ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

Profiling and Benchmarking of Openstack Swift Object Storage

Neelam Patil[1], Shubhada Walekar[2], Trupti Mane[3], Prof. A.S.Shitole[4]

Sinhgad Academy of Engineering, Pune

Abstract: Openstack object storage system (swift) being increasingly recognized as preferred way to expose one's storage infrastructure as a service. Openstack object storage is a scalable redundant storage system. Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. As Swift is extremely dynamic and flexible, it needs to be tested under various load scenarios to monitor its performance. It is very challenging to evaluate its performance under various workloads. To address this problem we are building up a solution which benchmark the object store with different load characteristics and provide us with detailed statistics.

Index Terms: Openstack, Swift, Benchmark Tool, Workload, Proxy server, Master, Worker, Throughput, Latency, Success ratio.

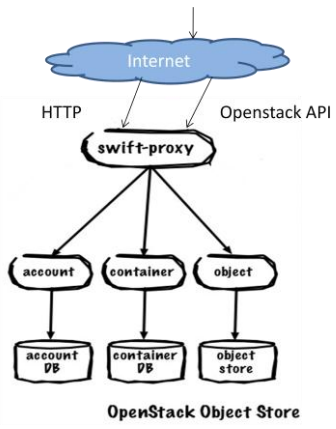
I. INTRODUCTION

A. OpenStack as IaaS

OpenStack is a collection of open source components to deliver public and private cloud. It is an IaaS cloud. Users of IaaS clouds can provision "processing, storage, network and other fundamental resources" on demand that is when needed, as long as needed, and paying only for what is actually consumed. The OpenStack components currently include OpenStack Compute (called Nova), OpenStack object storage (called Swift) and OpenStack image service (called Glance). OpenStack is a new effort and has received considerable momentum due to its openness and support of companies.

B. OpenStack Swift

Swift is a highly scalable cloud storage solution of Openstack. Swift is used to create scalable, redundant object storage using clusters of standardized servers in order to store petabytes of accessible data. Swift is suitable for archival storage purpose where static data can be uploaded for long
time storage, retrieved for analysis or writing over with new data. It is not a file system or real time data storage system.
The APIs are written in Python programming language and is deployable on servers that are running on any Linux OS. The Swift API can be installed standalone. The public API for Swift is exposed through a proxy server. The proxy server is responsible for finding the location of an account, container or object in the 'ring' and route the request accordingly. The ring represents the mapping between the names of entities stored on the disk and their physical location. There are three types of rings called account, object and container to perform each zones kind of operation. Ring maintains a mapping of zones, devices, partitions and replicas. Each replica that is usually up to 3 copies is guaranteed to reside in different zones. A zone can be a drive, server, cabinet or data center. The object server is responsible for facilitating the storage and retrieval of data files. Last write always wins when multiple put operations are performed on the same object and upon deletion all replicated copies are removed as well. Swift runs a container server whose primary object is to handle the listing of objects. Containers are similar to directory names or file folders. The listings are stored in the SqLite database. The account server is responsible for listing of containers.
The Swift architecture uses a replication process ensuring the data is protected in the event of disk failure and the network outages. It also make sure that all copies are up to date to the latest version and the deleted data is removed from all the replicated locations. There is an updater process which will take care of updating the data in a queue during periods of high load. Auditors check for the integrity of objects, containers and accounts and corrupted files. They will be quarantined and replaced from another replica.



OpenStack Object Store

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

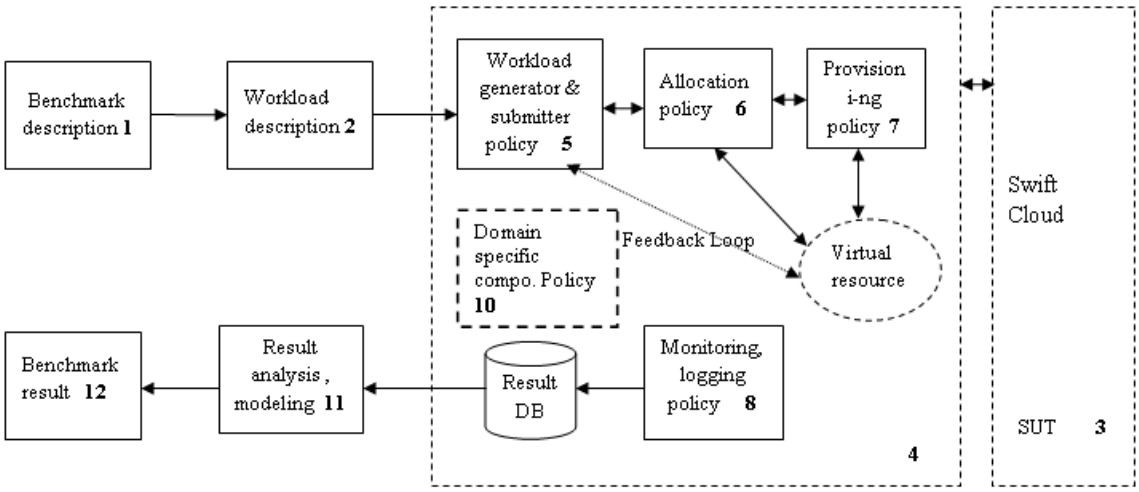
C. Need of benchmarking

We review in this section the main reasons for benchmarking. Benchmarking computer systems is the process of evaluating their performance and other non-functional characteristics with the purpose of comparing them with other systems or with industry-agreed standards. Traditionally, the main use of benchmarking has been to facilitate the informed procurement of computer systems through the publication of verifiable results by system vendors and third-parties. However, benchmarking has grown as a support process for several other situations. We view benchmarking as an empirical evaluation of performance that follows a set of accepted procedures and best-practices. We see a role for (statistical) simulation and mathematical analysis when the behavior of the system is well -understood and for long-running evaluations that would be impractical otherwise. Thus simulating new technology, such as cloud computing, requires careful (and time-consuming) validation of assumptions and models. Benchmarks, through their open-source nature and representation of industry accepted standards, can represent best-practices and thus be valuable training material.

II. DESIGN ARCHITECTURE OF IAAS SWIFT BENCH

We propose in this section a generic architecture for IaaS cloud benchmarking for OpenStack Swift. Our architecture focuses on conducting benchmarks as sets of (real-world) experiments that lead to results with high statistical confidence, on considering and evaluating Swift as evolving black-box systems, on employing complex workloads that represent multi-tenancy scenarios, on domain-specific scenarios, and on a combination of traditional and cloud-specific metrics.

In the architecture, the process begins with the user (e.g., a prospective IaaS cloud user) defining the benchmark configuration, that is, the complex workloads that definethe user's preferred scenario (component 1). The benchmarking system converts (component 2) the scenario into a set of workload descriptions, one per (repeated) execution. The workload may be defined before the benchmarking process, or change (in particular, increase) during the benchmarking process. After the preparation of the workload, the SUT (System Under Test) (component 3) is subjected to the workload through the job and resource management services provided by the testing system (component 4, which includes components 5–10). In the benchmarking architecture, the SUT can be comprised of one or several self-owned infrastructures, and public and private IaaS clouds. Here it is an OpenStack object storage Swift as SUT. The SUT provides resources for the execution of the workload; these resources are grouped into a *Virtual Resource Pool*. The results produced during the operation of the system may be used to provide a *feedback loop* from the Virtual Resource Pool back into the Workload Generator and Submitter (component 5); thus, the architecture can implement open and closed feedback loops. As a last important sequence of process steps, per-experiment results are combined into higher-level aggregates, first aggregates per workload execution (component 11), then aggregates per benchmark (component 12). We also envision the creation of a general database of results collected by the entire community and shared freely. The architecture supports various policies for provisioning and allocation of resources (components 6 and 7, respectively). The generic cloud benchmarking architecture also includes support for evolving black-box systems (components 9, 11, and 12), complex workloads and multi-tenancy scenarios (components 1, 2, and 5), domain-specific components (component 10), etc.



ARCHITECTURE OF IAAS SWIFT BENCH

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

III. OPEN CHALLENGES IN SWIFT IaaS CLOUD BENCHMARKING

We introduce in this section an open list of surmountable challenges in swift IaaS cloud benchmarking.

A. Methodological

Challenge 1. Experiment compression: Long setup times, for example of over a day, and/or long periods of continuous evaluation, for example of more than a day per result, reduce the usefulness of a benchmark for the general user. This is a general problem with any experimental approach, but for swift IaaS clouds it has the added disadvantage of greatly and visibly increasing the cost of benchmarking. Rsearch is needed to reduce the setup and operational time of benchmarks for IaaS clouds. This can be achieved through reduced input and application. Reduced benchmark input and application sets can be obtained by refining input workloads from real complex workloads, using theoretically sound methods (e.g., statistical models and goodness-of-fit tests). Such reduced benchmark inputs will contrast with traditional synthetic benchmarks, which incorporate many human-friendly parameter values (e.g., "10% queries of type A, 90% queries of type B") and thus may lack theoretical guarantees for representativeness.

B. System Properties

Challenge 2. Massive scale, multi-site benchmarking: One of the main product features of IaaS clouds is the promise of seemingly infinite capacity. Hence benchmarking this promise is difficult, very time-consuming, and very costly. Testing tools can be built to test infrastructures of thousands of cores, but performance evaluation tools that work at much larger scale in heterogeneous IaaS clouds have yet to be proven in practice. An important challenge here may be the ability to generate massive-scale workloads. Other cloud deployment models require the use of multiple sites, for reliability and vendor lock-in avoidance. We expect multi-site cloud use to increase, as more companies, with or without existing computational capacity, try out or even decide to use cloud services.[6] We argue that benchmarking across multiple sites raises additional challenges, not in the least the combined availability for testing and scalability of the infrastructure, and the increased cost.

Challenge 3. Performance isolation: The negative effects of the interaction between running jobs in a complex workload have been observed in distributed environments since at least the mid-1990s [6]. Following early work we argue that quantifying the level of isolation provided by an IaaS cloud is a new and important challenge.
Moreover, as IaaS clouds become more international, their ability to isolate performance may suffer most during periods of peak activity. Thus, studying the time patterns of performance isolation is worthwhile.

C. Workload

Challenge 4. Statistical models of workloads or of system performance:
Statistical workload modeling is the general technique of producing synthetic models from workload traces collected from real-world systems that are statistically similar to the real-world traces. We argue that building such statistical models raises important challenges, from data collection to trace processing, from finding good models to testing the validity of the models. We also see as an open challenge the derivation of statistical performance models, perhaps through linear regression, from already existing measurements. We envision that IaaS clouds will also be built for specific, even niche application domains, charging premium rates for the expertise required to run specific classes of applications. Toward building domain-specific benchmarks, we argue for building statistical models of domain-specific or at least programming model-specific workloads. We have conducted in the past extensive research in grid workloads.

III. PERFORMANCE

Performance is the primary measure by which swift measures capacity of cluster. This performance evaluation is carried out using following workloads:

A. *Read Benchmark*:

This simple test measures the raw ability of swift to handle thousands of reads . We test swift with files of varying sizes:

1 KB	10 MB
10 KB	100 MB
100 KB	1 GB
1 MB	

B. *Write Benchmark:*

This simple test measures the raw ability of swift to handle thousands of write. We test swift with files of varying sizes:

1 KB	10 MB
10 KB	100 MB
100 KB	1 GB
1 MB	

C. *Read-Write Benchmark:*

This test measures the raw ability of swift to handle thousands of read-write combinations. While giving combinations we can change percentage ratios of read write. For example 50% read 50% write or 40% read 60 % write and so on.

D. *Create-Update-Delete Benchmark:*

This test involves either combination or individual test for create-update-delete profiles. This test helps to analyze what how number of updates-deletes affect the

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

performance of swift under certain load. What is success ratio, throughput, latency are measured accurately in this benchmark test.

Benchmarking Parameters:

- To record First byte latency:
  - Min, Max, Avg
  - Min= Minimum Time Required for first response byte to arrive
  - Max= Maximum Time Required for first response byte to arrive
  - Avg= Average Latency
- To record Last byte latency:
  - Min, Max, Avg
  - Min= Minimum Time Required for last response byte to arrive
  - Max= Maximum Time Required for last response byte to arrive
  - Avg= Average Latency
- Throughput:
  - No. of operations per unit time
- Average Request per second
- Total number of processed request for each type of request
- Success ratio:
  - The Ratio of successful operations

$$\frac{No.of\ operation\ served}{No.of\ operations\ requested} * 100$$

IV. ACKNOWLEDGMENT

We express our special thanks and heartily gratitude to the Head of Department, Prof. B. B. Gite and our respected staff members for inspiring us throughout the completion of our project report.
We express our special thanks and heartily gratitude to Mr. Sameer Mainkar from Persistent System Pune, for providing us with his valuable inputs throughout the project report.
The acknowledgement would be incomplete if we don't record our sense of gratitude to our principal, Dr. A.G. Kharat who gave us the necessary guidance, encouragement, by providing us with all the facilities available to work on this project.

V. CONCLUSION

The importance of IaaS cloud benchmarking has grown proportionally to the increased adoption of this technology, from small and medium businesses to scientific users. We propose a generic approach for IaaS cloud swift benchmarking, in which resource and job management can be provided by the testing infrastructure, there is support for black-box systems that change rapidly and can evolve over time, where tests are conducted with complex workloads, and where various multi-tenancy scenarios can be investigated. We also discuss four variouschallenges in developing this approach: methodological, system property-related, workload related.

REFERENCES

[1]. grids.ucs.indiana.edu/ptliupages/laszewski-IEEECloud2012_id-4803.pdf
[2]. www.ucgrid.org/technical_reports/storage-paper.pdf
[3]. www.openstack.org/software/openstack-storage/
[4]. swift.openstack.org/
[5]. docs.openstack.org/developer/swift/development_saio
[6] Alexandru Iosup, Radu Prodan, and Dick Epema on IaaS cloud Benchmarking