COSBench: A Benchmark Tool for Cloud Object Storage Services

Qing Zheng*, Haopeng Chen
Reliable Intelligent and Scalable Systems Group
School of Software, Shanghai Jiao Tong University
Shanghai, China
{mark, chen-hp} @ sjtu.edu.cn

Yaguang Wang, Jiangang Duan, Zhiteng Huang
Intel Asia-Pacific Research and Development Ltd.
Shanghai, China
{yaguang.wang, jiangang.duan,
zhiteng.huang} @ intel.com

***Abstract*- With object storage services becoming increasingly accepted as replacements for traditional file or block systems, it is important to effectively measure the performance of these services. Thus people can compare different solutions or tune their systems for better performance. However, little has been reported on this specific topic as yet. To address this problem, we present COSBench (Cloud Object Storage Benchmark), a benchmark tool that we are currently working on in Intel for cloud object storage services. In addition, in this paper, we also share the results of the experiments we have performed so far.**

Keywords-component; benchmark; object storage; workload; performance evaluation

I. INTRODUCTION

According to IDC [1], the total amount of digital data worldwide will increase by 48% from last year and reach 2.7 zettabytes by 2012. Most of them are unstructured data such as images, videos, and documents. The tremendous amount of these data as well as the unprecedented growth rate poses a significant challenge on enterprise storage infrastructures. Today, many of them are moving from conventional SAN and NAS based systems to object storage services. They either rely on public services, such as S3 [2], Cloud Storage [3], and Cloud Files [4], or manage to build their own private clouds with the help of open source or proprietary solutions such as Walrus [5], Swift [6], and Haystack [7].

Object storage services provide RESTful interfaces for one to store and access files in a way that is similar to albeit simpler than regular file systems. In addition, these services are often characterized by what is lacking in traditional technologies: scalability, cost-effectiveness, and easy-of-use, if not high-performance and availability. For people who are responsible for providing high-quality object storage services, performance benchmarking can be of great importance and usefulness. For example, cloud builders may use benchmark tools to compare different software realizations or hardware configurations. Moreover, these tools can also help them conduct system refactoring or algorithm tuning in order to achieve optimal performance. However, to the best of our knowledge, there has been little, if not none, reported study focusing on benchmark technologies for object storage services. To help address this problem, we are investigating related methodologies with the goal of creating a benchmark tool for characterizing object storage services, thus allowing people to evaluate various implementations or configurations of object storage service.

* The work is performed in Intel, where the author is an intern.

There are other storage solutions such as SQL databases or novel key-value stores. Object stores are different in that they are designed for unstructured data rather than structured. In addition, unlike file or block systems which also handle unstructured data, object stores expose distinct interfaces that are object based and resource oriented.

Unfortunately, performance benchmarking against object storage services is not as straightforward as it may appear to be due to several reasons. To start with, there is currently no widely-adopted standard on the interfaces of object storage services, making it challenging for a benchmark tool to work with different service implementations. Second, a benchmark tool should be able to simulate diverse usage patterns, which can also be challenging as it requires a judicious abstraction of real-world workloads. Finally, it is challenging to design a tool that is simple, practical, and extensible simultaneously.

In this paper, we present our work on COSBench (Cloud Object Storage Benchmark), which is a tool that we are designing and implementing in Intel for benchmarking cloud object storage services, and is also our current answer to the challenges we listed above.

TABLE I. STORAGE INTERFACE

Operation	Sample RESTful Req.	Sample Res. Code
create an object	PUT /container/object	201
get an object	GET /container/object	200
delete an object	DELETE /container/object	204

II. BENCHMARK DESIGN

In object storage services, one creates containers and put objects into these containers for storage. Containers are just like directories except there is no sub-container. Objects are regular files though with limitations. For instance, objects cannot be locked as files can under POSIX file systems. The storage interfaces are protocols for operating containers and objects. Currently we only have 3 core operations defined in our interface, which is summarized in Table I. Tiny as it is, it still makes the benchmark tool quite practical, since the 3 operations are enough, or at least adequate, for one to take on tasks such as bottleneck locating and capacity measuring. That being said, we are still actively working on adding more operations, but only in a way that will keep the tool simple and universal. To achieve the latter, we are investigating the specification of CDMI (Cloud Data Management Interface) [8], extracting common operations shared among a variety of object storage services and then adding them into our storage interface. On the other hand, in order for the tool to work with real services, we employ adaptors to map our storage interface to theirs. The tool now supports S3 [2] and Cloud

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.52

Files [4] through the SDKs Amazon and Rackspace provide respectively. More adaptors will be added in the future.

To simulate diverse usage patterns, we create different workloads from workload models defined upon our storage interface. Our current workload model can be configured in terms of concurrency pattern, access pattern, usage limitation and others. The details are listed in Table II with examples. The container range is a numeric range of the names of the containers that will be used. The operation count and running time specify the max number of operations issued or the max period of time passed before a workload terminates. Finally, when marked as unprepared, a workload will put randomly generated data into an object store before it gets stressed by the workload. We are still investigating new attributes for the model so as to accommodate more complex usage patterns.

TABLE II. WORKLOAD MODEL

Workload Attribute		Examples
concurrency pattern	worker number	32
	container range	1-20
access pattern	object size	64KB
	read/write ratio	READ 80%; WRITE 20%
usage limitation	operation count	100
	running time	60s
miscellaneous	preparation	False

TABLE III. PERFORMANCE METRICS

Item	Description
response time	duration between operation initiation and completion
throughput	total number of operations performed per second
bandwidth	total amount of data transferred per second

III. CURRENT IMPLEMENTATION

COSBench now has two components, namely controller and driver, and can operate in two different modes, either independent or managed. In independent mode, only driver is used. At runtime, it loads configurations and spawns agent threads which stress the target service in a way consistent with the user-defined usage pattern. Under managed mode, on the other hand, both components are required in that the controller is added to supervise multiple drivers so that they can work collaboratively in a distributed environment. In this case, each driver will spawn an additional daemon thread for receiving and responding controller commands.

COSBench is currently capable of measuring mainly 3 performance metrics, as listed in Table III. To demonstrate the effectiveness of the tool, we used it to evaluate the read performance of a 6-node swift storage cluster we have set up in our lab. Swift is an open source object store donated by Rackspace as a project under Openstack. Swift exposes the same interface as Cloud Files, so COSBench can work on it. Our swift cluster is comprised of 1 proxy node and 5 storage nodes. We ran 4 driver instances on 4 different client nodes with each driver stressing the cluster from 4 to as many as 512 workers using 100% read operations on objects 64 KB in size from 128 different containers for 300 seconds.

The hardware configurations are listed in Table IV. The results are illustrated in Fig. 1 regarding the average response time and throughput. The read performance increases as the number of workers per client increases until the throughput gets saturated at 64 workers per client and reaches its peak at 128 with 5644 read operations performed per second and each taking 90ms to complete. More workers beyond that only lineally prolong the response time but do not further increase the throughput. More experiments will be conducted in the future where more types of workloads and other object storage solutions will be included and results compared.

TABLE IV. SWIFT CONFIGURATION

Hardware Configuration ***(for Swift 1.4.3 on RHEL 6.1)***	
Proxy Node (P)	
CPU	2 * Intel X7560 2.27GHz (HT)
RAM / Disk	64GB / 250GB SATA
Storage Node (S) and Client Node (C)	
CPU	2 * Intel X5570 2.93GHz (HT)
RAM / Disk	12GB / 12 * 73GB SAS (DAS)
Network between Nodes[a]	
S-S / S-P / P-C / C-C	1GbE / 10GbE / 10GbE / 2 * 1GbE (bonding)

a. In this table section, S stands for storage node, P proxy node, and C client node

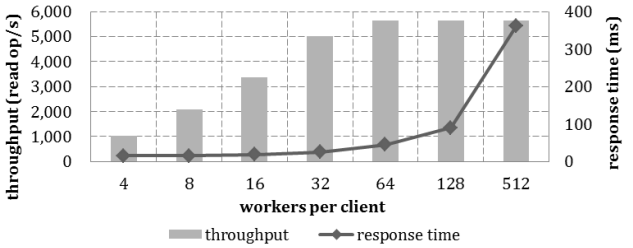

Figure 1. Read performance of the swift cluster

IV. CONCLUSION

We have presented our progress on creating a tool for benchmarking cloud object storage services. COSBench is still under our heavy development and is being enhanced in various aspects. To begin with, we are working on enriching the storage interface of our tool and adding more supports for other service implementations. In addition, there are efforts on the refinement of our existing workload model in order to sustain more sophisticated usage patterns. Moreover, as it is also worthwhile to spend time on improving usability, we are considering adding a GUI and a web-based user interface to the tool, with both interfaces capable of controlling drivers and showing results. Finally, we are going to evaluate more object storage service implementations so that we can gain more experience and a deeper understanding in this area.

REFERENCES

[1] IDC, http://www.idc.com/

[2] Amazon S3, http://aws.amazon.com/s3/

[3] Google Cloud Storage, http://www.google.com/enterprise/cloud/

[4] Rackspace Cloud Files, http://www.rackspace.com/cloud/

[5] Eucalyptus Walrus, http://open.eucalyptus.com/wiki/

[6] Openstack Swift, http://openstack.org/projects/storage/

[7] Beaver, D.; Kumar, S.; Li, H. C.; Sobel, J. & Vajgel, P., "Finding a needle in Haystack: facebook's photo storage", *in proceedings of the 9th USENIX conference on Operating systems design and implementation*, USENIX Association, 2010, 1-8

[8] CDMI, http://www.snia.org/cdmi