Integration of cloud-based storage in BES III computing environment

L Wang1, F Hernandez1,2 and Z Deng3

1IHEP computing centre, P.O. Box 918-7, 19B Yuquan Road, Beijing 100049, China
2IN2P3/CNRS computing centre, 43, bd du 11 Novembre 1918, 69622 Villeurbanne Cedex, France
3IHEP experimental physics centre, P.O. Box 918-1, 19B Yuquan Road, Beijing 100049, China

E-mail: Lu.Wang@ihep.ac.cn

Abstract. We present an on-going work that aims to evaluate the suitability of cloud-based storage as a supplement to the Lustre file system for storing experimental data for the BES III physics experiment and as a backend for storing files belonging to individual members of the collaboration. In particular, we discuss our findings regarding the support of cloud-based storage in the software stack of the experiment. We report on our development work that improves the support of CERN’s ROOT data analysis framework and allows efficient remote access to data through several cloud storage protocols. We also present our efforts providing the experiment with efficient command line tools for navigating and interacting with cloud storage-based data repositories both from interactive sessions and grid jobs.

1. Introduction
Object storage systems such as Amazon’s Simple Storage Service (S3) [1] have substantially developed in the last few years. The scalability, durability and elasticity characteristics of those systems make them well suited for a range of use cases where data, in the form of files, is written, seldom updated and frequently read. Storage of images, static web sites and backup systems are examples of the use cases where remote object storage systems have proven effective [2]. In the rest of this paper we use the term cloud storage to refer to object storage systems that expose a well-documented interface on top of standard protocols such as HTTP so that remote clients can interact with the systems both over local or wide area networks.

Generally speaking, experimental physics data are stored as immutable files, which are read several times for the purposes of filtering and analysis according to the experiment-specific data processing workflows. This write-once read-many access pattern seems well suited for cloud storage systems. We present in this paper an on-going work that aims to evaluate the suitability of cloud-based storage as a supplement to the Lustre file system [3] for storing experimental data for the BES III physics experiment [4] and as a backend for storing files belonging to individual members of the collaboration. For this evaluation, we deployed a test bed of OpenStack Swift [5], an open source, community-driven implementation of a cloud storage system used in production by several commercial cloud storage providers.
The rest of the paper is organized as follows. The next section presents the BES III physics experiment and its data processing needs that serve as motivation for this work. We also present in some detail the cloud storage system we used in this evaluation. In section 3 we describe our work for integrating cloud storage protocols into the BES III software stack and in section 4 we present the quantified results of our tests. In the last two sections we discuss our findings and draw some preliminary conclusions.

2. Context
This section provides the context under which this evaluation is performed, covering the physics experiment, its computing environment and the OpenStack Swift test bed deployed for this evaluation.

2.1. BES III experiment
The BESIII experiment at the Institute of High Energy Physics (IHEP) of the Chinese Academy of Sciences (Beijing, China) uses the high-luminosity BEPCII double-ring electron-positron collider to study physics in the tau-charm energy region around 3.7 GeV [6]. From the start of data acquisition in 2009 to the end of 2011, BEPCII produced the world's largest samples of $J/\psi$ and $\psi'$ events, about 0.2 billion and 0.1 billion events respectively, giving a total annual raw data volume of about 200 TB. Another 1 billion $J/\psi$ events were produced in 2012, approximately 5 times the size of the previous dataset. Simulation, calibration, reconstruction, and analysis use BOSS (BESIII Offline Software System), a software framework based on ROOT [7] among other components.

2.2. BES III computing environment
The computing environment for the experiment is composed of about 10 geographically distributed sites organized around IHEP computing centre, which plays the central role of data repository. Currently, IHEP computing centre provides BES III about 4500 CPU cores, 3PB of disk storage and 4PB of tape storage. It operates Lustre distributed file system for storage management and a customized version of CERN’s Castor [8] for tape storage. A substantial increase of resources is required for the near future in terms of both computing and storage capacity, to reach about 10,000 CPU cores, 5PB of disk and 10PB of tape storage.

Storage is the single most sensitive factor of high-energy physics computing performance. The growth of BES III data storage requirements exposed some limitations of the deployed system. First, the Lustre file system is not suitable for serving lots of small files to interactive users. Second, it requires specialized people to maintain a stable and reliable storage infrastructure, which is prohibitive for several of the small sites participating in the experiment’s distributed computing environment. Finally, the ability to share experimental data between IHEP and external sites is a requirement that is not easily fulfilled by Lustre itself.

By its inherent extensibility, accessibility and ease of operation cloud storage seems a good candidate to complement the current storage infrastructure of the experiment, which is the main motivation of this evaluation work.

2.3. OpenStack Swift test bed
In order to evaluate the potential of a Swift-based solution, we deployed at IHEP a dedicated test bed. Shown in figure 1, the test bed is composed of 2 head nodes and 4 storage nodes, all of them with 24 CPU cores and 24 GB RAM. Each head node has a single 10Gbps NIC and each storage node has a single 1Gbps NIC. The head nodes’ network connection is used for both external and internal traffic, that is, for communicating with both the clients of the test bed (machines in the login and batch farms) and with the storage nodes. Each storage node has three 2TB SATA disks. DNS round robin was used for balancing the load of the two head nodes, where Swift’s proxy server runs.

As a baseline for this evaluation, throughput and I/O operations per second (IOPS) of a single XFS-formatted SATA disk were measured using IOzone [9] and mdtest [10], respectively. These
measurements allow us to compute the maximum aggregated read throughput of our test bed to be 480MB/sec.

Regarding the software components, we used OpenStack Swift v1.7.4 on Scientific Linux v6. Swift was configured with a replication factor of 3, each disk of a storage node belonging to a separate zone. The optional ‘swift3’ module [11] was enabled in order for the test bed to expose an Amazon S3-compatible REST API in addition to the native Swift one. This was intended to compare different access protocols to the Swift storage system, as shown later in the paper.

Compared to Lustre, our operational experience with OpenStack Swift is very limited so it is too early to draw definitive conclusions. Nevertheless it appears that configuring the hardware and software components for a Swift cluster requires less expertise than deploying a comparable Lustre-based solution.

3. Enabling BES III to exploit cloud storage

Enabling the software stack of a physics experiment to exploit cloud storage encompasses several aspects. First, the experiment software must be able to interact with cloud storage systems to read files (or fragments of files) and to store new files in the system. Second, it is important for the experiment members to have tools to navigate the storage space, either by a human or by automated tools. We present in this section our work toward reaching these two goals.

3.1. Extending ROOT for supporting cloud storage protocols

BES III experimental data is currently stored in Lustre in the form of ROOT-formatted files. Jobs executing in the batch farm of IHEP computing centre, where some of the data processing workflows of the experiment are performed, access their input data via the POSIX-compliant file system interface exposed by the Lustre client. The experiment’s software framework builds on top of the input/output capabilities of ROOT.

Since version 5.34.05, released on February 2013, ROOT consolidated its built-in support for reading files hosted by any cloud storage provider that exposes Amazon’s S3 REST API. However, there is no support for OpenStack Swift native protocol included in ROOT. In addition to this constraint, the BES III experiment cannot easily upgrade its software framework to a recent version of ROOT: it requires v5.24, released on October 2009. So, in order to perform this evaluation using a real use case while satisfying all the constraints of a running physics experiment, it was necessary to find ways to use BES III software extended with cloud storage access capabilities, without requiring modifications to the experiment’s software framework.

For the purposes of this exploratory work, we developed an extension to ROOT which implements several protocols currently used for cloud storage, namely Amazon’s S3 and OpenStack’s Swift. This
extension can be used with any version of ROOT since v5.24 up to the current recommended production version, covering all versions released in the last 4 years.

This experiment-neutral extension does not require any modification neither of ROOT nor the experiment-specific software built on top of it. It is composed of a lightweight shared library (~500KB) and a plugin deployable on top of an existing ROOT installation by a non-privileged user. It enables its users to read data files stored on commercial cloud storage services such as Amazon S3 and Google Storage, on storage services powered by OpenStack’s Swift (such as Rackspace) and on storage appliances such as Huawei’s UDS. Source code and documentation are publicly available [12].

We leveraged this software to extend the capabilities of the legacy version of ROOT used by BES III without any modification of the experiment’s software framework. We were able to run unmodified jobs the experiment routinely runs for physics analysis: the jobs were configured to read files stored on our Swift test bed instead of those stored in Lustre, as they usually do. In the next section we present quantitative results of our observations.

3.2. Command line interface to cloud storage systems

OpenStack Swift comes with a Python-based command line interface for interacting with the system [13]. This interface allows operations such as create containers, retrieve the contents of containers, upload, download and delete objects. It can be used both from an interactive session and from a script.

In the framework of this evaluation, we also developed a prototype of a command-line client for interacting with an OpenStack Swift backend. It is implemented in the GO programming language [14] and leverages some features suitable for this use case, such as inherent concurrency and the ability to create a stand-alone executable which can be easily distributed and installed.

This tool, which we plan to release the source code of in the near future, was used for uploading the set of BES III data files we used for this evaluation and for some of the performance tests we conducted.

3.3. File system-based interface to cloud storage

From the end-user point of view, a file-system interface to cloud storage is a desirable property. It allows for easy navigation of the storage space and for using standard tools in Unix-based systems (cp, ls, tar, etc.) for manipulating files.

For this evaluation we have explored S3fs [15], a FUSE-based implementation of a file system for S3-compatible storage back ends. S3fs allows for a Linux machine to mount a single Amazon S3 bucket and expose its contents as a file system. With some modification on the format of the HTTP requests it issues, it is also possible to mount a Swift container through S3fs.

Cloud-based object storage combined with traditional file-system interface seems a well suited mechanism for serving lots of relatively small files, such as the ones found in the home directory of individual users, in software repositories for users or experiments and in individual users’ backup areas. LHC experiments routinely use a variant of this model for distributing read-only copies of their software through CernVM-FS [16] over hundreds of sites.

However, some limitations in S3fs makes it not ready for our purposes in its current status. First, it relies on some feature of S3 exposed by Amazon, namely, the possibility to use a bucket name as a virtual host name. Second, S3fs allows only a single bucket to be exposed as a file system.

In the next section we present some quantitative results of our utilization of S3fs in a realistic use case.

4. Evaluation results

As introduced in the previous sections, in order to quantify the viability of Swift as a storage backend for physics and individual user’s files, we have performed several types of tests. The purposes of those tests were to measure the capacity of the system to update its metadata, to deliver metadata information and to quantify the throughput of the system when storing and delivering data.
In this section we present the results of those tests and discuss some of the issues we observed in the process.

4.1. Performance tests with small files
One of the key characteristics of a storage system is its capacity to update and deliver file metadata. Updating metadata in Swift occurs, for instance, when a new account, new container or new file is created in the system. Delivering metadata is necessary for responding to requests such as retrieving the list of existing containers in an account or the list of objects stored in a container.

We performed a series of tests to measure the capacity of our Swift test bed to update and deliver file metadata. All those tests involve a predefined set of 100 user accounts, each one with 20 containers. The upload test consisted of simultaneously creating zero-bytes files in those containers, as requested by five machines located in IHEP’s production login farm simultaneously emitting requests. The metadata query test was intended to measure the capacity of Swift to deliver metadata. The client machines emitted simultaneous metadata retrieval requests of randomly selected files within randomly selected containers. Each request retrieved the file’s creation date, file size, file content’s hash and extended attributes. Both tests were intended to simulate the aggregated metadata-related activity generated by 100 individual users simultaneously using Swift for storing small files.

The combined results of these two tests are presented in figure 2. It shows the number of operations per second successfully performed by our test bed both for updating and delivering metadata, versus the number of simultaneous requests. It is worth noting that each test was performed separately and the Swift test bed was always devoted to the test. We can observe that there is a noticeable difference in performance between metadata update and retrieval. While the number of metadata update operations does not significantly increase while increasing the number of simultaneous requests, the number of query operations does increase up to a threshold of 500 simultaneous requests. After that, the number of operations decreases as Swift responds with ‘Internal server errors’ to a significant fraction of the retrieval requests.

A separate test was designed to measure the capacity of our Swift test bed to store and deliver small files, i.e. files of 5MB each, such as the ones stored by individuals in their home directory. The results are presented in figure 3. In this test we measure the throughput of the system separately for both types of operations. The download operation reaches a peak of 450 MB/sec when 100 simultaneous requests are issued; this limitation is explained by the aggregated network capacity of the 4 storage nodes of our Swift test bed which each has a single 1Gbps network interface card. As a comparison point, the software system used at IHEP computing center for storing individual users’ files, namely AFS, delivers in production 30MB/sec on average and ingests a few MB/sec.

Figure 2. Metadata performance test: number of read and write operations of zero-bytes files using 100 user accounts, 20 containers per account. Swift native protocol over HTTP was used in this test.

Figure 3. Observed throughput while uploading and downloading 5MB files using 100 accounts, 20 buckets per account. Swift native protocol over HTTP was used in this test.
4.2. Performance tests with experimental physics data files

To validate the possibility of running BES III data processing activity supported by Swift, an unmodified typical \( J/\psi \) analysis job was chosen. The I/O activity of the job, as profiled by SystemTap [17], shows that it generates large-sized read requests (between 1MB and 2MB each) and small-sized write requests (less than 128KB). About half of the I/O operations are followed by file seeks. The ratio of data read to data written by the job is about 100:1. Read performance is therefore a key factor for reaching good job efficiency for this kind of jobs.

4.2.1. Efficiency of access protocols. Swift exposes its own native protocol and can be configured to also expose S3. Those two protocols can be used on top of either HTTP or HTTPS, although the recommended deployment configuration is to use HTTPS, for security reasons.

We measured the wall clock time needed for the BES III job (running in the local batch farm) to process the same data served by Lustre and by Swift across the local area network. In the case of data stored on Swift, we measured the time when using both native Swift and S3 access protocols, on top of both HTTP and HTTPS. In addition, we tested accessing the same data when a Swift container is mounted and exposed as a file system by S3fs. The results of those tests are presented in figure 4.

Relative to Lustre, the overhead of accessing data using native Swift or S3 protocols over HTTP is low; this holds when the data is accessed either by the job itself using the ROOT extension or through the file system interface exposed by S3fs. However, a noticeable performance penalty is measured when using HTTPS, likely due to the encryption and decryption phases inherent to this protocol. When using Swift at the level of a local area network, care must be taken when deciding if the increased security risk associated to not using HTTPS is worth the performance gain.

\[ \text{Figure 4. Efficiency of data access protocols for reading data stored in Swift, relative to Lustre.} \]

4.2.2. Throughput tests. Figure 5 shows the aggregated throughput reached when a set of simultaneous BES III analysis jobs were executed at IHEP’s local batch farm, reading data stored in Swift both through its native protocol and through S3. It can be seen that the aggregate throughput grows linearly when a small number of simultaneous jobs are executed, until the threshold of 128 simultaneous jobs using Swift native protocol is reached (64 when using S3 protocol). A peak throughput performance of 410MB/sec is then reached, which corresponds to 85% of the theoretical maximum throughput our test bed can deliver. When using S3 protocol, peak performance is reached at 280MB/sec, significantly lower than when using Swift native protocol.

Figure 6 shows the average CPU efficiency, that is the ratio CPU time vs. wall clock time, when the jobs use native Swift or S3 protocol. Our test bed can feed up to 128 jobs while keeping them above the 80% efficiency level. However, for reaching the same efficiency for jobs reading their input files through the S3 protocol only 32 jobs can be executed simultaneously.
Therefore, better performance can be reached when using Swift native protocol. The overhead observed with S3 protocol may be explained by the necessary protocol translation done at the head nodes by the ‘swift3’ gateway.

5. Discussion
This evaluation work led us to consider Swift as a suitable candidate for supplementing Lustre file system for BES III computing for the use cases presented below.

5.1.1. Storage back end for small files
A Swift storage back end seems well suited as a repository of files for individual users and groups. The typical file size for this particular use case is a few MB up to 100MB, they are mostly read-only and the access patterns are such that a high-throughput system is not required. Most of the time, those files are accessed in the context of a local area network but the Swift storage cluster could be also exposed to the wide area network, making the user’s files accessible from every connected device.

However, not all the software tools currently available are satisfactory for a generalized deployment involving end-users. Although there are commercial and free tools for mounting S3- or Swift-based containers and expose them as a file system at the user’s personal computer, they are not at the desired level either of price or reliability. In addition, command line tools for interacting with the user’s storage area powered by Swift exist with several degrees of usability. We intend to continue our work and make a contribution to improve the current situation.

5.1.2. Physics data storage back end for small sites
The vast majority of the data processing sites contributing to the BES III experiment are characterized by their relatively small size and limited available manpower to manage their computing infrastructure. Solutions implementing cloud storage such as Swift, which provide reliability by redundancy using low-cost hardware without requiring too much operations and management effort, seem well suited for those sites. With our contribution, no modification to the BES III software is required for the experiment for immediately leveraging private (or commercial) cloud storage, so participating sites may consider this option in their future storage strategy.

5.1.3. Data sharing among the sites participating in the experiment
IHEP computing centre plays the role of central data repository for the BES III experiment. Simulation data produced at external sites needs to be transported to IHEP for permanent storage. Conversely, datasets need to be transferred from IHEP to external sites for local analysis. Cloud storage systems could be used as a way for the participating sites to expose their data to their peer sites. Interested sites can then select their data sets of interest and download them directly over well-known and supported
protocols, such as the ones exposed by Swift. Tools for interactively browsing the datasets available at a site could also be used to make the remote data more easily accessible by human users.

6. Conclusions
We have presented in this paper our results of evaluating OpenStack Swift as a storage system for supplementing traditional file systems such as Lustre, for some well-identified use cases.

We demonstrated that it is possible for physics experiments that build their I/O software stack on top of ROOT, to immediately benefit from cloud storage technologies without modifying their software framework. The source of the software developed for this evaluation is openly available and is independent on any experiment software.

We also demonstrated the high potential of cloud storage technologies to be used as the main backend for repositories of files for individual users or group of users. We showed in addition that a cloud-based solution is attractive for small sites with limited specialized manpower both for storing physics data but also as a mechanism for sharing data with external sites participating in the same experiment.

It is our intention to go beyond this evaluation work and proceed to validate our findings in a production-like environment involving data access across wide area networks. Leveraging our previous work in this field [18], we aim to contribute improving the situation relative to the relative lack of tools for making cloud storage accessible to the end-users for their daily work.

Acknowledgments
We warmly thank Qiumei MA of the BES III collaboration for her contribution to this work.

References
[1] Amazon Simple Storage Service (Amazon S3), http://aws.amazon.com/s3
[2] Amazon S3 – Two Trillion Objects, 1.1 Million Requests/Second, Amazon Web Services Blog, http://aws.typepad.com/aws/2013/04/amazon-s3-two-trillion-objects-11-million-requests-second.html
[3] Lustre File System, http://www.lustre.org
[4] M. Ablikim et al., 2010, 614: 345-399, Nuclear Instruments and Methods in Physics Research A, Design and Construction of the BESIII Detector
[5] OpenStack, http://www.openstack.org
[6] Nicholson C, Lin L, Deng Z Y, et al. 2012 C J. Phys.: Conf. Ser. 396(3): 032078, File and metadata management for BESIII distributed computing
[7] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP’96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch
[8] J Baud et al., 2003, CASTOR status and evolution, http://arxiv.org/abs/cs/0305047v1
[9] IOZone filesystem benchmark, http://www.iozone.org
[10] mdtest, http://www.nersc.gov/systems/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/mdtest/
[11] Swift3 module for OpenStack Swift, https://github.com/fujita/swift3
[12] Cloud storage extension for ROOT data analysis framework, https://github.com/airnandez/root-cloud
[13] OpenStack End User Guide, http://docs.openstack.org/user-guide/content
[14] The GO programming language, http://golang.org
[15] S3fs, FUSE-based file system backed by Amazon S3, https://code.google.com/p/S3fs
[16] J Blomer et al.; 2011 J. Phys.: Conf. Ser. 331 042003, Distributing LHC application software and conditions databases using the CernVM file system
[17] SystemTap, https://sourceware.org/systemtap/wiki
[18] F Hernandez et al 2012 J. Phys.: Conf. Ser. 396 032056, Mucura: your personal file repository in the cloud