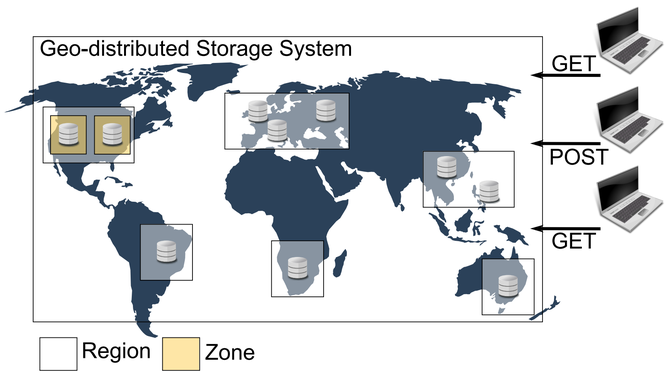
# Big Data Analytics on Object Stores

## A Performance Study

Lukas Rupprecht*,[1] Rui Zhang, Dean Hildebrand
*Imperial College London, IBM Research - Almaden

## Why Object Stores?

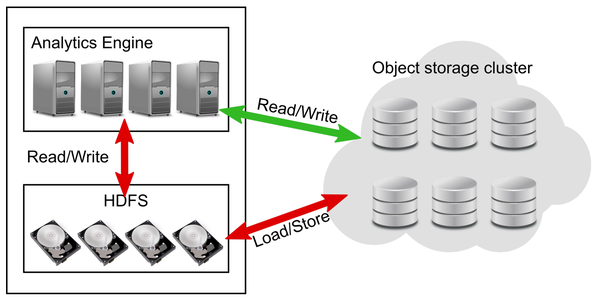



- Object Stores (e.g. OpenStack Swift) are **highly scalable** storage systems to **cheaply store billions of objects.**
- They provide a **RESTful API** and **fine-grained security** via role-based authentication and HTTPS

➤ **Cheap and convenient global sharing/distribution of scientific data.**
➤ **Scalable long-term archive store**

## Why Analytics on Object Stores?

Analytics engines are typically running on top of distributed file systems such as HDFS. If Object Stores are used for archiving, running analytics on archived data (**active archive**) requires an **extra load/store step** and an additional analytics cluster. By **directly** running on the Object Store, the **extra step is avoided**.




## How To Run Them Together Efficiently?

What are **common problems** when running analytics directly on Object Stores and how can they be overcome?

Can we **leverage certain capabilities** of Object Stores?
- Fast file lookups
- Fast metadata access
- Rich metadata

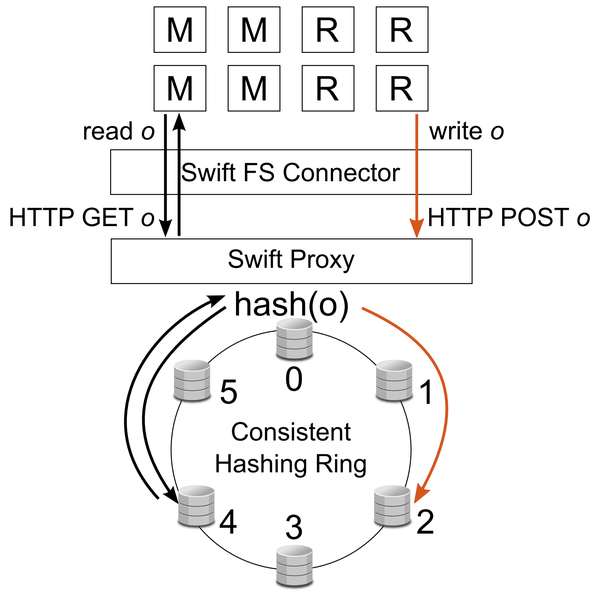
To answer these questions we compare the performance of running Apache Hadoop **MapReduce** on top of **OpenStack Swift** (Object Store) and Apache Hadoop **HDFS** using a recently developed connector [1].

[1] http://docs.openstack.org/developer/sahara/userdoc/hadoop-swift.html

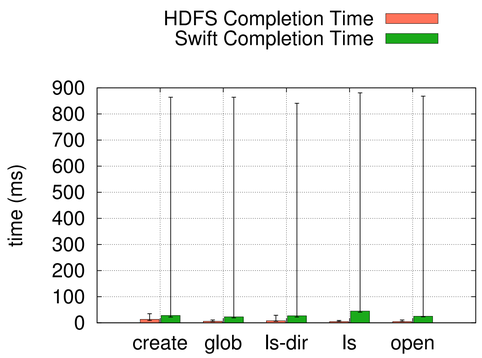
[1] Work done during internship at IBM Research - Almaden

## OpenStack Swift + Hadoop MapReduce

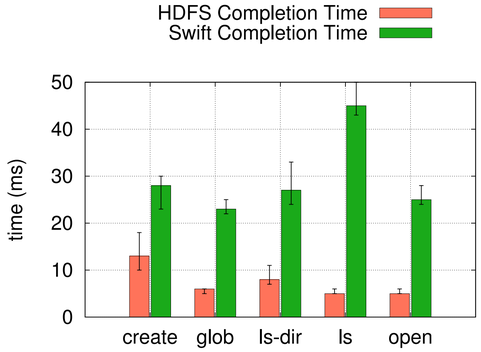
- Connector is Hadoop `FileSystem` implementation
- File System calls are translated to HTTP requests to the Swift Proxy
- Swift Proxy stores/retrieves objects via **name-based consistent hashing**




## File Access Times: Swift vs. HDFS




- Heavy outliers for Swift
- Initial call requires **authentication**
- Overhead of **~1s** (as no authentication for HDFS)

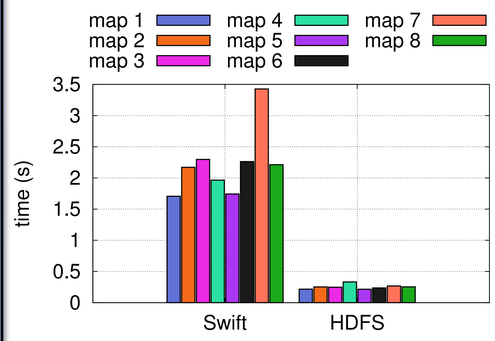



- Without authentication **Swift** is **2-9x slower**
- Reason: HTTP overhead (FS talks to HTTP server, request processing, ...)
- **No benefit** of **constant lookups**

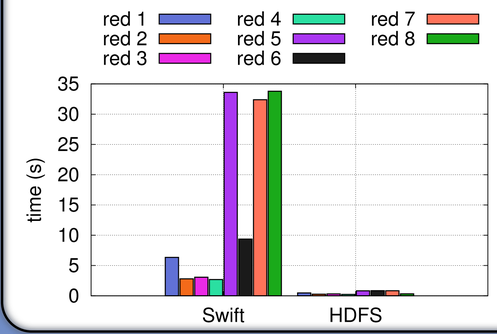
**Authentication and additional HTTP communication** in Swift causes calls to be slower.

## MapReduce: Swift vs. HDFS

- How much time do mappers and reducers spend in storage system calls?
- Setup: Small (3 nodes) private cluster running MapReduce, HDFS, and Swift colocated
- Workload: Batch-oriented wordcount job on 12GB dataset (HiBench generated)

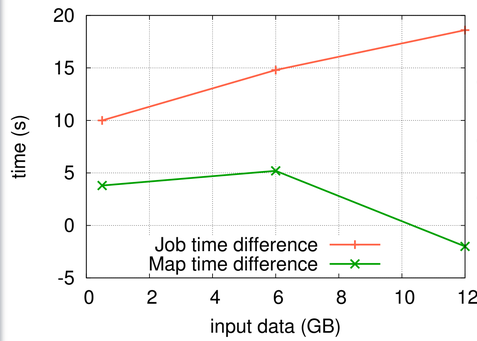



Mappers
- Swift overhead of ~2s, i.e. **~10x slower**
- Overhead due to HTTP + authentication
- **Stable** behavior

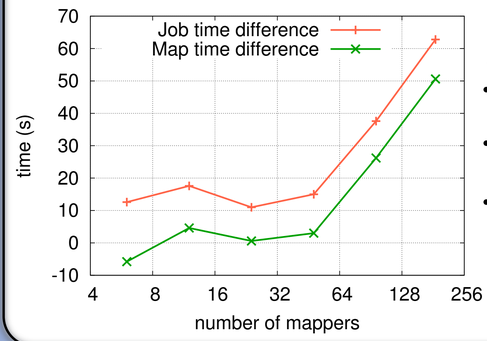



Reducers
- Swift overhead of up to 33s, i.e. **~150x slower**
- Overhead due to **rename**
- **Rename causes (remote) data copy**

## How Does The Swift Overhead Scale?




Scaling Datasize
- **Mapper** overhead stays **constant**
- **Reducer** overhead increases **linearly**
- Larger output files cause longer copies during rename




Scaling Mappers
- **Mapper** overhead increases **linearly**
- **Reducer** overhead stays **constant**
- More mappers cause more authentication requests

## Conclusion and Future Work

We presented a first performance study of MapReduce running on top of Swift and identified problems for such a setup. A major problem is that objects cannot be renamed without a data copy. This can have significant impact on job completion time for large input data and latency sensitive jobs.

In the future, we plan to increase the scale of our setup and look at different workloads, replication, and the implications of object size to data locality.