Apache Spark and Object Stores —What you need to know

Steve Loughran
stevel@apache.org

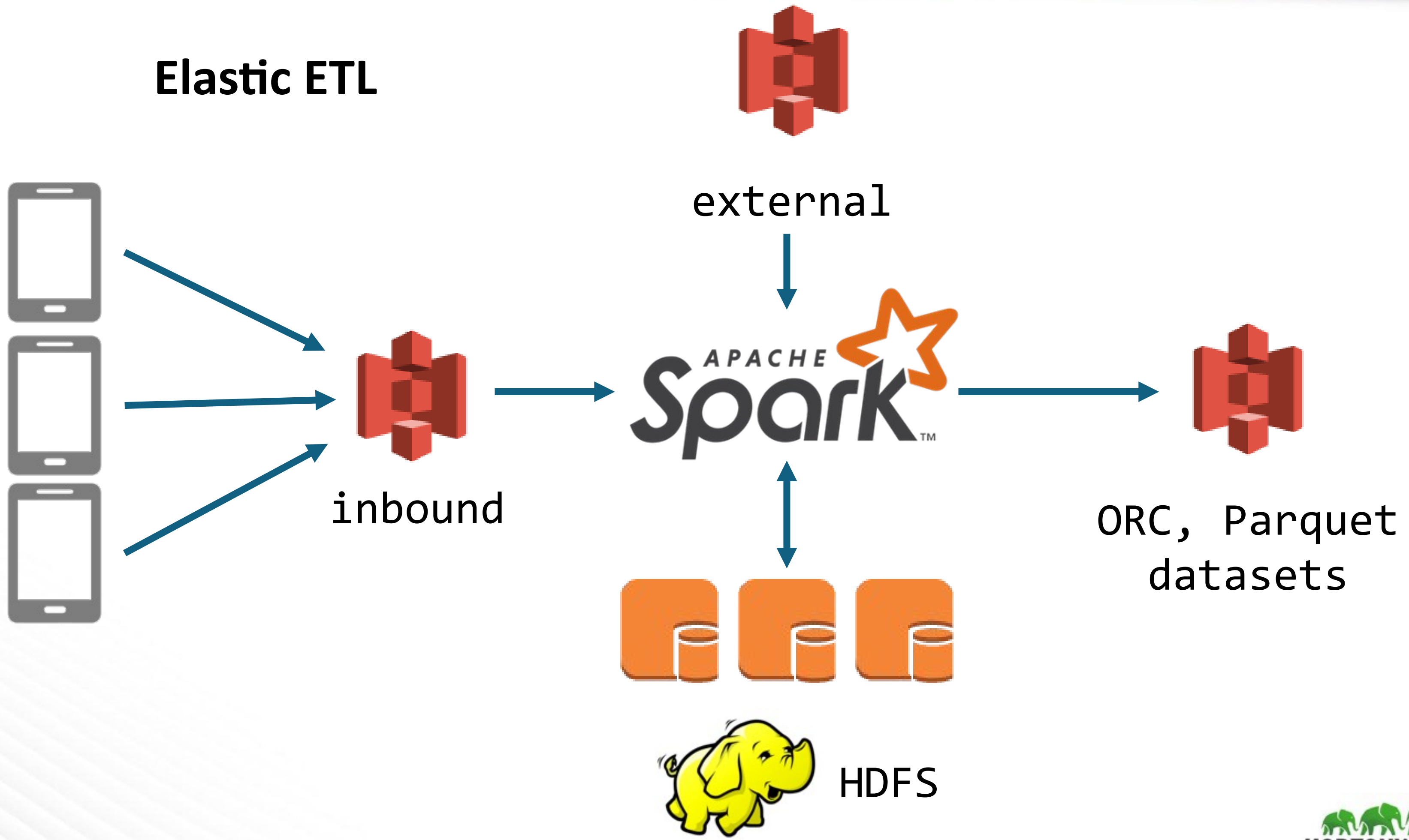
October 2016

Steve Loughran,
Hadoop committer, PMC member, …

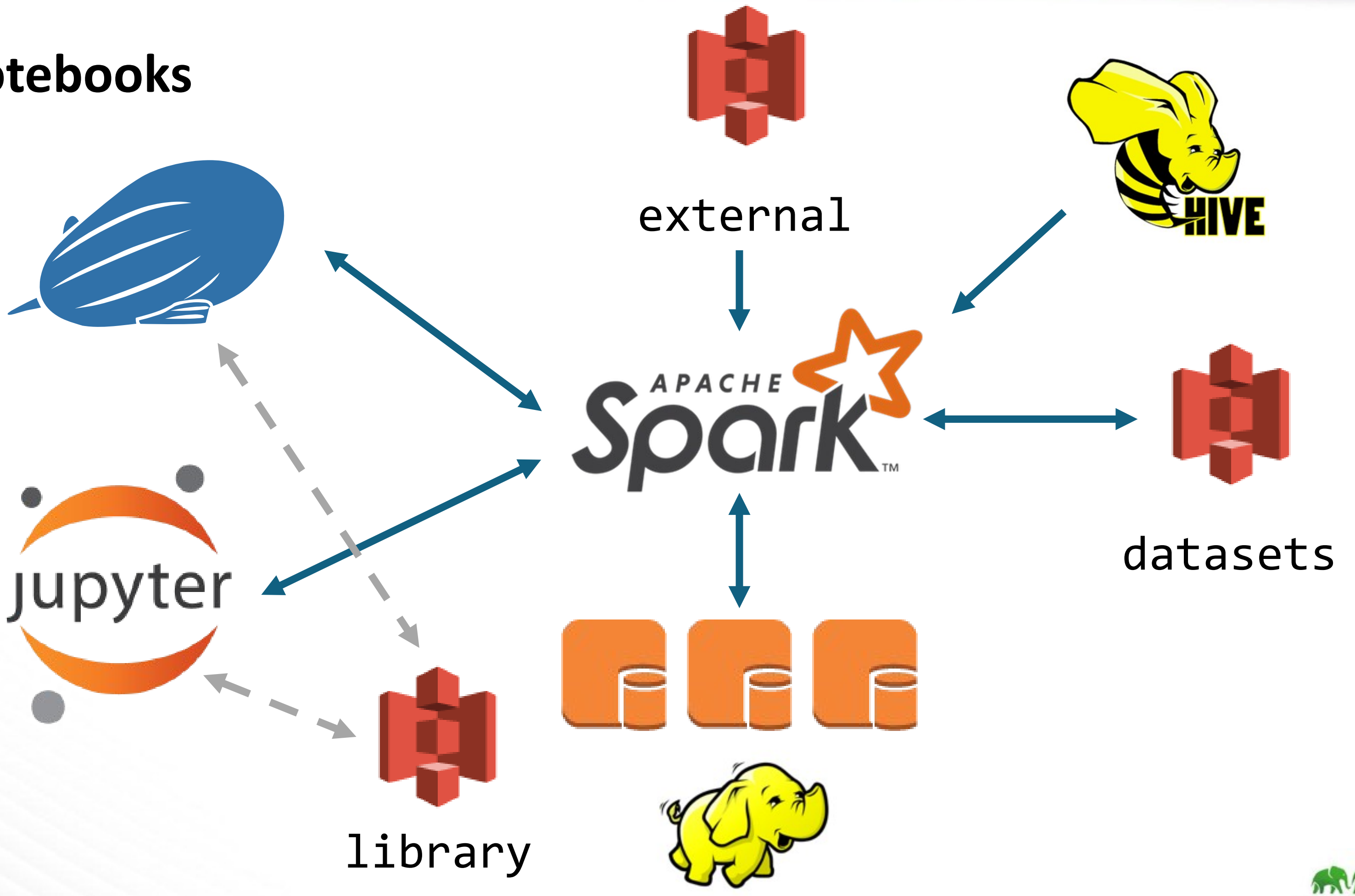
Chris Nauroth,
Apache Hadoop committer & PMC
ASF member

Rajesh Balamohan
Tez Committer, PMC Member

Elastic ETL

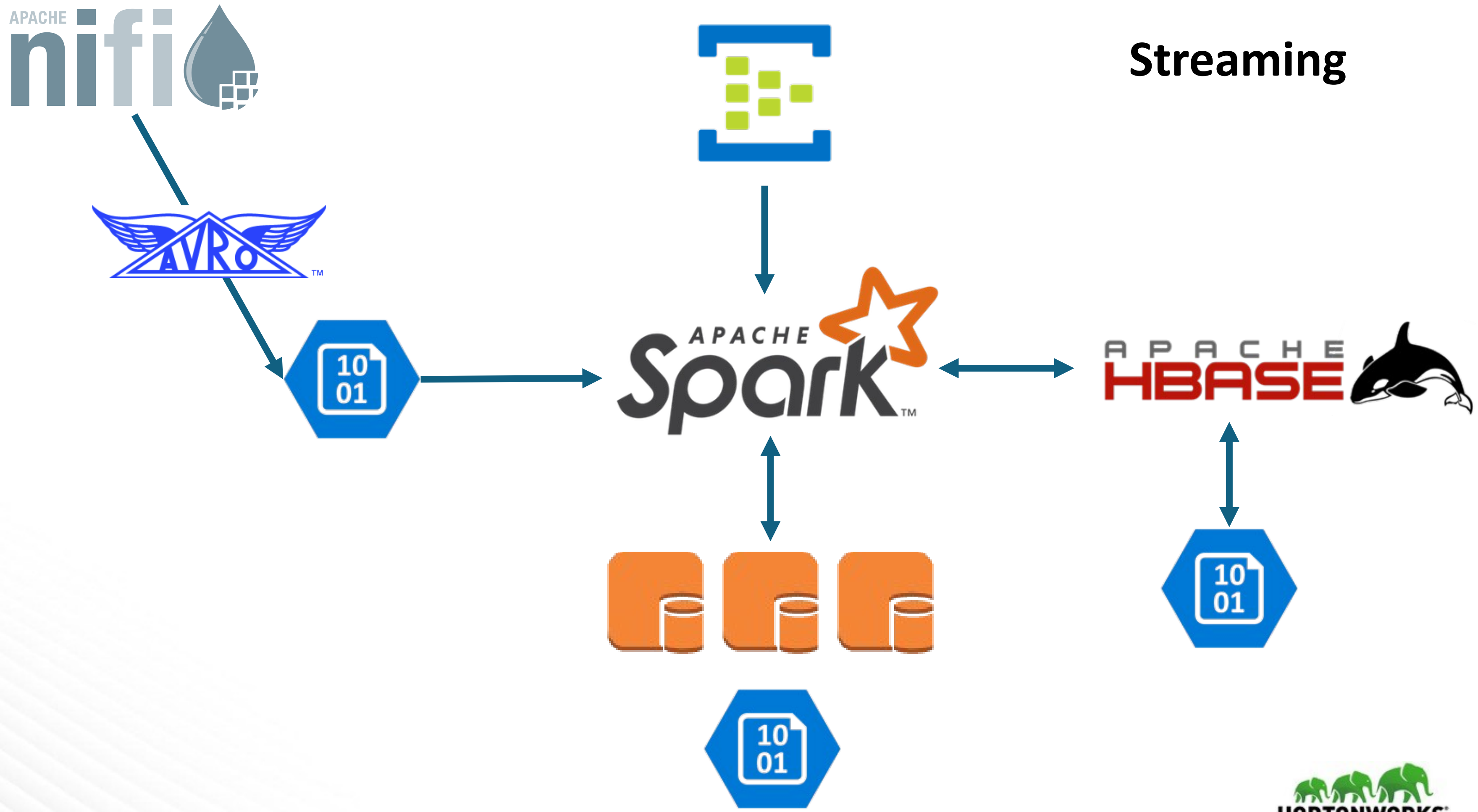


Notebooks



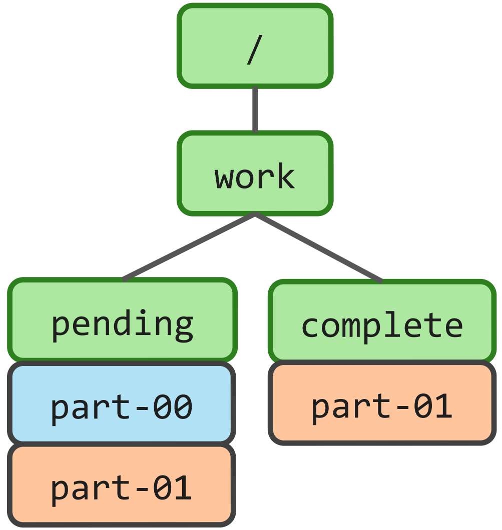
HORTONWORKS

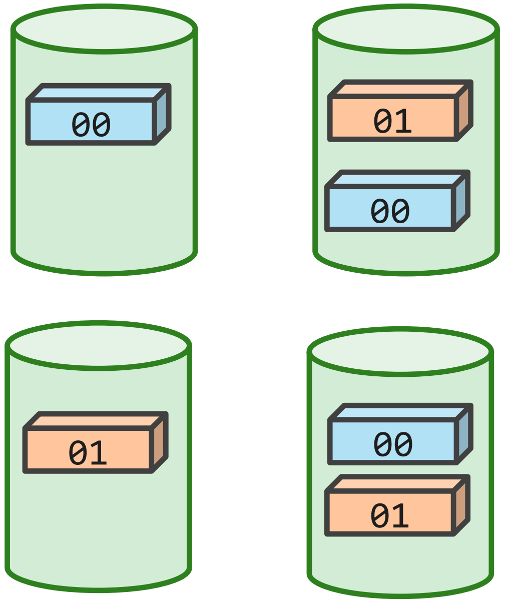
Streaming



HORTONWORKS

A Filesystem: Directories, Files → Data

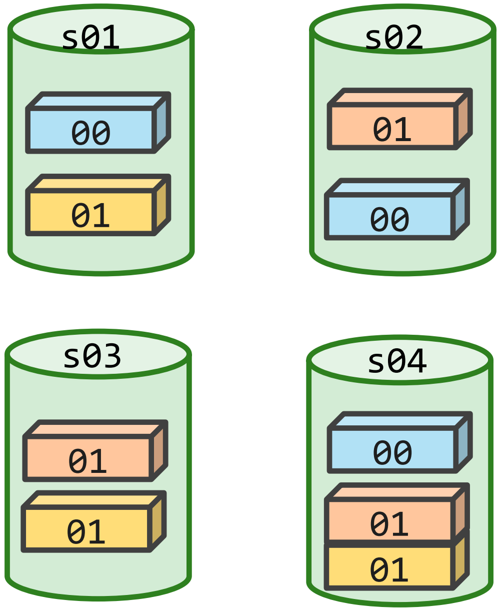




```
rename("/work/pending/part-01", "/work/complete")
```

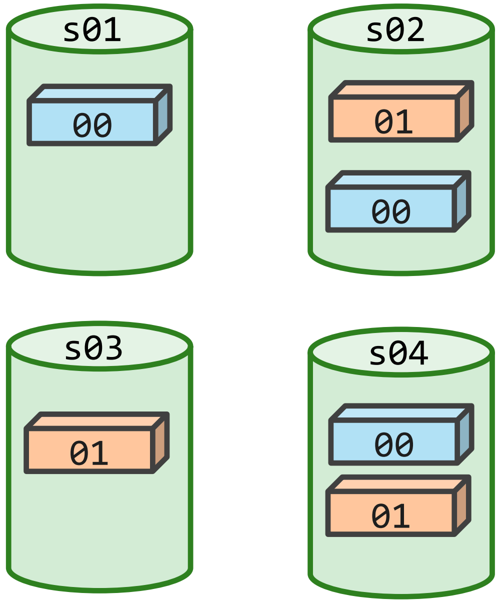
Object Store: hash(name)->blob

```
hash("/work/pending/part-00")
 ["s01", "s02", "s04"]

hash("/work/pending/part-01")
 ["s02", "s03", "s04"]

copy("/work/pending/part-01",
     "/work/complete/part01")


delete("/work/pending/part-01")
```



REST APIs

```
PUT /work/pending/part-01
... DATA ...

GET /work/pending/part-01
Content-Length: 1-8192

PUT /work/complete/part01
x-amz-copy-source: /work/pending/part-01

DELETE /work/pending/part-01

HEAD /work/complete/part-01

GET /?prefix=/work&delimiter=/
```



org.apache.hadoop.fs.FileSystem

hdfs wasb s3a swift adl gs

Four Challenges

1. Classpath
2. Credentials
3. Code
4. Commitment

Let's look At S3 and Azure

Use S3A to work with S3

(EMR: use Amazon's s3://)

Classpath: fix “No FileSystem for scheme: s3a”

```
hadoop-aws-2.7.x.jar

aws-java-sdk-1.7.4.jar
joda-time-2.9.3.jar
(jackson-*-2.6.5.jar)
```

Get Spark with Hadoop 2.7+ JARs

See SPARK-7481

Credentials

`core-site.xml` or `spark-default.conf`

```
spark.hadoop.fs.s3a.access.key MY_ACCESS_KEY
spark.hadoop.fs.s3a.secret.key MY_SECRET_KEY
```

`spark-submit` automatically propagates Environment Variables

```
export AWS_ACCESS_KEY=MY_ACCESS_KEY
export AWS_SECRET_KEY=MY_SECRET_KEY
```

NEVER: share, check in to SCM, paste in bug reports…

Authentication Failure: 403

```
com.amazonaws.services.s3.model.AmazonS3Exception:
 The request signature we calculated does not match
 the signature you provided.
 Check your key and signing method.
```

1. Check `joda-time.jar` & JVM version
2. Credentials wrong
3. Credentials not propagating
4. Local system clock (more likely on VMs)

Code: just use the URL of the object store

```
val csvdata = spark.read.options(Map(
    "header" -> "true",
    "inferSchema" -> "true",
    "mode" -> "FAILFAST"))
  .csv("s3a://landsat-pds/scene_list.gz")
```

...read time O(distance)

DataFrames

```
val landsat = "s3a://stevel-demo/landsat"
csvData.write.parquet(landsat)

val landsatOrc = "s3a://stevel-demo/landsatOrc"
csvData.write.orc(landsatOrc)

val df = spark.read.parquet(landsat)
val orcDf = spark.read.parquet(landsatOrc)
```

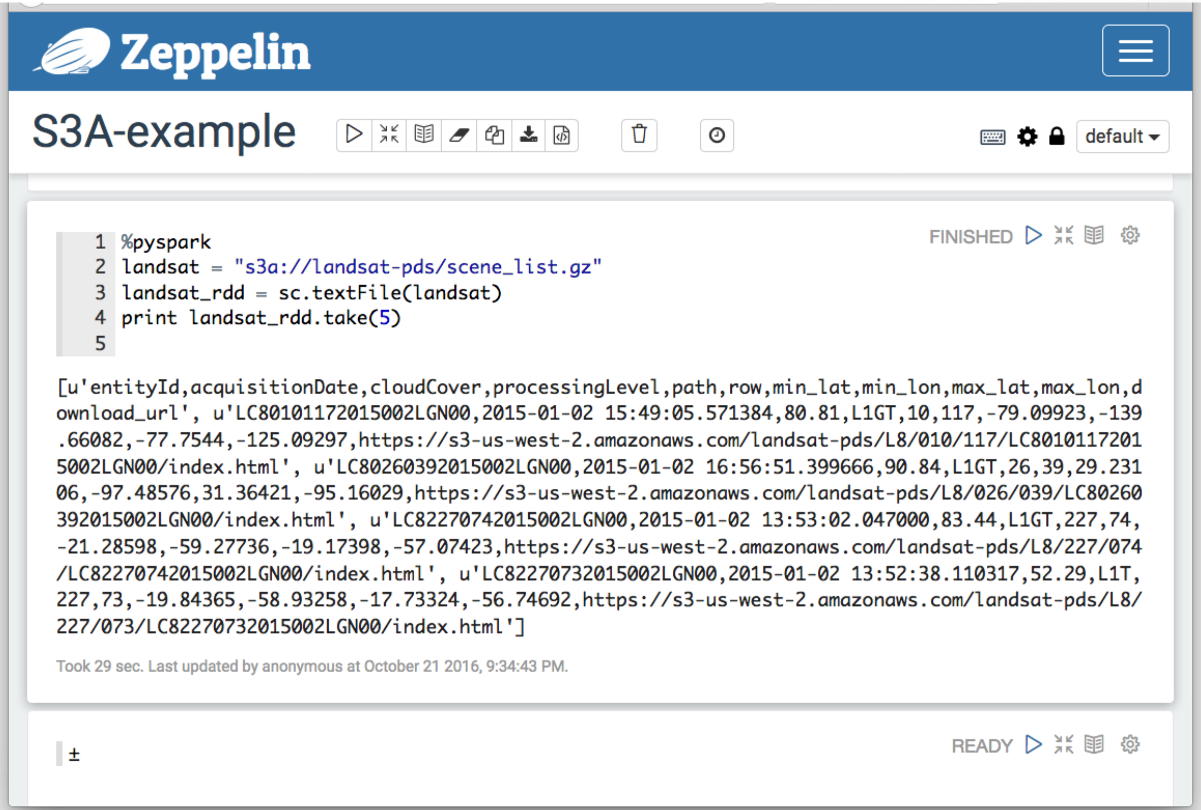
Finding dirty data with Spark SQL

```
val sqlDF = spark.sql(
  "SELECT id, acquisitionDate, cloudCover"
 + s" FROM parquet.`${landsat}`")

val negativeClouds = sqlDF.filter("cloudCover < 0")
negativeClouds.show()
```

* filter columns and data early
* whether/when to cache()?
* copy popular data to HDFS

spark-default.conf

```
spark.sql.parquet.filterPushdown true
spark.sql.parquet.mergeSchema false
spark.hadoop.parquet.enable.summary-metadata false

spark.sql.orc.filterPushdown true
spark.sql.orc.splits.include.file.footer true
spark.sql.orc.cache.stripe.details.size 10000

spark.sql.hive.metastorePartitionPruning true
```



```
%pyspark
landsat = "s3a://landsat-pds/scene_list.gz"
landsat_rdd = sc.textFile(landsat)
print landsat_rdd.take(5)
```

```
[u'entityId,acquisitionDate,cloudCover,processingLevel,path,row,min_lat,min_lon,max_lat,max_lon,d
ownload_url', u'LC80101172015002LGN00,2015-01-02 15:49:05.571384,80.81,L1GT,10,117,-79.09923,-139
.66082,-77.7544,-125.09297,https://s3-us-west-2.amazonaws.com/landsat-pds/L8/010/117/LC801011720l
5002LGN00/index.html', u'LC80260392015002LGN00,2015-01-02 16:56:51.399666,90.84,L1GT,26,39,29.231
06,-97.48576,31.36421,-95.16029,https://s3-us-west-2.amazonaws.com/landsat-pds/L8/026/039/LC80260
392015002LGN00/index.html', u'LC82270742015002LGN00,2015-01-02 13:53:02.047000,83.44,L1GT,227,74,
-21.28598,-59.27736,-19.17398,-57.07423,https://s3-us-west-2.amazonaws.com/landsat-pds/L8/227/074
/LC82270742015002LGN00/index.html', u'LC82270732015002LGN00,2015-01-02 13:52:38.110317,52.29,L1T,
227,73,-19.84365,-58.93258,-17.73324,-56.74692,https://s3-us-west-2.amazonaws.com/landsat-pds/L8/
227/073/LC82270732015002LGN00/index.html']
```

Took 29 sec. Last updated by anonymous at October 21 2016, 9:34:43 PM.

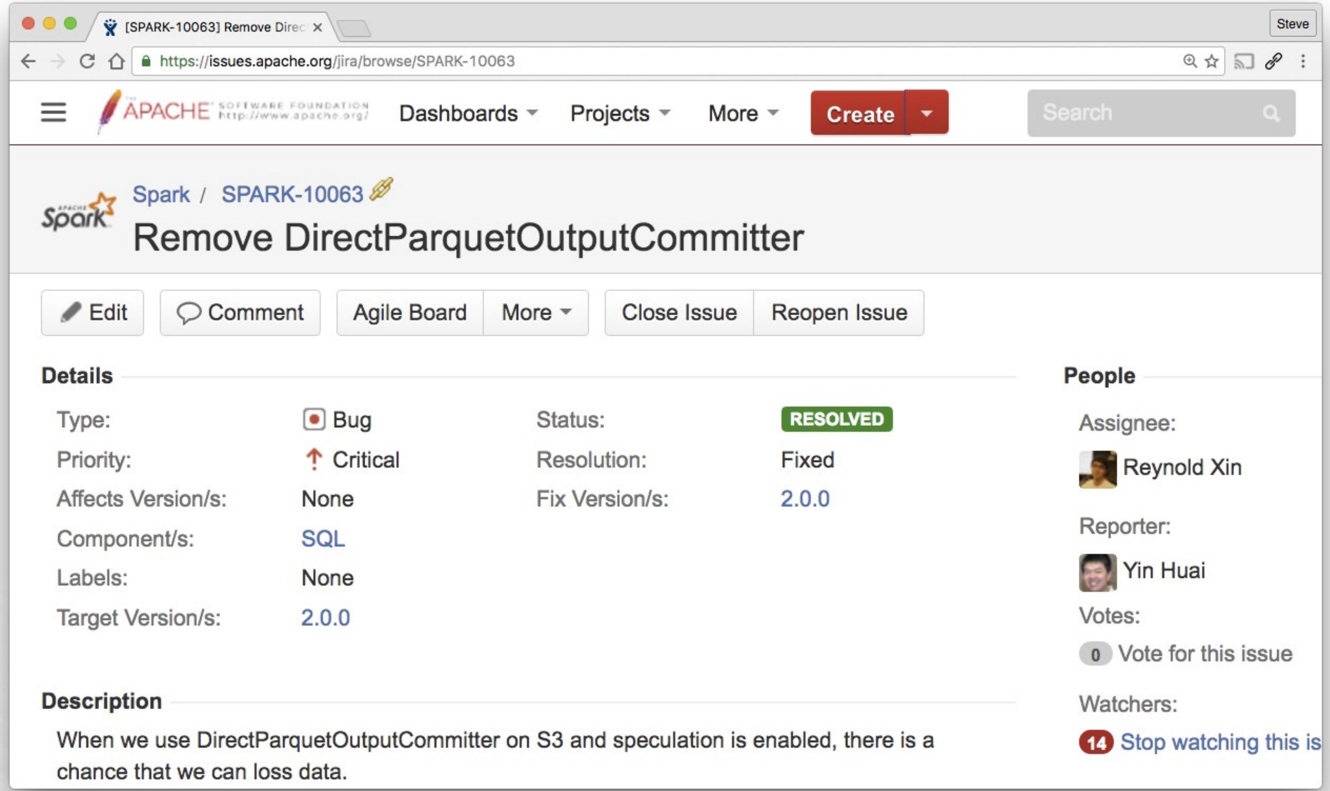
Notebooks? Classpath & Credentials

The Commitment Problem

- `rename()` used for atomic commitment transaction
- time to copy() + delete() proportional to data * files
- S3: 6+ MB/s
- Azure: a lot faster —*usually*

```
spark.speculation false
spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version 2
spark.hadoop.mapreduce.fileoutputcommitter.cleanup.skipped true
```

What about Direct Output Committers?



Spark / SPARK-10063

Remove DirectParquetOutputCommitter

Details

Type:	Bug	Status:	RESOLVED
Priority:	Critical	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	2.0.0
Component/s:	SQL		
Labels:	None		
Target Version/s:	2.0.0		

People

Assignee: Reynold Xin

Reporter: Yin Huai

Votes: 0 Vote for this issue

Watchers: 14 Stop watching this is

Description

When we use DirectParquetOutputCommitter on S3 and speculation is enabled, there is a chance that we can loss data.

Recent S3A Performance (Hadoop 2.8, HDP 2.5, CDH 5.9 (?))

```
// forward seek by skipping stream
spark.hadoop.fs.s3a.readahead.range 157810688

// faster backward seek for ORC and Parquet input
spark.hadoop.fs.s3a.experimental.input.fadvise random

// PUT blocks in separate threads
spark.hadoop.fs.s3a.fast.output.enabled true
```

Azure Storage: wasb://

A full substitute for HDFS

Classpath: fix "No FileSystem for scheme: wasb"

`wasb://` : Consistent, with very fast rename (hence: commits)

```
hadoop-azure-2.7.x.jar

azure-storage-2.2.0.jar

+ (jackson-core; http-components, hadoop-common)
```

Credentials: `core-site.xml / spark-default.conf`

```
<property>

<name>fs.azure.account.key.example.blob.core.windows.net</name>
 <value>0c0d44ac83ad7f94b0997b36e6e9a25b49a1394c</value>
</property>

spark.hadoop.fs.azure.account.key.example.blob.core.windows.net
  0c0d44ac83ad7f94b0997b36e6e9a25b49a1394c


wasb://demo@example.blob.core.windows.net
```

Example: Azure Storage and Streaming

```
val streaming = new
StreamingContext(sparkConf,Seconds(10))
val azure = "wasb://demo@example.blob.core.windows.net/in"
val lines = streaming.textFileStream(azure)
val matches = lines.map(line => {
    println(line)
    line
  })
matches.print()
streaming.start()
```

* PUT into the streaming directory
* keep the dir clean
* size window for slow scans

Not Covered

- Partitioning/directory layout
- Infrastructure Throttling
- Optimal path names
- Error handling
- Metrics

Summary

- Object Stores look just like any other URL
- …but do need classpath and configuration
- Issues: performance, commitment
- *Use Hadoop 2.7+ JARs*
- *Tune to reduce I/O*
- *Keep those credentials secret!*

APACHE:

BIG_DATA

EUROPE

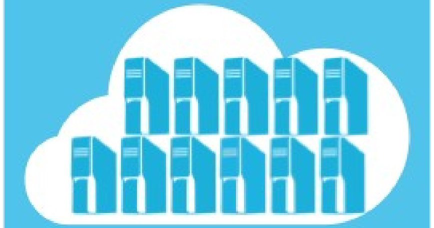
November 14 - 16, 2016 Melia Sevilla, Seville, Spain

#ApacheBigData

Backup Slides

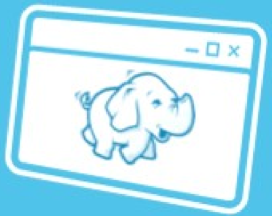
Why run your code in the cloud?

No Upfront
HW Costs



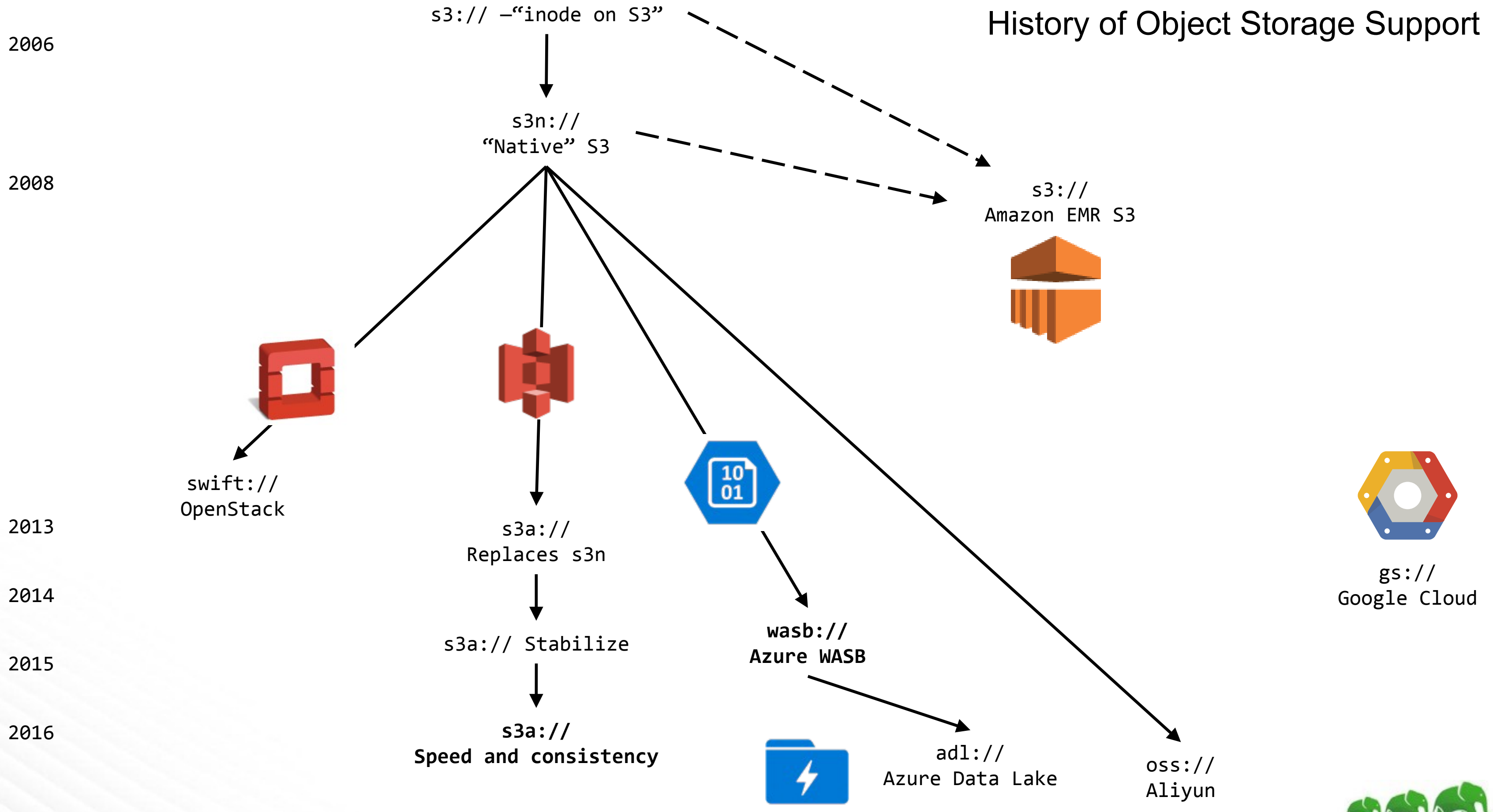
Unlimited
Elastic Scale

Ephemeral &
Long-Running



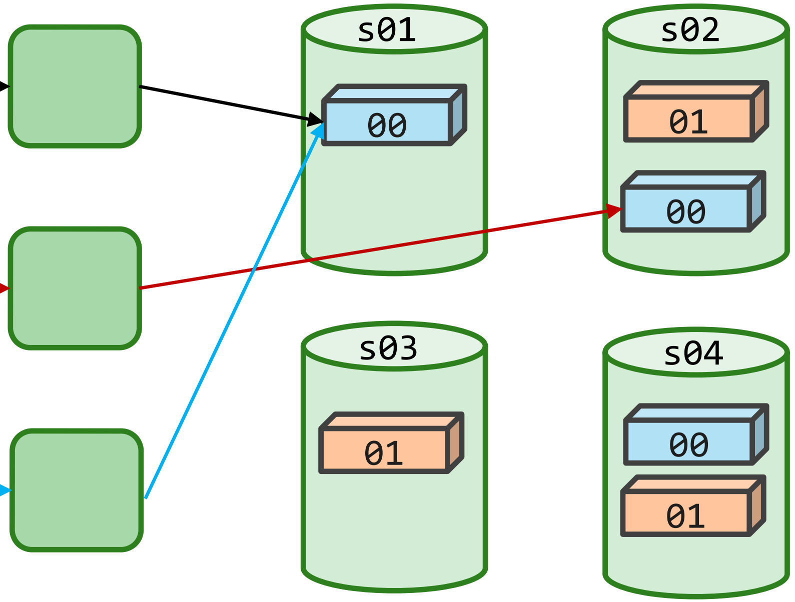
IT &
Business Agility

History of Object Storage Support



Often: Eventually Consistent

```
GET /work/pending/part-00
200

DELETE /work/pending/part-00
200

GET /work/pending/part-00
200
```



S3 Server-Side Encryption

- Encryption of data at rest at S3
- Supports the SSE-S3 option: each object encrypted by a unique key using AES-256 cipher
- Now covered in S3A automated test suites
- Support for additional options under development (SSE-KMS and SSE-C)

Next Steps for all Object Stores

- Output Committers
  - Logical commit operation decoupled from rename (non-atomic and costly in object stores)
- Object Store Abstraction Layer
  - Avoid impedance mismatch with FileSystem API
  - Provide specific APIs for better integration with object stores: saving, listing, copying
- Ongoing Performance Improvement
- Consistency

Performance Considerations When Running Queries

- Splits Generation
  - File formats like ORC provides threadpool in split generation
- ORC Footer Cache
  - hive.orc.cache.stripe.details.size > 0
  - Caches footer details; Helps in reducing data reads during split generation
- Reduce S3A reads in Task side
  - hive.orc.splits.include.file.footer=true
  - Sends ORC footer information in splits payload.
  - Helps reducing the amount of data read in task side.

Cloud Storage Connectors

Azure	WASB	● Strongly consistent ● Good performance ● Well-tested on applications (incl. HBase)
	ADL	● Strongly consistent ● Tuned for big data analytics workloads
Amazon Web Services	S3A	● Eventually consistent - consistency work in progress by Hortonworks ● Performance improvements in progress ● Active development in Apache
	EMRFS	● Proprietary connector used in EMR ● Optional strong consistency for a cost
Google Cloud Platform	GCS	● Multiple configurable consistency policies ● Currently Google open source ● Good performance ● Could improve test coverage

HORTONWORKS

Scheme	**Stable since**	**Speed**	**Consistency**	**Maintenance**
`s3n://`	Hadoop 1.0			(Apache)
`s3a://`	**Hadoop 2.7**	**2.8+**	***ongoing***	**Apache**
`swift://`	Hadoop 2.2			Apache
`wasb://`	**Hadoop 2.7**	**Hadoop 2.7**	**strong**	**Apache**
`adl://`	Hadoop 3			
`EMR s3://`	AWS EMR		For a fee	Amazon
`gs://`	???			Google @ github

Dependencies in Hadoop 2.8

```
hadoop-aws-2.8.x.jar

aws-java-sdk-core-1.10.6.jar
aws-java-sdk-kms-1.10.6.jar
aws-java-sdk-s3-1.10.6.jar
joda-time-2.9.3.jar
(jackson-*-2.6.5.jar)


hadoop-aws-2.8.x.jar

azure-storage-4.2.0.jar
```