Extension of S3 REST API for Providing QoS Support in Cloud Storage

Yusuke Tanimura[1] and Seiya Yanagita[1,2]
[1] *National Institute of Advanced Industrial Science and Technology (AIST), Japan*
[2] *SURIGIKEN Co., Ltd.*
yusuke.tanimura@aist.go.jp, yanagita@suri.co.jp

1 Introduction

In the IaaS (Infrastructure as a Service) cloud environment where virtual machines are deployed, a scalable and reliable storage service is indispensable for hosting virtual machine images, backups, large volumes of application data, etc. Amazon S3 (Simple Storage Service) [3] is one of the storage services to meet such demands for Amazon EC2 [2] users. Swift [9] takes the similar role in OpenStack [7] and provides the S3 compatible interface in addition to its own interface. Although the S3 implementation considers scalability, there is a concern for performance instability when many applications heavily access S3, due to the characteristics of shared resources. Replication and load balancing mitigate the problem but they might not respond every individual request of the applications. Amazon EBS [1] which delivers dedicated performance to the application would not be reasonable for storing large data, in terms of the financial cost and the limitation of shared use. It would be useful that users store large data in S3 for a long term, with low cost, and only when they request with more payment, they can access the data with a certain level of performance.

This paper presents an approach which allows users to have specific I/O throughput in use of S3, by making an advance performance reservation. The reservation is intended to be used as a user's explicit request to S3 in the above. To provide this feature, a storage system which supports the performance reservation is used at the S3 backend, and I/O operations of S3 REST API are extended to be conformed to the reservation-based access to the storage system. Both S3 client and server prototypes are implemented and then evaluated to examine the QoS capability at the S3 data transfer, under the situation where multiple S3 clients access the same resources.

2 Design

In this study, Papio is used at the S3 backend for providing the QoS-enabled I/O function through the S3 server. Papio is an object-store type storage that supports parallel I/O and the performance reservation functionality [10]. The 'bucket/object' semantics of Papio is similar to the S3 one. However, when a bucket is created on Papio, an amount of disk space that a user requested is reserved for the bucket. Then the user can reserve I/O performance with desired throughput (e.g. MB/sec), access type (write or read), and access time (from start to end), for the bucket to write, or for an object in the bucket to read. The reservation ID is issued by Papio when the reservation is accepted. The user's program as a Papio client can access the bucket or the object with the reservation ID. Internally Papio allocates storage resources according to the reservation, and controls I/O throughput of storage devices and the storage network to provide the requested throughput to the Papio client.

The extension of S3 REST API to use the Papio's reservation functionality is designed as follows. First '*PUT Bucket*' of the S3 operation, for creating a new bucket, is extended for the space reservation. Optional parameters for specifying a space requirement; size, lifetime, write and read throughputs at access time, etc. are added in the request element of '*PUT Bucket*.' Second '*PUT Object*,' '*GET Object*' and '*Initiate Multipart Upload*' for writing or reading data are extended for the reservation-based access. The optional parameter named '*X-Papio-Access-ID*' for setting a previously obtained reservation ID is added in the request header of those S3 operations. When the parameters are set in the client request, the S3 server uses them at the access to Papio. Besides, when they are not set, the S3 server automatically makes a reservation to Papio, with a default minimum requirement given by the S3 administrator. Since the automatic performance reservation is made just before the access, if resources are not available to meet the default requirement, the access will be an error. However, this design allows the S3 server to process both original S3 and extended requests. The performance of the original S3 operations are also controlled by Papio.

3 Implementation

A set of client and server implementation to support the extended operations is called PapioS3. The PapioS3

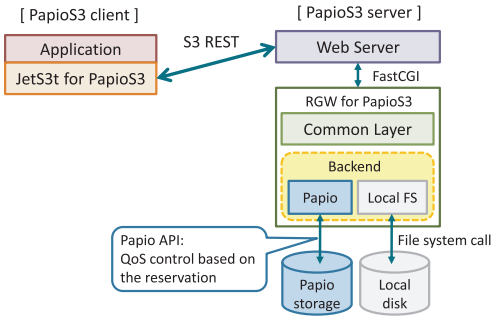

Figure 1: PapioS3 overview

Figure 2: Performance of concurrent writes

server has been implemented based on RADOS Gateway (RGW) [8]. As shown in Figure 1, RGW runs as a CGI program and interacts with the Web server via FastCGI. The PapioS3 client has been implemented with JetS3t [6], which is modified to support the S3 extension and improve multipart uploads and downloads. Support of the multipart operations is important in S3 because a single stream of '*PUT Object*' and '*GET Object*' does not achieve enough performance between S3's client and server even when backend I/O resources are available. Note that the performance is neither reserved nor controlled between the S3's client and server. In the multipart operations, as data are transferred in parallel streams, a single S3 client can have high performance.

4 Performance Evaluation

The performance of the prototype was evaluated on the situation where 5 clients accessed the same S3 server with different performance requirements. In this experiment, the S3 server had only one storage server of Papio and therefore Papio needed to control I/O throughput of the storage server to satisfy the demand of each client. Figure 2 and 3 show the measured performance of the concurrent operations for write and read respectively. Available throughput of the storage server was shared properly and every client achieved slightly higher rate than the requested rate.

Figure 3: Performance of concurrent reads

5 Conclusion and Future Work

Extended I/O operations of S3 REST API have successfully provided the QoS capability to respond the individual user's request. However, to examine the effect of PapioS3 in the production environment, further evaluation in various concurrent access cases should be conducted on a larger environment. In addition, the performance reservation is not supported in the S3 interface. At present, the reservation can be made by the command-line tool included in Papio or another Web-services based protocol [5]. Integration of the reservation into the cloud storage management through S3, CDMI [4], etc., and co-allocation with other types of cloud resources, for example the network resource between S3's client and server, are a future work.

Acknowledgment

A part of this work was supported by KAKENHI (23680004).

References

[1] Amazon EBS. http://aws.amazon.com/ebs/.
[2] Amazon EC2. http://aws.amazon.com/ec2/.
[3] Amazon S3. http://aws.amazon.com/s3/.
[4] CDMI (Cloud Data Management Interface). http://www.snia.org/cdmi.
[5] GNS-WSI version 3. http://www.g-lambda.net/.
[6] JetS3t. http://jets3t.s3.amazonaws.com/index.html.
[7] OpenStack. http://www.openstack.org/.
[8] RADOS Gateway. http://ceph.com/docs/master/radosgw/.
[9] Swift. http://swift.openstack.org/.
[10] TANIMURA, Y., KOIE, H., KUDOH, T., KOJIMA, I., AND TANAKA, Y. A Distributed Storage System Allowing Application Users to Reserve I/O Performance in Advance for Achieving SLA. In *Proceedings of the 11th ACM/IEEE International Conference on Grid Computing* (2010), pp. 193–200.