Walmart
Save money. Live better.

9 May 2017

Swifta

A performant Hadoop file system driver for Swift

Mengmeng Liu   Andy Robb   Ray Zhang

Spark
presto

Our Big Data Journey

- One of two teams that run multi-tenant Hadoop ecosystem at Walmart
- Large, shared clusters since 2012
- Project to enable single-tenant YARN/Spark/Presto via OpenStack and OneOps
  - Predictable job performance
  - Software version flexibility
  - Use case flexibility (e.g. streaming)
  - Independent expansion for compute vs storage
  - Maintenance for persistent vs hyper-automated/virtualized
  - Maintain "user environment"
- (Different team) started building on-prem OpenStack/Ceph in 2016

Anticipated Audience (very low-level details ahead)

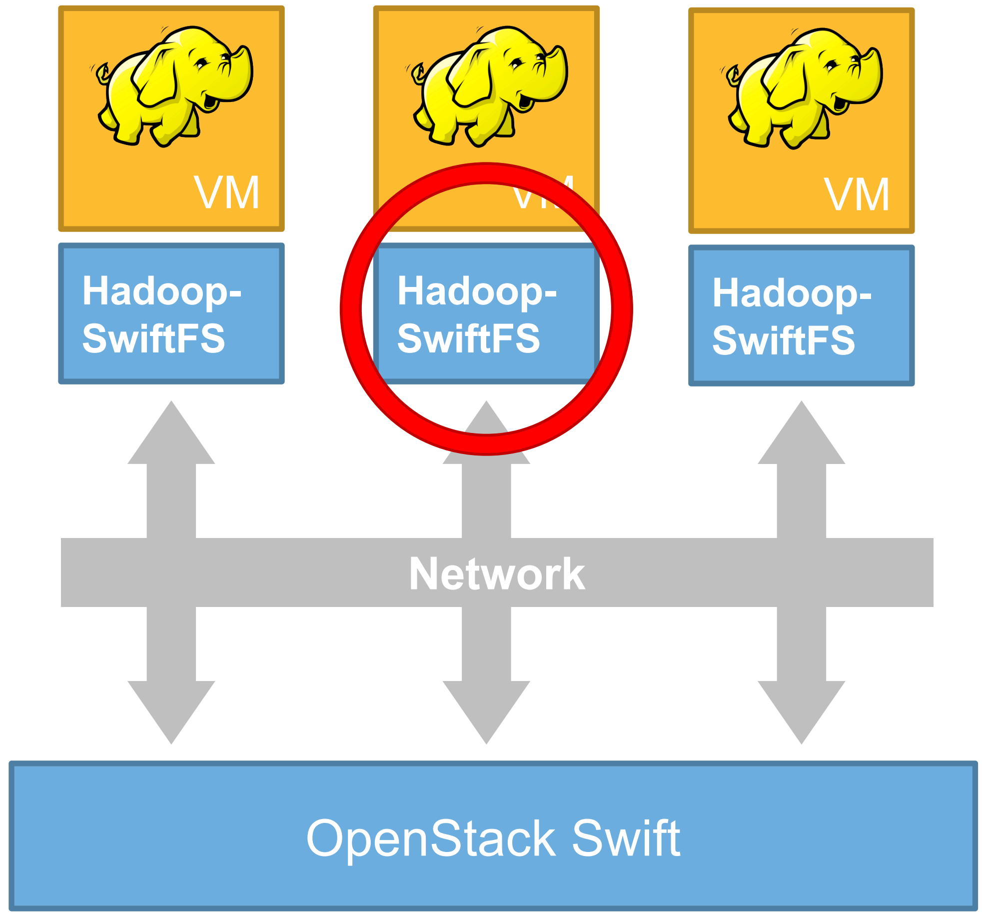
- Contributors and operators of Swift, Ceph, and OpenStack
- Operators of Hadoop-ecosystem* software that uses the Swift API
- Community members from the Hadoop-ecosystem*
  - In particular file system folks
- Potential operators and highly technical users of any of the above

* Any software that can use the Hadoop FileSystem API

Hadoop + Swift 101

- How does Hadoop interact with Swift?
  - Hadoop "SwiftFS" implements Hadoop FileSystem interface on top of OpenStack Swift REST API

- **Content courtesy Comcast at OpenStack Tokyo 2015** https://youtu.be/fu7nmIPsYOo?t=22m17s



Prior and Related Work

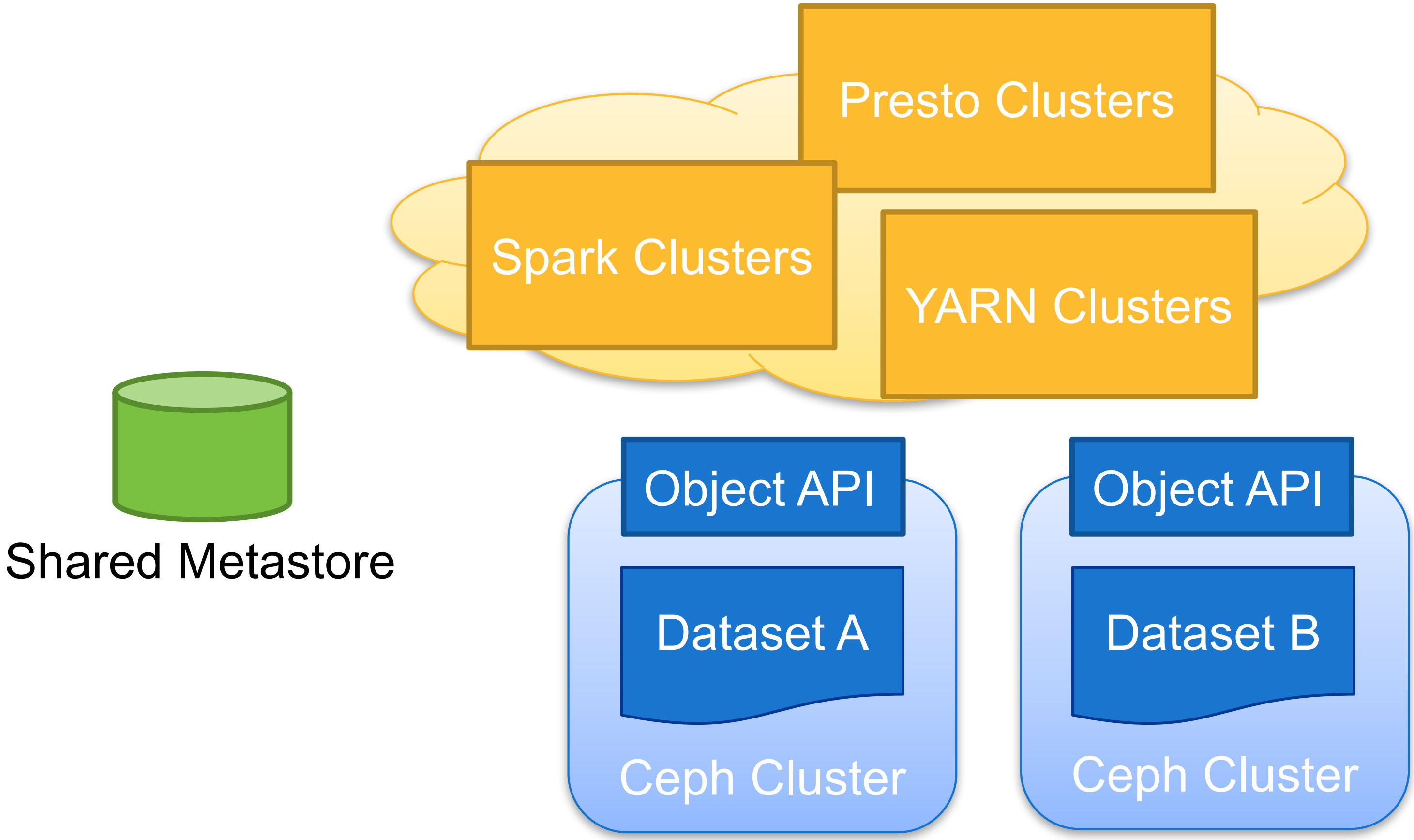
- Sahara-extra Hadoop file system implementation for Swift
  - https://github.com/openstack/sahara-extra

- Hadoop OpenStack (RackSpace, Hortonworks, Mirantis)
  - May be a fork of Sahara-extra implementation?
  - https://issues.apache.org/jira/browse/HADOOP-8545
  - https://github.com/apache/hadoop/tree/trunk/hadoop-tools/hadoop-openstack

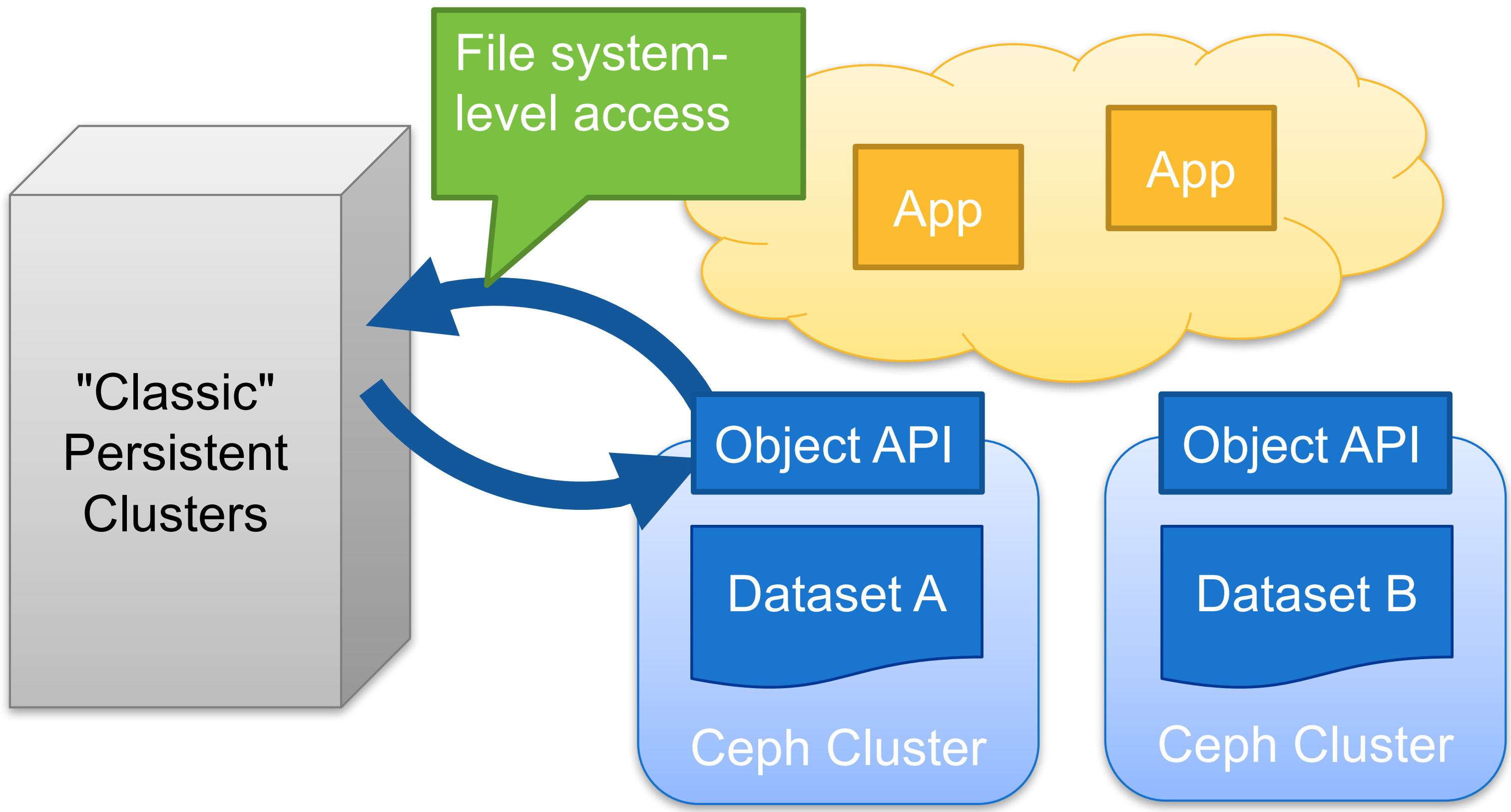
- Comcast
  - Contributions to Sahara-extra implementation
  - https://youtu.be/fu7nmIPsYOo?t=14m33s

Walmart
Save money. Live better.

General Architecture



Extended Architecture



Object Storage APIs in Ceph: Swift and S3

- S3 has broad client-side support
- S3 clients aren't always aware of non-canonical implementations
- General concern around a "closed" standard
- Swift client-side support isn't universal
- Swift support won't get better without adoption
- In theory, performance tweaks can happen faster/better with Swift

Limitations of Sahara-extra driver (patched icehouse branch)

- ORC "range seeks" fail causing job failures
- Uncontrolled number of HTTP connections
  - Jobs effectively DDoS RGWs
- Slow delete/rename/copy operations with high object count
- Large object lists truncate at 10,000 objects
- Re-auth deadlock kills queries from long-running processes (Presto)
- Large object support (>5GB) didn't work for us

Why Swifta

- Spent several months patching existing codebase
- Evolved from experiment evaluating a partial rewrite of Sahara-extra
- To more quickly add performance features to our experimental build
- Name intended to mark our build as an alternate implementation of the Swift driver, avoid confusion with the Sahara-extra reference implementation

Features of Swifta

- **Bounded thread pools** for list, copy, delete, and rename
- Multiple **write policies** adjust local storage and upload behavior
- Re-designed **range seek** support
  - Supports ORC behavior in Hive 2.1+
- **Pagination** for large object lists minimizing memory footprint
- **LRU cache** to minimize number of header calls
- **Lazy seek** optimizes when HTTP requests are made
  - Supports stream behaviors (e.g., in Presto)
- Along with Ceph RGW patch, **resolve Large Object performance penalty**

Dynamic Large Object Support and Associated Challenges

- Couldn't get client-side to split large objects (we were using an old code base)
  - Built upon the existing primitives in Sahara-extra
- Severe performance penalty in a common "pseudo-directory" case
  - Can't identify which subdirectories are actually DLOs
  - Patch in Ceph shows dramatic improvement

Asterisk *

- We have not tested against a Swift "proper" cluster!
- The Swift bulk LIST API does not natively provide an efficient mechanism to flag and provide the size of large objects, unlike S3
  - Large objects appear as directories to a user when listing the parent directory
  - Does not affect STAT call against large object itself
- Severe performance penalty in order to present "correct" hadoop fs -ls results to user
  - We don't currently do this in our "main" Swifta code
  - Causes some Hibench jobs to fail, causes issues with user scripts
- We addressed this with a "hack" of Ceph's Swift implementation, and some client-side code
- Patch to Ceph Swift API server-side implementation holds arbitrary user-provided data
  - https://github.com/ceph/ceph/pull/14592
- Using that field to populate flag for/total size of large objects

Walmart Save money. Live better.

Featured Performance Results

- Bounded thread pools
  - Parallelism where it did not exist or limited *
  - File system operations (delete, rename)

- Write policies
  - File system operations (upload)
  - HiBench WordCount (MR jobs)

* Direct comparisons of Swifta against patched Sahara-extra driver, icehouse branch

Description of Evaluation Parameters

- OpenStack VMs
  - 16 vCPU
  - 52GB memory
  - 500GB **SSD** local volume

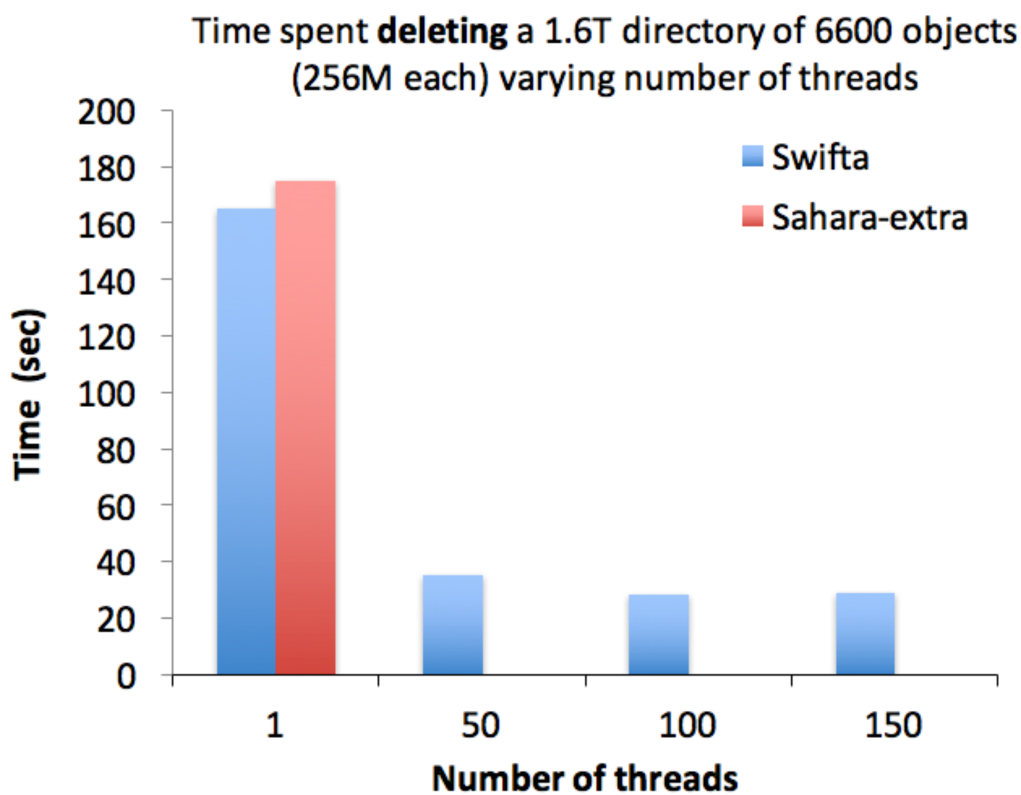
- **HDD** storage clusters
  - Ceph version 10.2.5-28redhat1xenial
  - LVM cache using NVMe and HDD based OSD
  - File based journal
  - Erasure coding, k=8 m=3 for 1.375x overhead
  - 25Gbps NICs, 1x "public", 1x "private"

- Important shared parameters
  - merge/split thresholds: 48/16

Walmart
Save money. Live better.

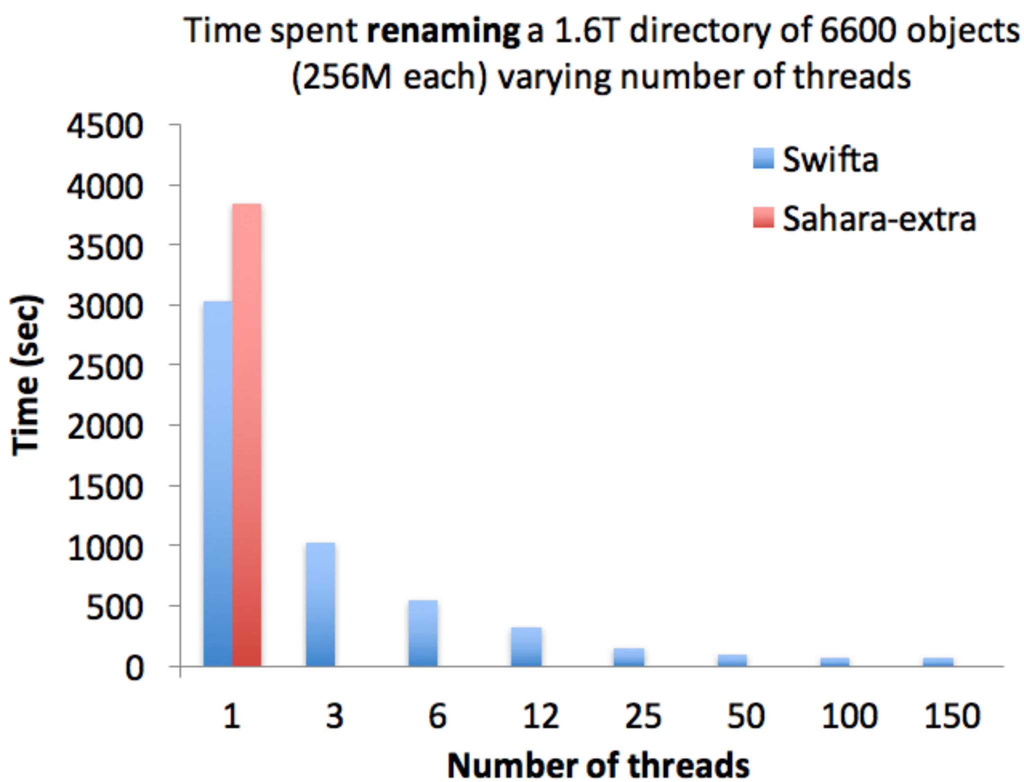
Bounded Thread Pool: Delete

- hadoop fs -rm on a single SSD node
- Thread pools of swifta provides improvement
- Higher thread counts caused Ceph RGW response time to increase



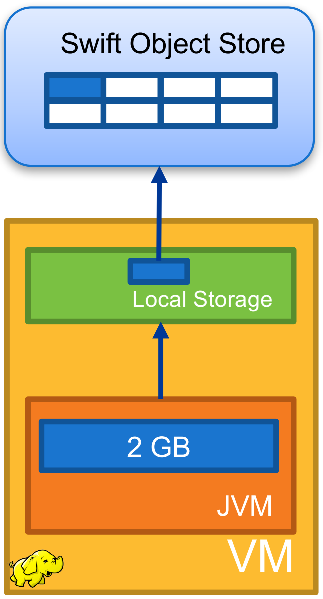
Bounded Thread Pool: Rename

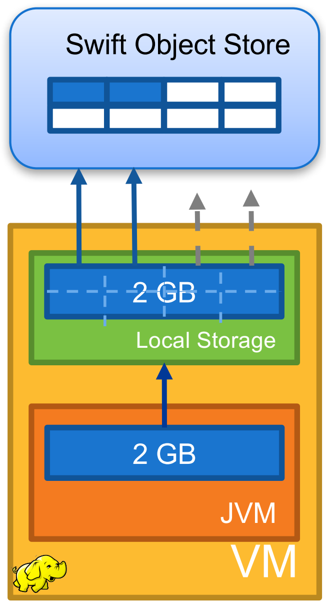
- hadoop fs -mv on a single SSD node
- Thread pools of swifta reduces execution time of rename operations (copy and delete) to trivial levels

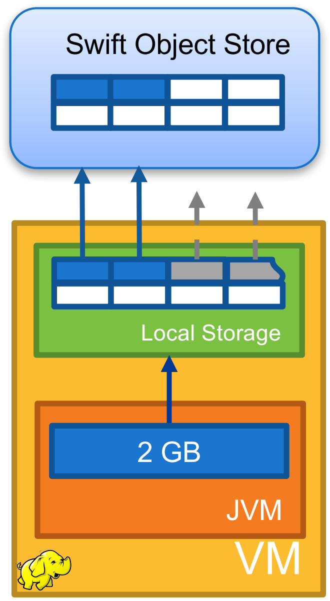


Swifta Write Policies

	Policy: Multipart Single Thread	Policy: Multipart no Split	Policy: Multipart with Split
Local Storage	`split size * 1` For default split size (256MB), max disk use of 256MB	Entire file saved to local storage	`split size * threads`
Upload Threads	Single thread uploading one pre-split object	Many threads uploading objects via local byte ranges in parallel	Many threads uploading pre-split objects asynchronously from local writes



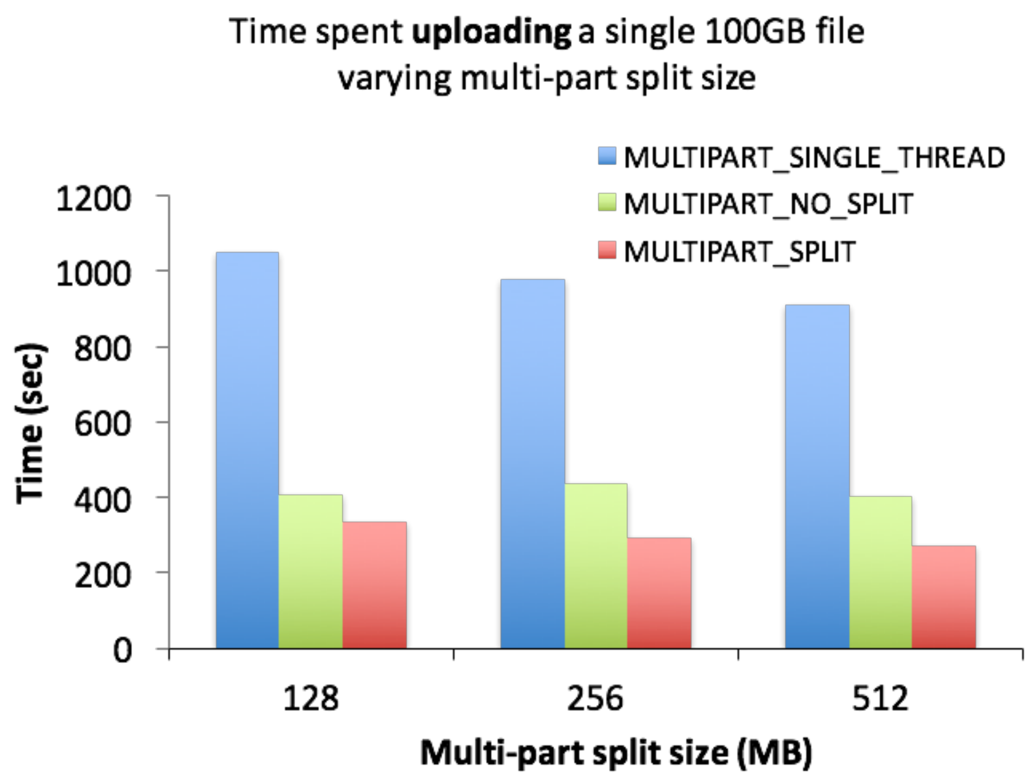




Walmart Save money. Live better.

Write Policy: Performance Comparison of Uploading a Single 100GB File

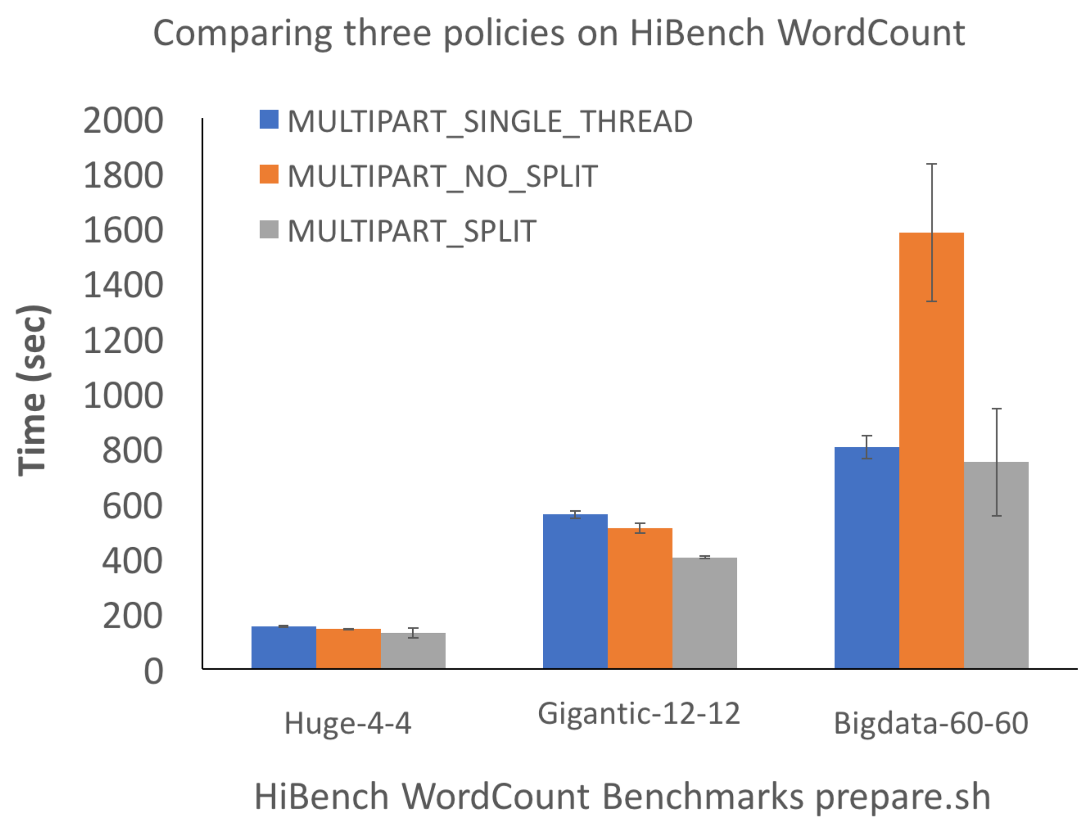
- hadoop fs -put on a single SSD node
- While "Single-Thread-One-Split" is slowest, it requires the least local storage
- "No-Split-Whole-File" policy requires 100GB local storage for this test
- All three policies used 20 threads in swifta thread parameters other than the uploading thread



Walmart Save money. Live better.

Write Policy: Performance Comparisons on HiBench WordCount

- HiBench 6.0 released version, a MR job of WordCount prepare.sh
- Three "scale-# of mappers-# of reducers": Huge-4-4, Gigantic-12-12, and Bigdata-60-60, 4GB memory per mapper/reducer, 10 compute SSD nodes
- Default settings of Swifta thread parameters



Walmart Save money. Live better.

Lazy Seek

- Seek only when necessary to read data
- Reduce connection overheads to input streams (e.g., huge improvements in Presto queries)
- A feature implemented similar to S3A: https://issues.apache.org/jira/browse/HADOOP-12444

Future Work

- Open source after internal workload validation
- Local tiered storage for buffering
- Multiple read policies to improve read performance
- Abstract calls to support both Swift and S3 protocol

Take-away

- Swifta scales the Swift API for large Hadoop-ecosystem workloads
- Prefer to merge our work upstream
- Welcome help to merge, or just make current code better
- Still work to be done in the Swift community, and we would love to help (large object support, in particular)

Swifta

Q&A

Mengmeng Liu (mengmeng.liu@walmartlabs.com)

Andy Robb (arobb@walmartlabs.com)

Ray Zhang (LZhang@walmartlabs.com)