

S3A and S3Guard

August 2017

The screenshot shows the Microsoft Azure portal interface for creating a new Data Lake Store. The page title is 'New Data Lake Store'. The form includes the following fields:

- Name:** 'petastore' (with a green checkmark icon and the URL 'petastore.azuredatalakestore.net' below it).
- * Subscription:** 'Visual Studio Enterprise' (dropdown menu).
- * Resource Group:** 'Create new' (selected) and 'Use existing' (radio buttons).
- * Location:** 'North Europe' (dropdown menu).
- Pricing:** '1 PB for 24,880 USD' (dropdown menu, highlighted with a red circle).
- Encryption Settings:** 'Enabled' (with a right arrow icon).
- Pin to dashboard:** An unchecked checkbox.
- Buttons:** 'Create' (blue button) and 'Automation options' (blue link).

Why Cloud?

- No upfront hardware costs – Pay as you use
- Elasticity
- Often lower TCO
- Natural for Data ingress for IoT, mobile apps, ..
- Business agility

Shared Data and Cloud Storage

- ◆ Cloud Storage is the Shared Data Lake
 - For both Hadoop and Cloud-native (non-Hadoop) Apps
 - Lower Cost
 - HDFS via EBS can get expensive
 - HDFS's role changes
 - Built-in geo-distribution and DR
- ◆ Challenges
 - Cloud storage designed for scale, low cost and geo-distribution
 - Performance is slower – was not designed for data-intensive apps
 - Cloud storage segregated from compute
 - API and semantics not like a FS – especially wrt. consistency

Making Object Stores work for Big Data Apps

◆ Focus Areas

- Address cloud storage consistency
- Performance (changes in connectors and frameworks)
- Caching in memory and local storage

◆ Other issues not covered in this talk

- Shared Metastore, Common Governance, Security across multiple clusters
- Columnar access control to Tabular data

See Hortonworks cloud offering

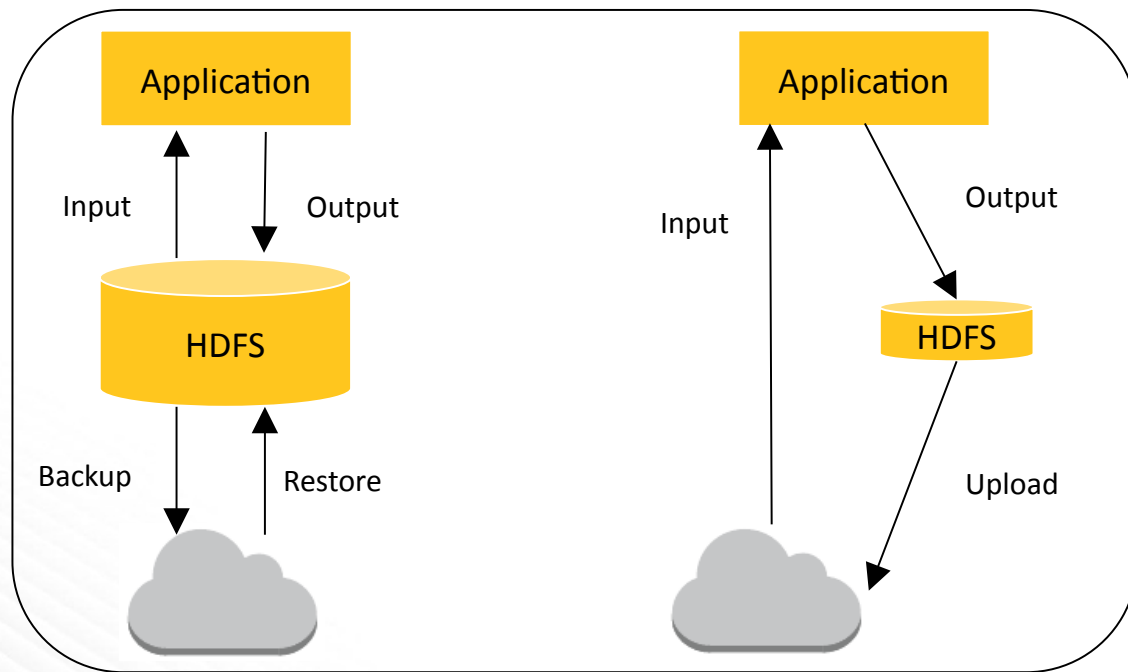
Also Attend 4pm Talk today “Cloudy with a chance of Hadoop...”



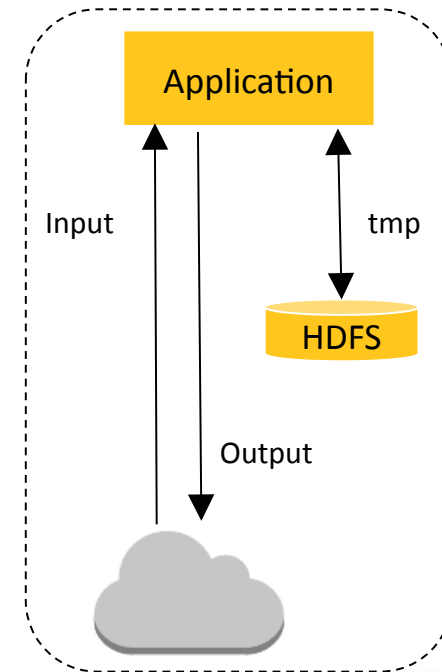
Cloud Storage Integration: Evolution for Agility

Evolution towards cloud storage as the persistent Data Lake

Goal



AWS -today ->>>>



Azure



Some Use Cases

Elastic ETL



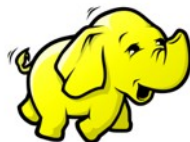
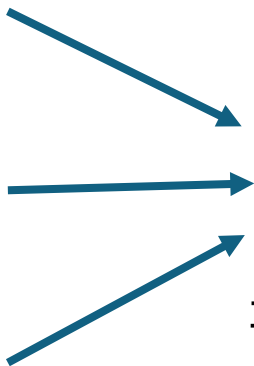
external



ORC
datasets

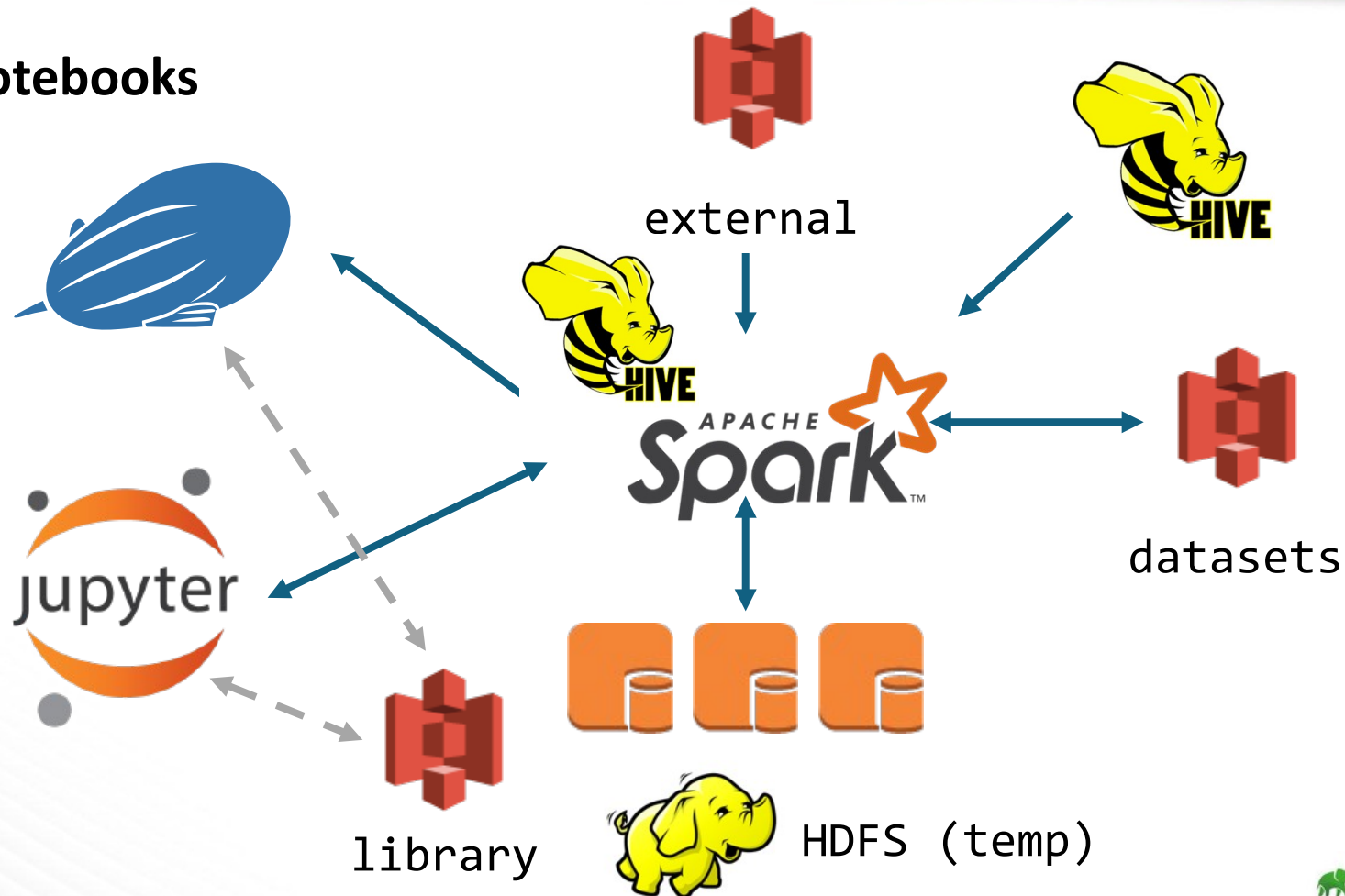


inbound



HDFS (temp)

Notebooks





Streaming

*Flink, Spark,
Storm*

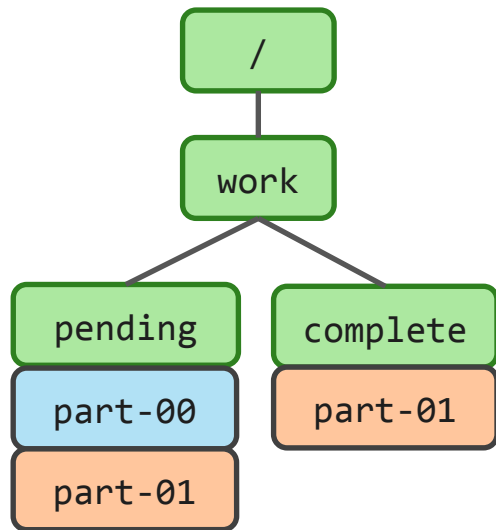




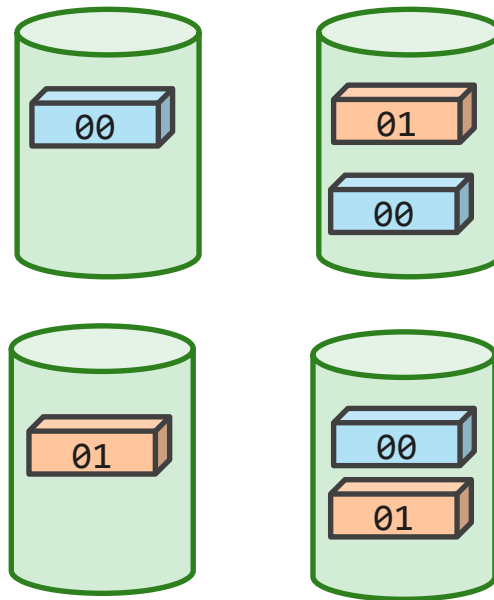
Danger: Object stores are not filesystems

Cost & Geo-distribution over
Consistency and Performance

A Filesystem: Directories, Files → Data



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



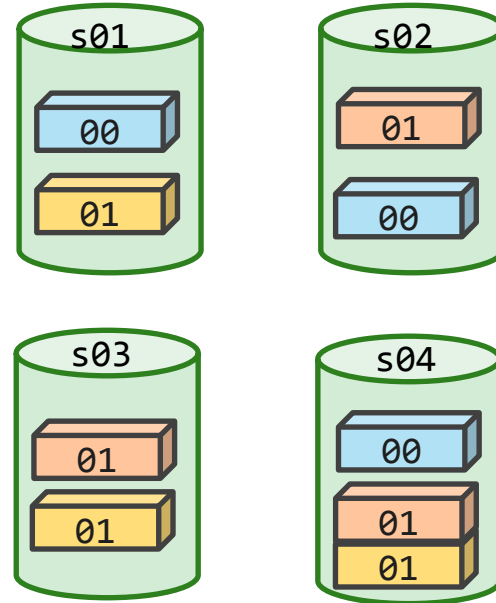
Object Store: hash(name)⇒data

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

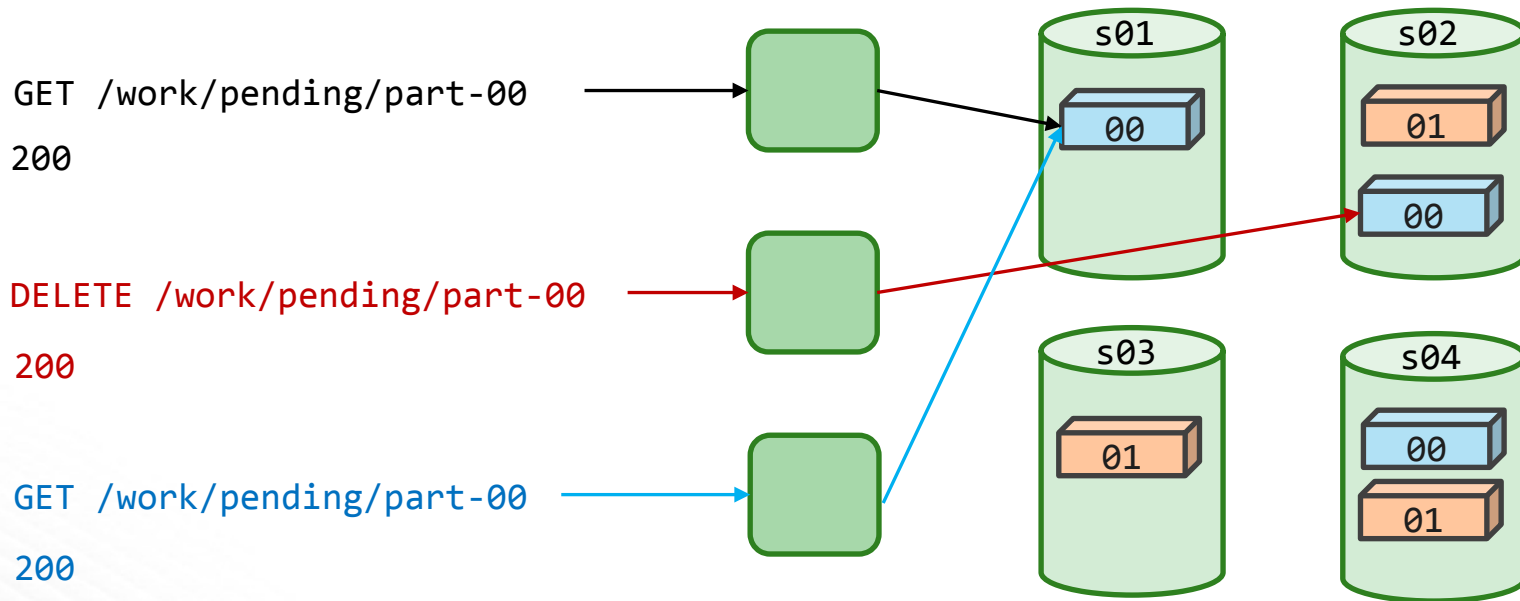
```
hash("/work/pending/part-01")  
["s02", "s03", "s04"]
```

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

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



Often: Eventually Consistent



Eventual Consistency problems

- ◆ When listing a directory
 - Newly created files may not yet be visible, deleted ones still present
- ◆ After updating a file
 - Opening and reading the file may still return the previous data
- ◆ After deleting a file
 - Opening the file may succeed, returning the data
- ◆ While reading an object
 - If object is updated or deleted during the process

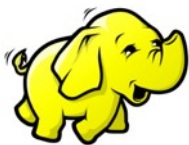
The dangers of Eventual Consistency and Atomicity

- ◆ Temp Data leftovers
 - Annoying Garbage or Worse if direct output committer is used
- ◆ List inconsistency means new data may not be visible
- ◆ Lack of atomic rename() can leave output directories inconsistent

***You can get bad or missing data and not even notice
Especially if only a portion of your large data is missing***



`org.apache.hadoop.fs.FileSystem`



hdfs



wasb



s3a



swift

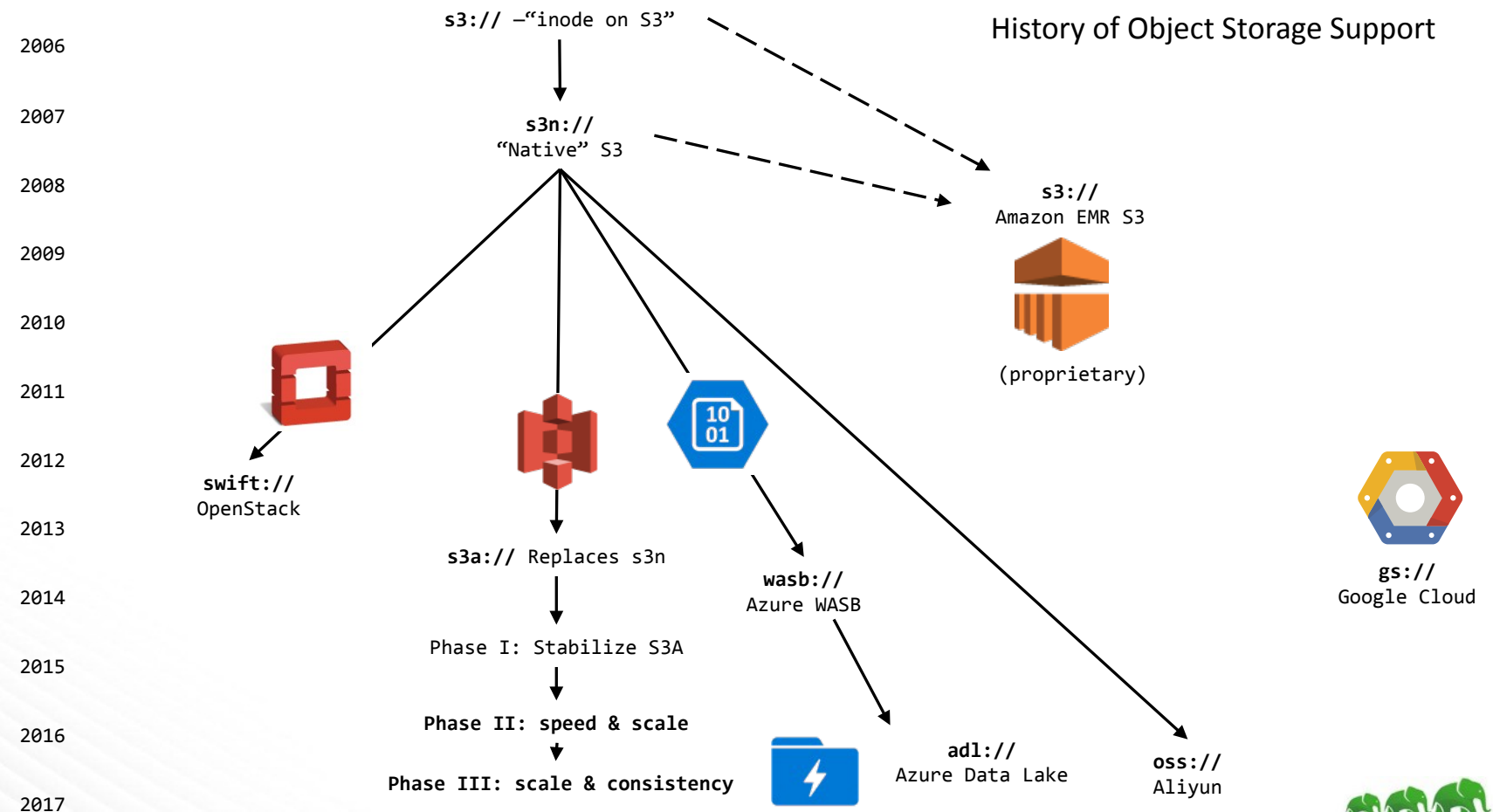


adl



gs

History of Object Storage Support



Cloud Storage Connectors

Azure	WASB	<ul style="list-style-type: none">● Strongly consistent● Good performance● Well-tested on applications (incl. HBase)
	ADL	<ul style="list-style-type: none">● Strongly consistent● Tuned for big data analytics workloads
Amazon Web Services	S3A	<ul style="list-style-type: none">● Eventually consistent - consistency work in progress by Hortonworks● Performance improvements recently and in progress● Active development in Apache
	EMRFS	<ul style="list-style-type: none">● Proprietary connector used in EMR● Optional strong consistency for a cost
Google Cloud Platform	GCS	<ul style="list-style-type: none">● Multiple configurable consistency policies● Currently Google open source● Good performance● Could improve test coverage

Problem: S3 Analytics is too slow/broken

1. Analyze benchmarks and bug-reports
2. Optimize the non-io metadataops (very cheap on HDFS)
3. Fix Read path for Columnar Data
4. Fix Write path
5. Improve query partitioning
6. The Commitment Problem



Flame Graph

readFully(pos)

getFileStatus()

read()

org/apache/orc/impl/TreeReaderFact... org/apac... or...

org/apache/hadoop/hive/llap/io/decode/OrcEncod... org/apac...

org/apache/hadoop/hive/llap/io/decode/OrcEncod... org/apache/tez/run... org/apache/te...

org/apache/hadoop/hive/llap/io/encoded/OrcEncodDat... org/apache/tez/run... org/apache/te...

org/apache/hadoop/hive/llap/io/encoded/OrcEncodDat... [p... call_stub call_stub o... call_stub

call_stub Ja... JVM_DoPrivileged JVM_DoPrivile... J... JVM_DoPrivileged

JavaCalls::call_helper JV... java/security/Acce... java/security/AccessCon... JavaCalls::call_helper

JVM_DoPrivileged ja... org/apache/hadoop/hive/llap/io/encoded/OrcEncodDat... org/apache/tez/run... org/apache/te... org/apache/hadoop/secur...

org/apache/hadoop/hive/llap/io/encoded/OrcEncodDat... ja... org/apache/tez/run... org/apache/te... org/apache/tez/runtime/...

org/apache/tez/common/CallableWithNdc::call or... org/apache/tez/common/C... org/apache/tez/runtime/...

org/apache/hadoop/hive/llap/daemon/impl/StatsRecording... or... org/apache/commons... org/apache/ha... org/apache/hadoop/hive/...

java/util/concurrent/FutureTask::run or... org/apache/hadoop/hive/llap/exec/vecto... java/util/concurrent/Futu...

Interpreter java/util/concurrent/ThreadPoolExecu... java/util/concurrent/ThreadPo...

Interpreter [perf-70288.map] java/util/concurrent/ThreadPo...

Interpreter java/lang/Thread::run

call_stub

C.. JavaCalls::call_helper

C.. JavaCalls::call_virtual

C.. JavaCalls::call_virtual

Car.. Co.. thread_entry

Old.. St.. JavaThread::thread_main_inner

GCTaskThr.. JavaThread::run

java_start

start_thread

java

The Performance Killers

`getFileStatus(Path) (+ isDirectory(), exists())`

```
HEAD path           // file?
HEAD path + "/"     // empty directory?
LIST path           // path with children?
```

`read(long pos, byte[] b, int idx, int len)`

`readFully(long pos, byte[] b, int idx, int len)`

`listFiles(Path)`

`rename(source, dest)`

Positioned reads: close + GET, close + GET

```
read(long pos, byte[] b, int idx, int len)
    throws IOException {
    long oldPos = getPos();
    int nread = -1;
    try {
        seek(pos);
        nread = read(b, idx, len);
    } catch (EOFException e) {
    } finally {
        seek(oldPos);
    }
    return nread;
}
```

seek() is the killer, especially the seek() back

HADOOP-12444 Support lazy seek in S3AInputStream

```
public synchronized void seek(long pos)
    throws IOException {
    nextReadPos = targetPos;
}
```

+configurable readahead before open/close()

```
<property>
  <name>fs.s3a.readahead.range</name>
  <value>256K</value>
</property>
```

But: ORC reads were still underperforming

Hadoop 2.8/HDP 2.6 transforms I/O performance!

// forward seek by skipping stream

`fs.s3a.readahead.range=256K`

// faster backward seek for Columnar Storage

`fs.s3a.experimental.input.fadvise=random`

// Write-IO - enhanced data upload (parallel background uploads)

// Additional flags for mem vs disk

`fs.s3a.fast.output.enabled=true`

`fs.s3a.multipart.size=32M`

`fs.s3a.fast.upload.active.blocks=8`

// Additional per-bucket flags

—see *HADOOP-11694* for lots more!

HADOOP-13203: fs.s3a.experimental.input.fadvise

// Before

```
request = new GetObjectRequest(bucket, key)
    .withRange(pos, contentLength - 1);
```

// after

```
finish = calculateRequestLimit(inputPolicy, pos,
    length, contentLength, readahead);
```

```
request = new GetObjectRequest(bucket, key)
    .withRange(pos, finish);
```

High performance random IO for column data

Every HTTP request is precious

- ◆ HADOOP-13162: Reduce number of getFileStatus calls in mkdirs()
- ◆ HADOOP-13164: Optimize deleteUnnecessaryFakeDirectories()
- ◆ HADOOP-13406: Consider reusing filestatus in delete() and mkdirs()
- ◆ HADOOP-13145: DistCp to skip getFileStatus when not preserving metadata
- ◆ HADOOP-13208: listFiles(recursive=true) to do a bulk listObjects

see HADOOP-11694

Write Pipeline

- ◆ PUT blocks as part of a multipart
- ◆ Parallel uploads during data creation
- ◆ Buffer to disk (default), heap or byte buffers
- ◆ Great for distcp

```
fs.s3a.fast.upload=true  
fs.s3a.multipart.size=32M  
fs.s3a.fast.upload.active.blocks=8
```

S3guard

Fast, consistent S3 metadata

HADOOP-13445

S3Guard: Fast And Consistent S3 Metadata

◆ Goals

- Provide consistent list and get status operations on S3 objects written with S3Guard enabled
 - `listStatus()` after put and delete
 - `getFileStatus()` after put and delete
- Performance improvements that impact real workloads
- Provide tools to manage associated metadata and caching policies.

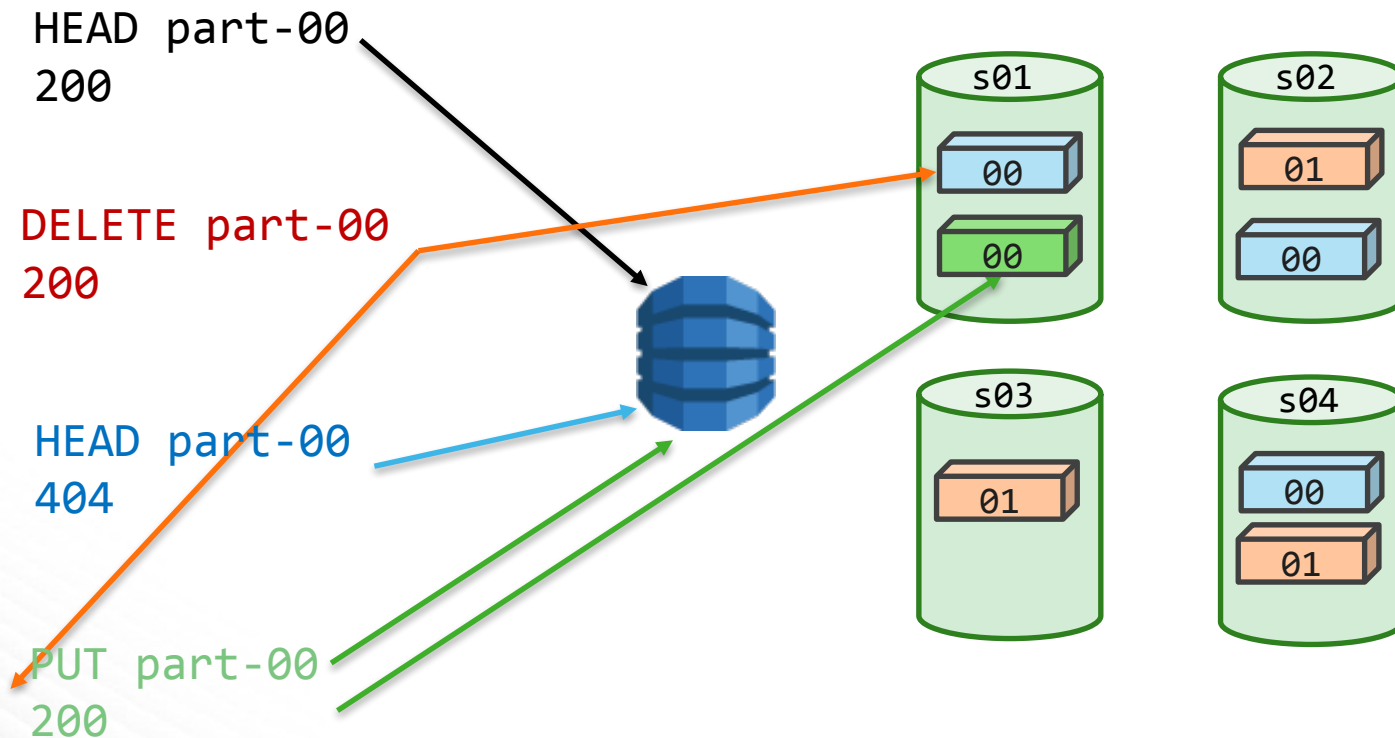
◆ Again, 100% open source in Apache Hadoop community

- Hortonworks, Cloudera, Western Digital, Disney ...

◆ Inspired by Apache licensed *S3mper* project from Netflix

◆ Seamless integration with S3AFileSystem

Use DynamoDB as fast, consistent metadata store



Installation and use

- ◆ Provision a DynamoDB table in same region as HDC cluster & buckets (best cost/IO resilience through shared table across >1 bucket)
- ◆ Enable S3Guard in clients
- ◆ Authoritative mode = fastest —but can we trust it yet?

Settings

```
<property>  
  <name>fs.s3a.metadatastore.impl</name>  
  <value>org.apache.hadoop.fs.s3a.s3guard.DynamoDBMetadataStore</value>  
</property>
```

```
<property>  
  <name>fs.s3a.s3guard.ddb.table</name>  
  <value>ireland-team</value>  
</property>
```

```
<property>  
  <name>fs.s3a.metadatastore.authoritative</name>  
  <value>>false</value>  
</property>
```


hadoop s3guard

```
hadoop s3guard init -meta dynamodb://ireland-team s3a://ireland-1
```

```
hadoop s3guard import s3a://ireland-1
```

```
hadoop s3guard diff s3a://ireland-1
```

```
hadoop s3guard destroy s3a://ireland-1
```

```
hadoop s3guard prune -days 7 s3a://ireland-1
```

(see also DynamoDB TTL option)

Dynamic table creation

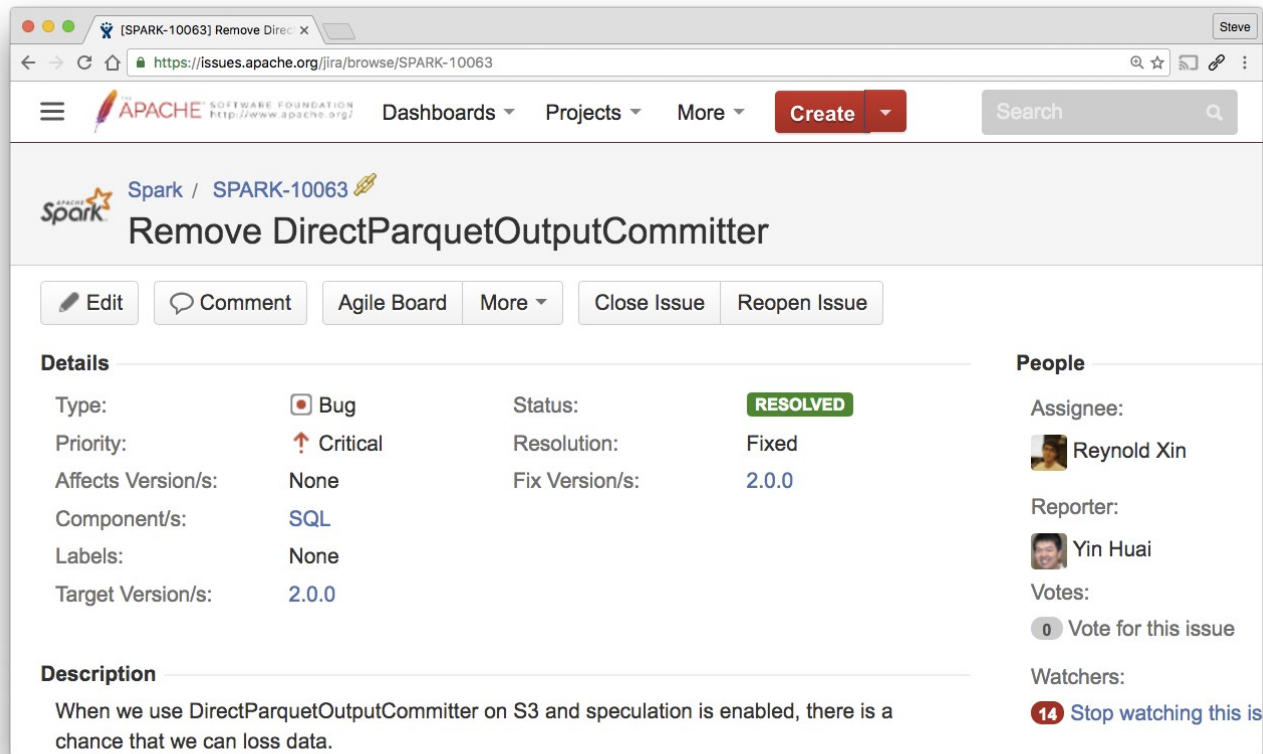
```
<property>  
  <name>fs.s3a.s3guard.ddb.table.create</name>  
  <value>true</value>  
</property>
```

```
<property>  
  <name>fs.s3a.s3guard.ddb.table.capacity.read</name>  
  <value>10</value>  
</property>
```

```
<property>  
  <name>fs.s3a.s3guard.ddb.table.capacity.write</name>  
  <value>10</value>  
</property>
```

Better to share table, explicitly create, monitor in AWS console

Spark's Direct Output Committer? Risk of Corruption of data



The screenshot shows the Apache JIRA issue page for SPARK-10063. The browser address bar shows the URL <https://issues.apache.org/jira/browse/SPARK-10063>. The page header includes the Apache Software Foundation logo and navigation links: Dashboards, Projects, More, Create, and Search. The issue title is "Remove DirectParquetOutputCommitter". Below the title are buttons for Edit, Comment, Agile Board, More, Close Issue, and Reopen Issue. The "Details" section lists the following information:

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

The "Description" section contains the text: "When we use DirectParquetOutputCommitter on S3 and speculation is enabled, there is a chance that we can loss data."

The "People" section lists the Assignee: Reynold Xin, Reporter: Yin Huai, and Votes: 0. It also shows 14 watchers and a link to "Stop watching this issue".

[HADOOP-13786] Add S3Guard X

Secure https://issues.apache.org/jira/browse/HADOOP-13786

☆

APACHE SOFTWARE FOUNDATION http://www.apache.org/

Dashboards ▾

Projects ▾

Issues ▾



Service Desk ▾

Agile ▾

Create

Search

?

 Hadoop Common / HADOOP-13786 

Add S3Guard committer for zero-rename commits to consistent S3 endpoints

Edit

Comment


Assign

More ▾


Cancel Patch


Resolve Issue

Resume Progress



Details

Type:  New Feature


Priority:  Major

Affects Version/s: HADOOP-13345

Component/s: fs/s3

Labels: None


Target Version/s: HADOOP-13345


Status:  PATCH AVAILABLE


Resolution: Unresolved


Fix Version/s: None

People

Assignee:  Steve Loughran

Reporter:  Steve Loughran

Votes:  1

Watchers:  22 [Stop watching this issue](#)

Dates

Created: 02/Nov/16 18:16

Updated: Yesterday

Agile

[View on Board](#)

Description

A goal of this code is "support O(1) commits to S3 repositories in the presence of failures". Implement it, including whatever is needed to demonstrate the correctness of the algorithm. (that is, assuming that s3guard provides a consistent view of the presence/absence of blobs, show that we can commit directly).

I consider ourselves free to expose the blobstore-ness of the s3 output streams (ie. not visible until the close()), if we need to use that to allow us to abort commit operations.

Netflix Staging Committer

1. Saves output to file://
2. Task commit: upload to S3A as multipart PUT —*but does not commit the PUT, just saves the information about it to hdfs://*
3. Normal commit protocol manages task and job data promotion in HDFS
4. Final Job committer reads pending information and generates final PUT —possibly from a different host

Outcome:

- ◆ No visible overwrite until final job commit: resilience and speculation
- ◆ Task commit time = data/bandwidth
- ◆ Job commit time = POST * #files

Availability

- Read + Write in HDP 2.6 and Apache Hadoop 2.8
- S3Guard: preview of DDB caching in HDC
- Commit to ASF trunk/branch-2 this month
...then backport
- Zero-rename commit: *work in progress*

Big thanks to:

Rajesh Balamohan

Mingliang Liu

Chris Nauroth

Dominik Bialek

Ram Venkatesh

Everyone in QE, RE

+ everyone who reviewed/tested, patches and
added their own, filed bug reports and measured
performance

Questions?

stevel@hortonworks.com
sanjay@hortonworks.com

@steveloughran
@srr

