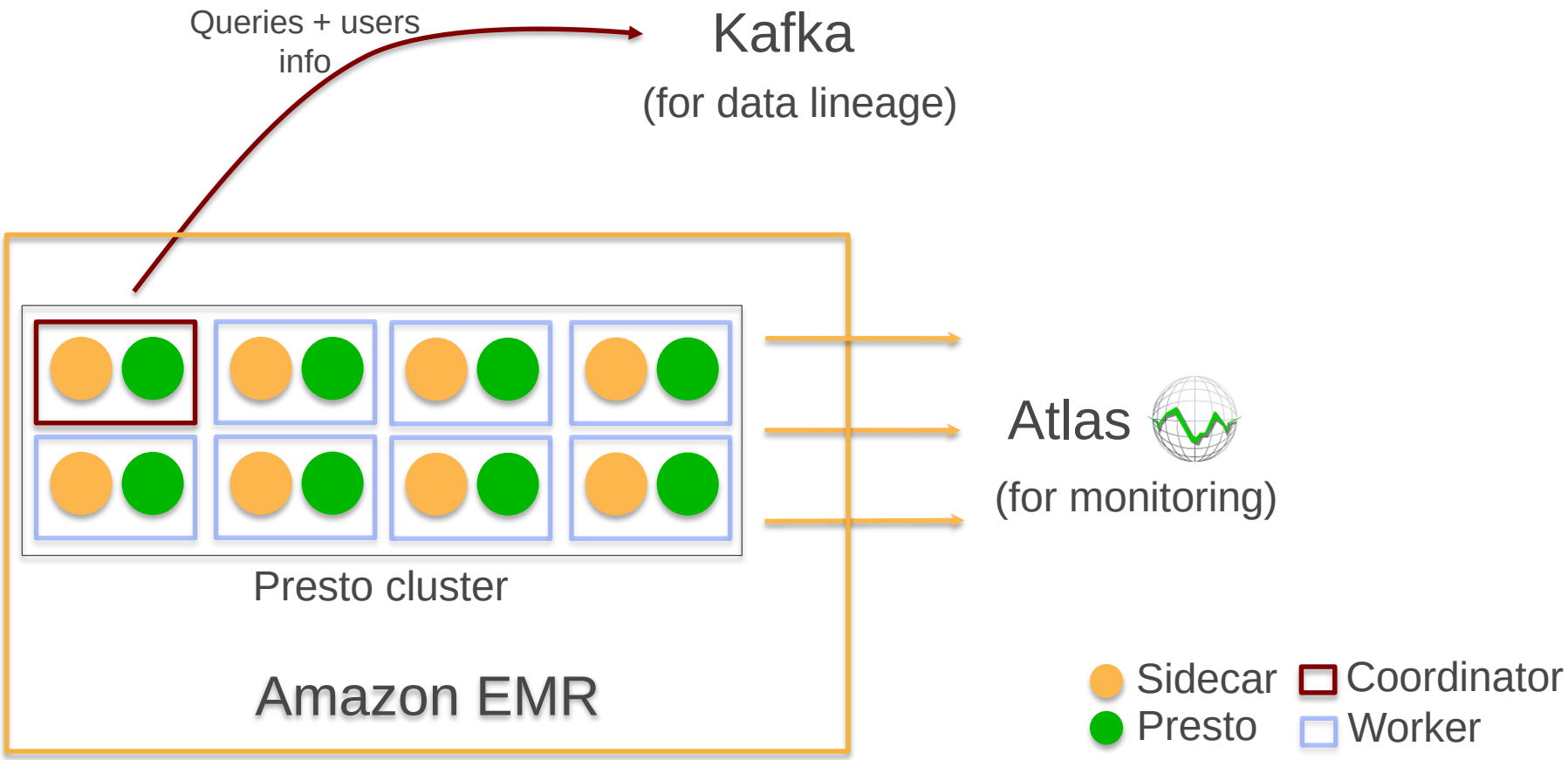
1 coordinator (r3.4xl) + 4 workers + dynamic workers (r3.xl, r3.2xl, r3.4xl)

Dynamic cluster sizing via Netflix spinnaker API

Dynamic Cluster Sizing



Integration with Netflix Infrastructure



Presto Contributions

Our Contributions

Amazon S3 File System

Multi-part upload

AWS instance credentials

AWS role support*

Reliability

Complex Types

Various functions for map, array and struct types

Comparison operators for array and struct types

Query Optimization

Single distinct -> group by

Joins with similar sub-queries*

Parquet File Format Support

Schema evolution

Predicate pushdown

Vectorized read**

Parquet

Columnar file format

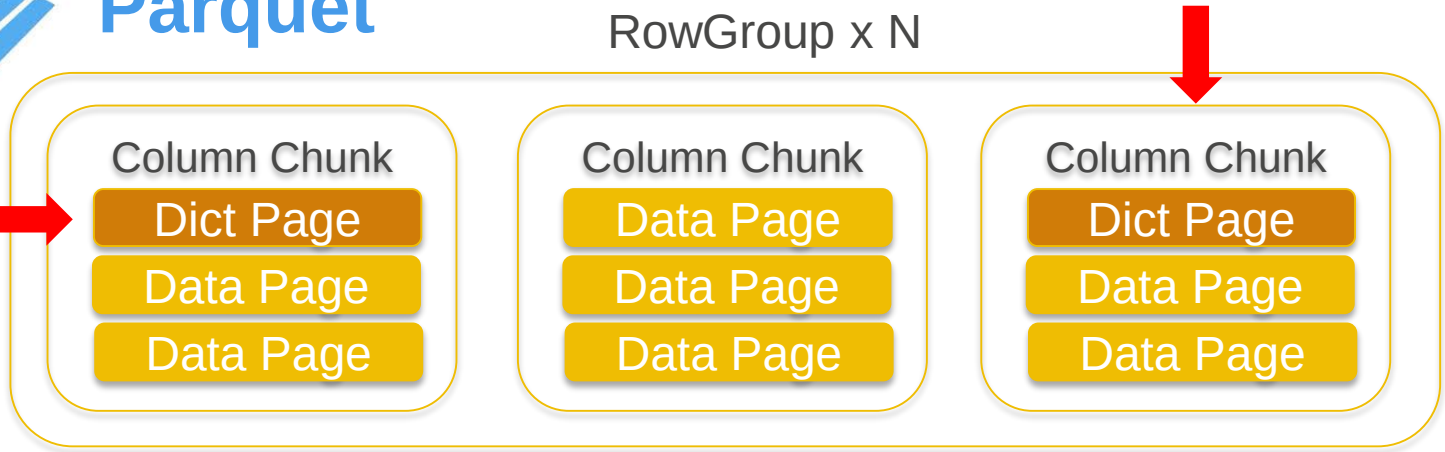
Supported across Hive, Pig, Presto, Spark

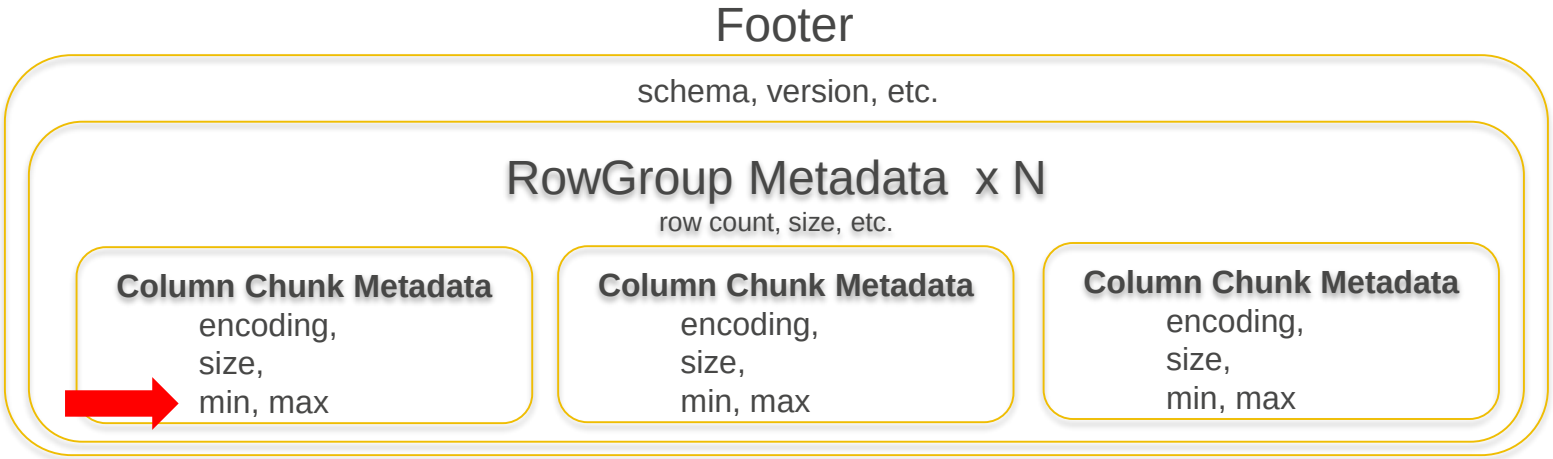
Performance benefits across different processing engines

Good performance on Amazon S3

Majority of our DW is on Parquet FF

Parquet


RowGroup x N
Column Chunk
Dict Page
Data Page
Data Page
Column Chunk
Data Page
Data Page
Data Page
Column Chunk
Dict Page
Data Page
Data Page


Footer
schema, version, etc.
RowGroup Metadata x N
row count, size, etc.
Column Chunk Metadata
encoding,
size,
min, max
Column Chunk Metadata
encoding,
size,
min, max
Column Chunk Metadata
encoding,
size,
min, max

Vectorized Read

Parquet: read column chunks in batches instead of row-by-row

Presto: replace ParquetHiveRecordCursor with ParquetPageSource

Performance improvement up to 2x for Presto

Beneficial for Spark, Hive, and Drill also

Pending parquet-131 commit before we can publish a Presto patch

Predicate Pushdown

Example: SELECT… WHERE abtest_id = 10;

Statistics pushdown

column chunk stats [20, 30]

skip this row group

Dictionary pushdown

column chunk stats [5, 20]

dictionary page <5,11,18,20>

skip this row group

Predicate Pushdown

Works best if data is clustered by predicate columns

Achieves data pruning like Hive partitions w/o the metadata overhead

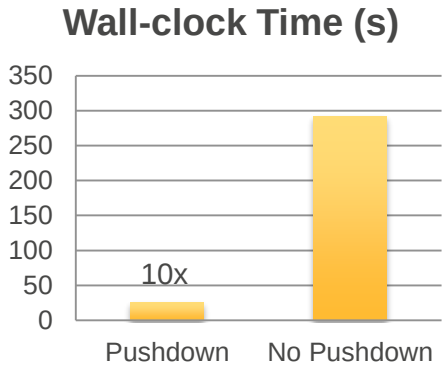
Can also be implemented in Spark, Hive, and Pig

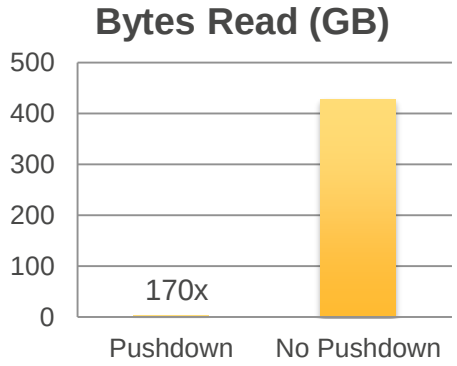
Atlas Analytics Use Case

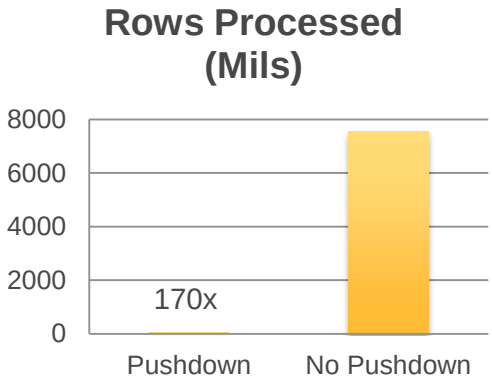
Example query: Analyze 4xx errors from Genie for a day

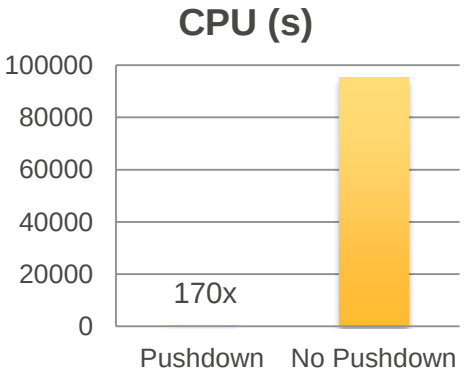
High cardinality/selectivity for app name and metrics name as predicates

Data staged and clustered by predicate columns









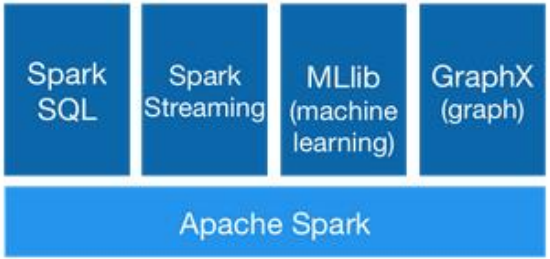
Stay Tuned…

Two upcoming blog posts on techblog.netflix.com

- Parquet usage @ Netflix Big Data Platform
- Presto + Parquet optimization and performance

Spark

Apache Spark™ is a fast and general engine for large-scale data processing.



Why Spark?

Batch jobs (Pig, Hive)

- ETL jobs
- Reporting and other analysis

Interactive jobs (Presto)

Iterative ML jobs (Spark)

Programmatic use cases

Deployments @ Netflix

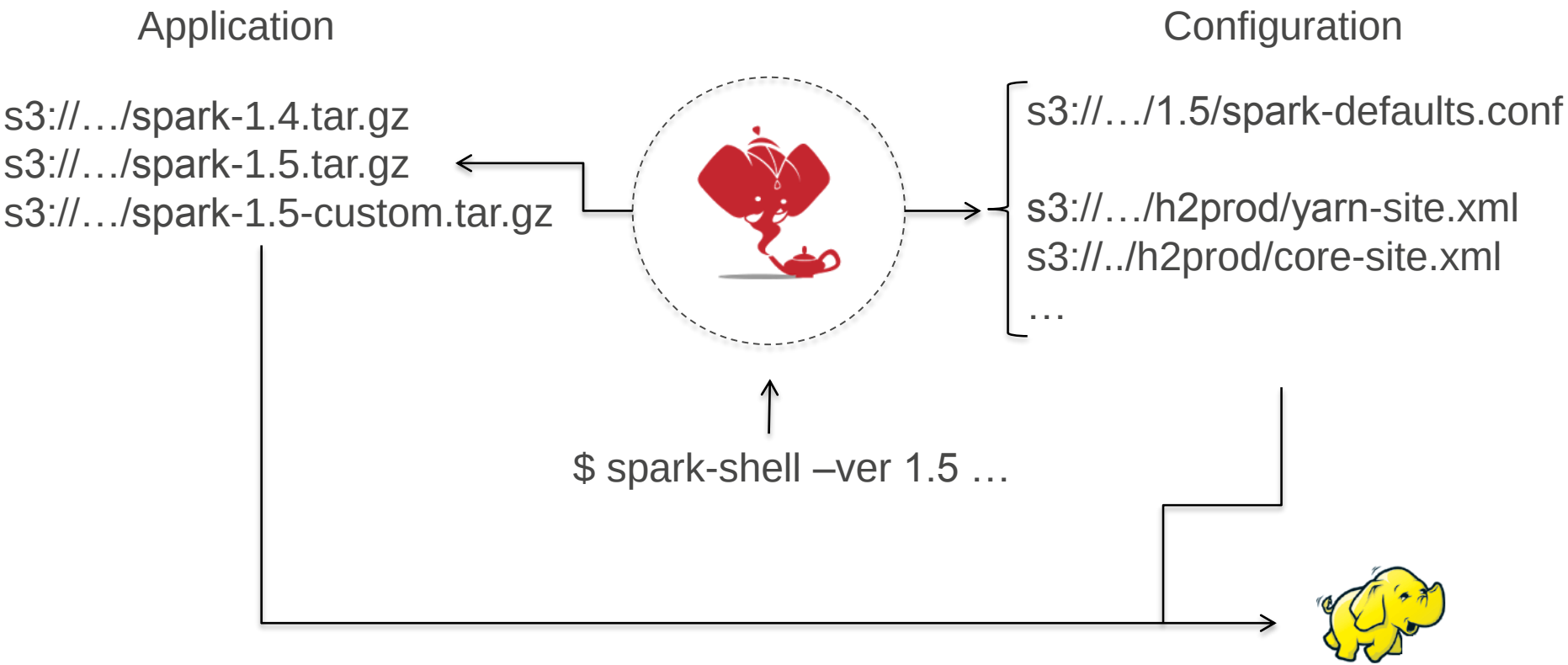
Spark on Mesos

- Self-serving AMI
- Full BDAS (**B**erkeley **D**ata **A**nalytics **S**tack)
- Online streaming analytics

Spark on YARN

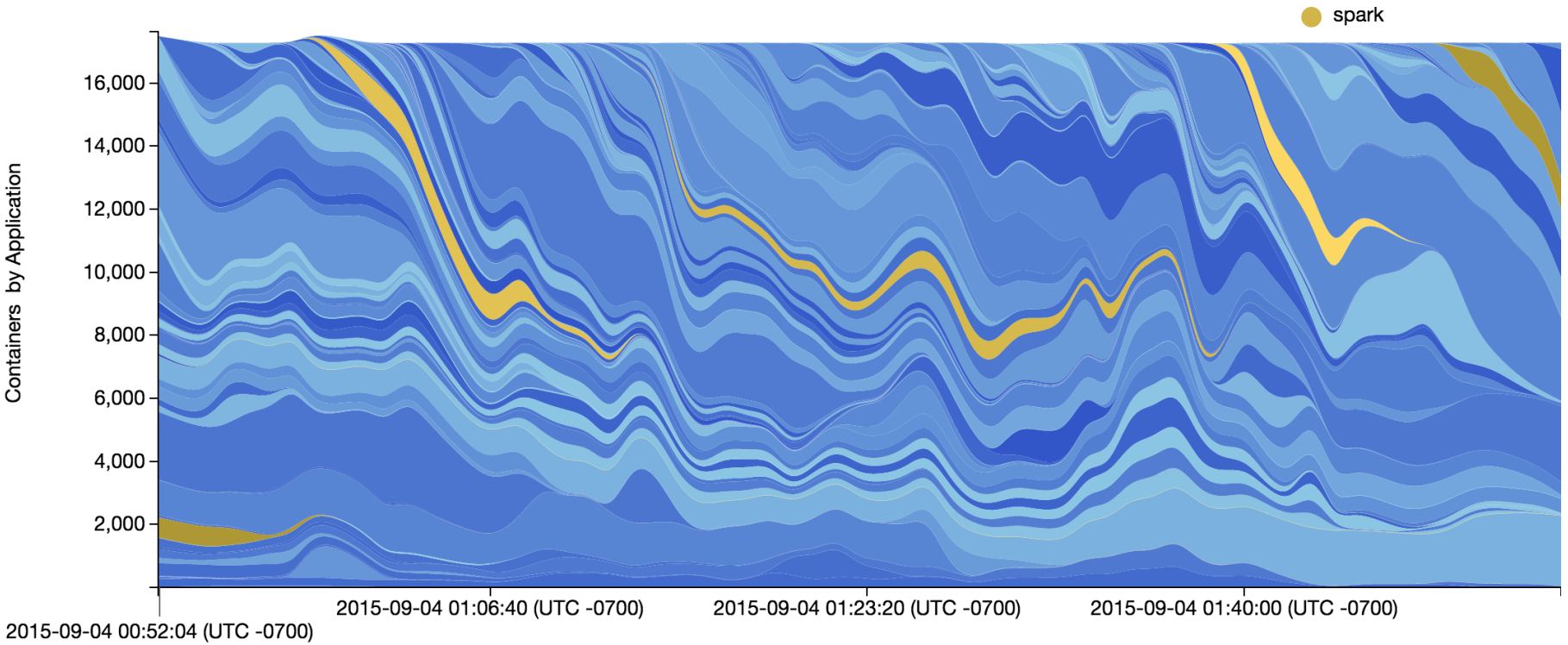
- Spark as a service
- YARN application on Amazon EMR Hadoop
- Offline batch analytics

Version Support



Multi-tenancy

Multi-tenancy



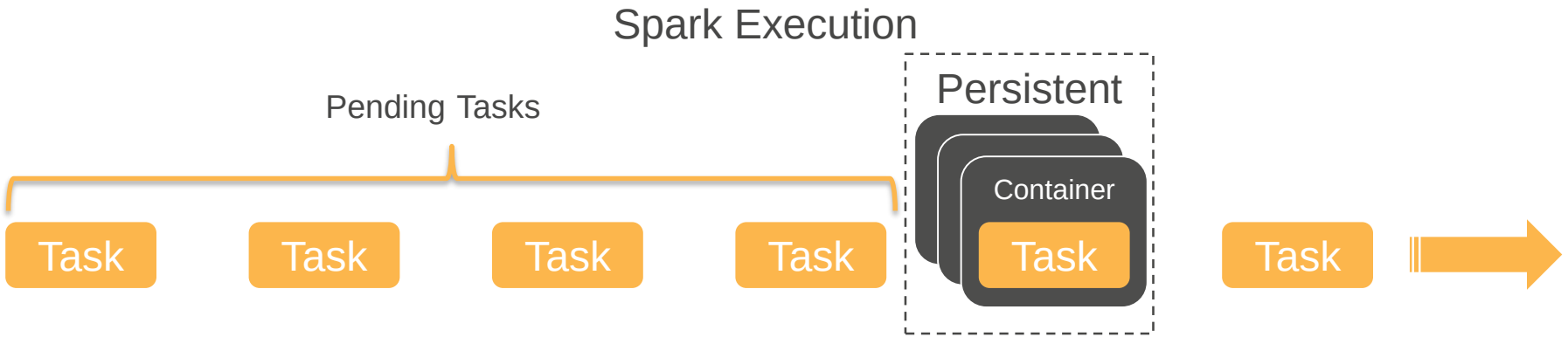
Dynamic Allocation on YARN

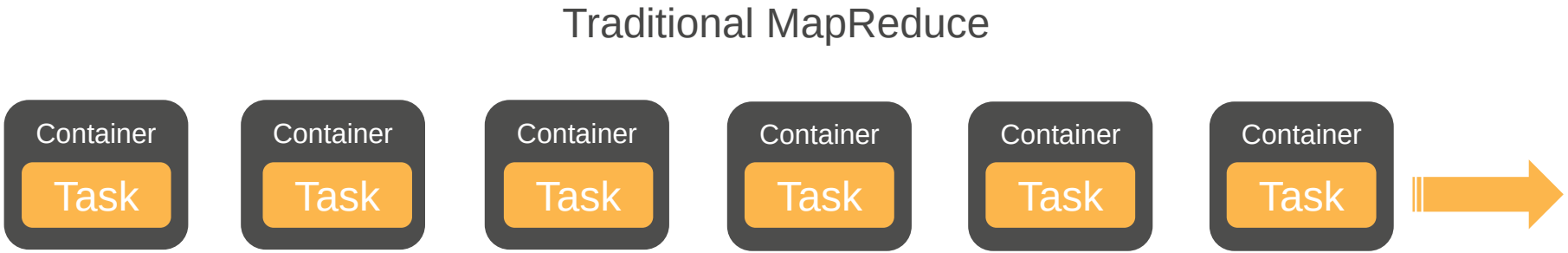
Optimize for resource utilization

Harness cluster’s scale

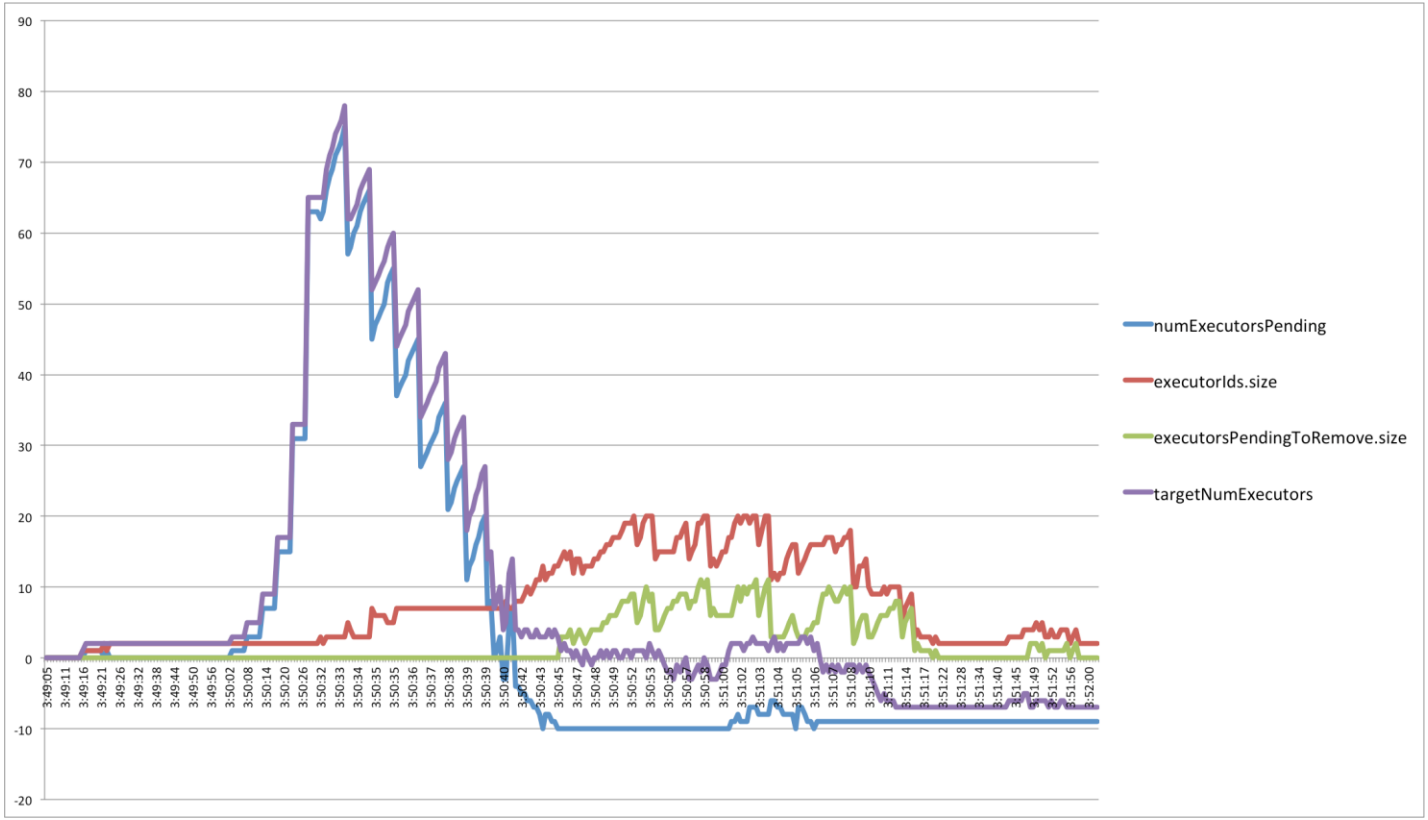
Still provide interactive performance

Task Execution Model





Dynamic Allocation [SPARK-6954]



Cached Data

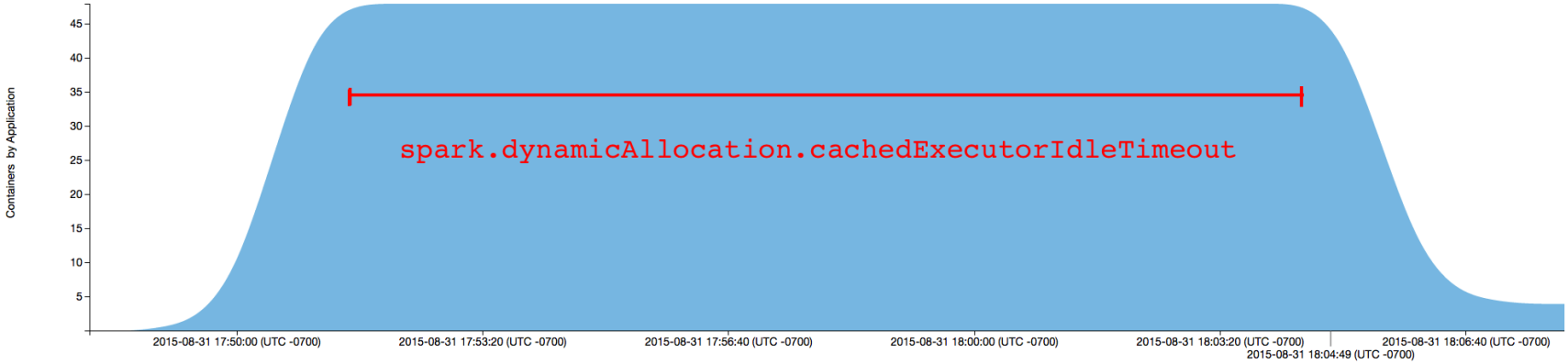
Spark allows data to be cached

- Interactive reuse of data set
- Iterative usage (ML)

Dynamic allocation

- Removes executors when no tasks are pending

Cached Executor Timeout [SPARK-7955]




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

Preemption [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

Reading / Processing / Writing

Amazon S3 Listing Optimization

Problem: Metadata is big data

- Tables with millions of partitions
- Partitions with hundreds of files each

Clients take a long time to launch jobs

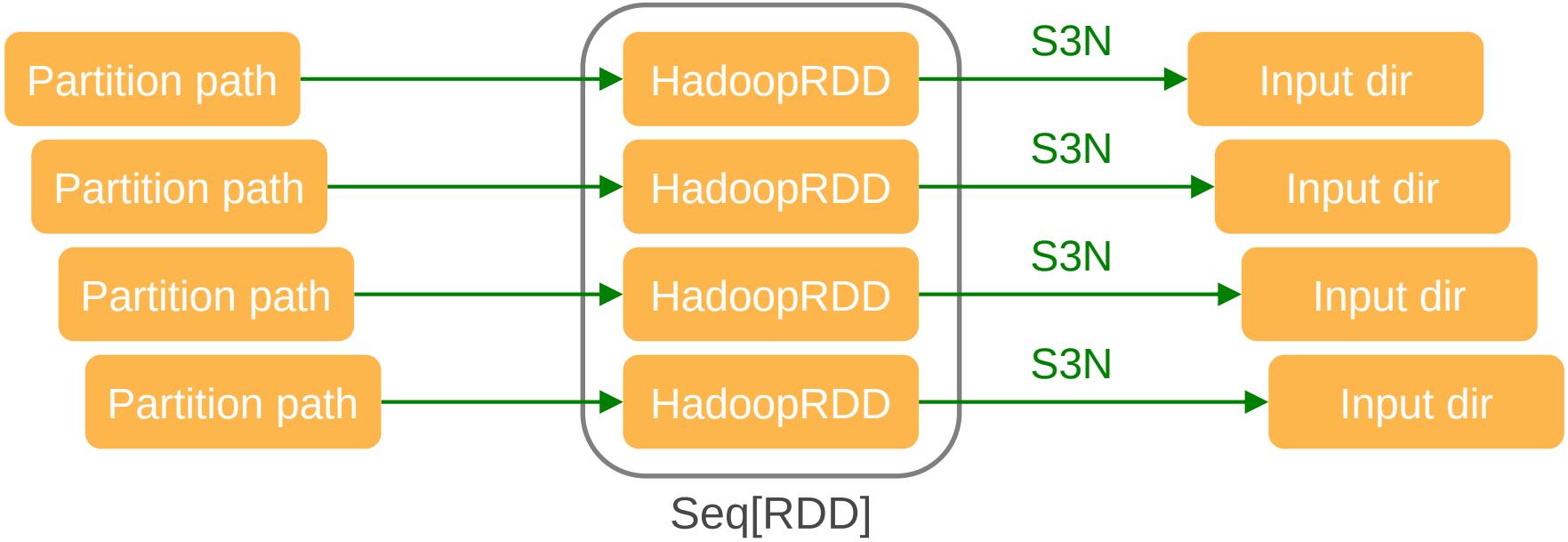
Input split computation

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 Amazon S3 is slow

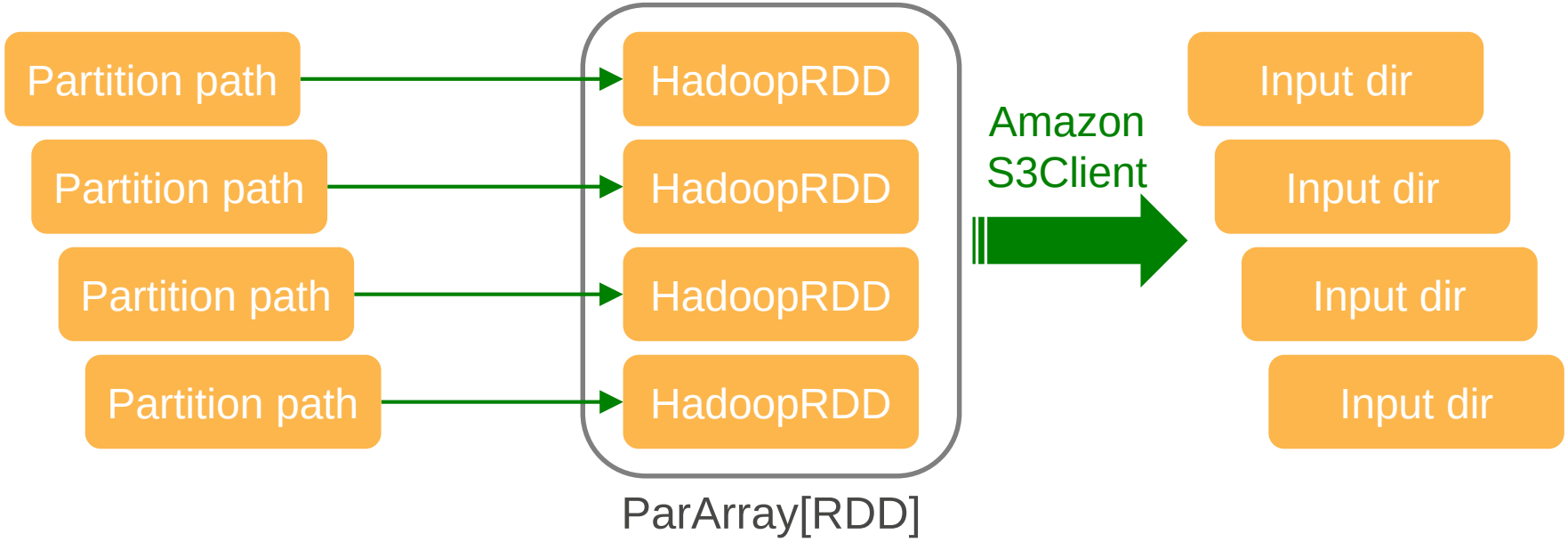
Cause

- Listing files on a per partition basis is slow
- S3N file system computes data locality hints

Solution

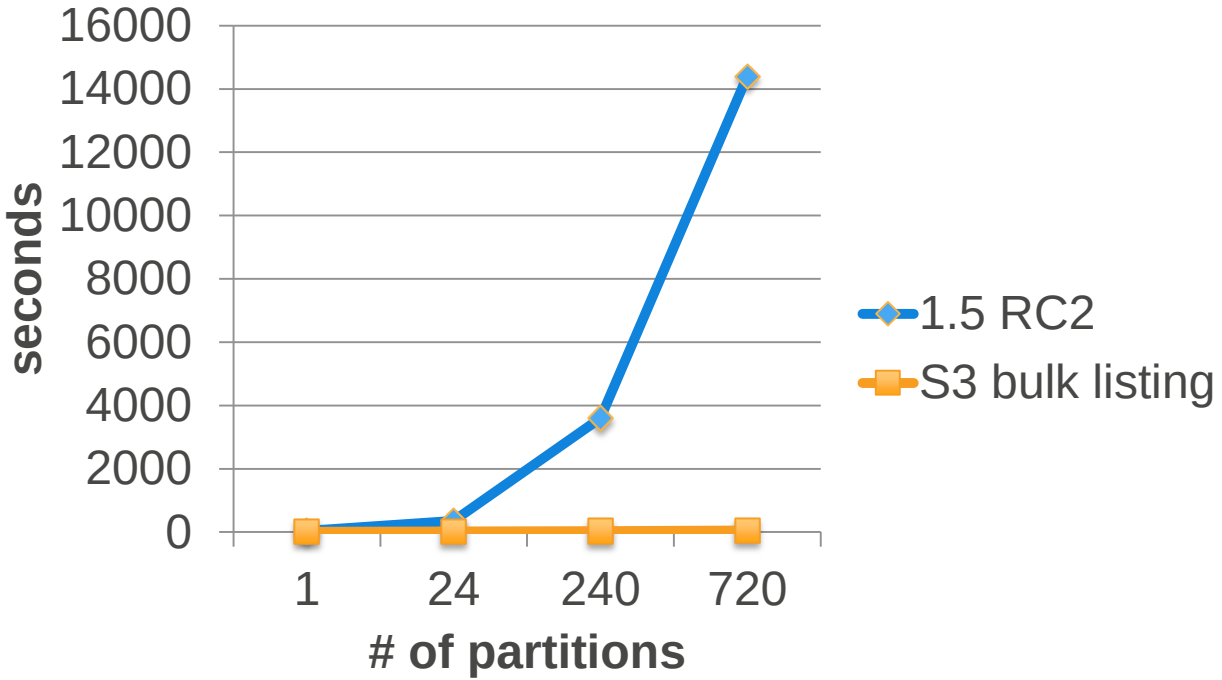
- Bulk list partitions in parallel using AmazonS3Client
- Bypass data locality computation for Amazon S3 objects

Amazon S3 Bulk Listing



Amazon S3 listing input dirs in parallel via AmazonS3Client

Performance Improvement




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

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

Challenges with Amazon S3

- Amazon S3 rename is copy and delete (non-atomic)
- Amazon S3 is eventually consistent

Amazon S3 Output Committer

How it works

- Each task writes output to local disk
- Output committer copies first successful task's output to Amazon S3

Advantages

- Avoid redundant Amazon S3 copy
- Avoid eventual consistency
- Always write to new paths

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

Next Steps for Netflix Integration

Metrics

Data lineage

Parquet integration

Key Takeaways

Our DW source of truth is on Amazon S3

Run custom Presto and Spark distros on Amazon EMR

- Presto as stand-alone clusters
- Spark co-tenant on Hadoop YARN clusters

We are committed to open source; you can run what we run

What’s Next

On Scaling and Optimizing Infrastructure…

Graceful shrink of Amazon EMR +

Heterogeneous instance groups in Amazon EMR +

Netflix Atlas metrics +

Netflix Spinnaker API =

Load-based expand/shrink of Hadoop YARN clusters

On Presto and Spark…

Expand new Presto use cases

Integrate Spark in Netflix big data platform

Explore Spark for ETL use cases

Remember to complete your evaluations!

Thank you!

Eva Tse & Daniel Weeks

Office hour: 12:00pm – 2pm Today @

Netflix Booth **#326**