SOFTWARE **Open Access**

OSSperf – a lightweight solution for the performance evaluation of object-based cloud storage services

Christian Baun*, Henry-Norbert Cocos and Rosa-Maria Spanou

Abstract
This paper describes the development and implementation of a lightweight software solution, called OSSperf, which can be used to investigate the performance of object-based public cloud storage services like Amazon S3, Google Storage and Microsoft Azure Blob Storage, as well as private cloud re-implementations. Furthermore, this paper presents an explanation of the output of the tool and some lessons learned during the implementation.

Keywords: Cloud computing, Storage services, Performance analysis

Introduction

A form of cloud services, which belong to the Infrastructure as a Service (IaaS) delivery model, is the group of object-based storage services. Examples for public cloud offerings, which belong to this kind of services, are the Amazon Simple Storage Service (S3) [1], the Google Cloud Storage (GCS) [2] and Microsoft Azure Blob Storage [3]. Furthermore, several free private cloud solutions exist, which re-implement the S3-functionality. Examples for service solutions, which implement the S3 API and are licensed according to a free software license, are Eucalyptus Walrus [4, 5], Ceph [6, 7], Nimbus Cumulus [8, 9], Fake S3 [10], Minio [11], Riak Cloud Storage [12], S3ninja [13], S3rver [14] and Scality S3 [15]. OpenStack Swift [16], which is the object storage component of the IaaS solution, provides a similar functionality, but implements the Swift API, which is quite similar to the S3 API.

In this work, different approaches and already existing solutions for the performance evaluation of object-based storage services are evaluated and a new solution, called OSSperf is developed. This new benchmark testing solution is intended to be lightweight, which means it causes only little effort for installation and usage. It shall support a wide number of different public and private cloud services with their different APIs. Additionally, it shall analyze the performance of the most important features of object-based storage services and in order to simulate scenarios of different degrees of utilization, the tool must provide a parallel operation mode for the upload and download of objects.

This paper is organized as follows. "Related work" section contains a discussion of related work and explains the motivation for the development of OSSperf.

In "Development and implementation of OSSperf" section, the design and implementation of the software and its functioning are discussed. In addition, this section provides a description of some challenges, we were facing during the development and implementation process.

"Analyzing the benchmark" section presents some benchmark results of OSSperf and explains how to analyze them.

Finally, "Conclusions" section presents conclusions and directions for future work.

Related work

Analyzing the performance of object-based storage services is a task, which has been addressed by several other works. First works on this topic were carried out and published shortly after the availability of the Amazon S3 service offering in 2006 (for the United States) and 2007 (for Europe), that became a blueprint for a large number of commercial competitors or free re-implementations. In the literature, several works cover the topic of measuring the performance of object-based storage services with the

*Correspondence: christianbaun@fb2.fra-uas.de
Frankfurt University of Applied Sciences, Nibelungenplatz 1, 60318 Frankfurt am Main, Germany

S3 interface and with the Microsoft Azure Blob Storage interface.

Garfinkel [17] evaluated in 2007 the throughput, which the Amazon S3 service offering can deliver via `HTTP GET` requests with objects of different sizes over several days from several locations by using a self-written client. The tool was implemented in C++ and used libcurl [18] for the interaction with the storage service. The focus of this work is the download performance of Amazon S3. Other operations like the upload performance are not investigated. Unfortunately, this tool has never been released by the author and the work does not investigate the performance and functionality of private cloud scenarios.

Palankar et al. [19] evaluated in 2008 the ability of Amazon S3 to provide storage to large-scale science projects from the perspective of cost, availability and performance. Among others, the authors evaluated the throughput (`HTTP GET` operations) which S3 can deliver in single-node and multi-node scenarios. They also measured the performance from different remote locations. No used tool has been released by the authors and the work does not mention the performance and functionality of private cloud scenarios.

Li et al. [20] analyzed the performance of the four public cloud service offerings Amazon S3, Microsoft Azure Blob Storage and Rackspace Cloud Files with their self developed Java software solution CloudCmp [21] for objects of 1 kB and 10 MB in size. The authors among others compare the scalability of the mentioned blob services by sending multiple concurrent operations and were able to make bottlenecks visible when uploading or downloading multiple objects of 10 MB in size.

Calder et al. [22] measured in 2011 the throughput (`HTTP GET` and `HTTP PUT` operations) of the Microsoft Azure Blob Storage service for objects of 1 kB and 4 MB in size. Unfortunately, the authors do not describe which tools they did use for their work and the authors do not mention the performance and functionality of further public cloud offerings or of private cloud scenarios.

Zheng et al. [23, 24] described in 2012 and 2013 the Cloud Object Storage Benchmark – COSBench [25], which is able to measure the performance of different object-based storage services. It is developed in Java and provides a web-based user interface and helpful documentation for users and developers. The tool is licensed according to the Apache 2.0 license and it supports the S3 API and the Swift API, but not the API of the Microsoft Azure Blob Storage service. COSBench can simulate different sorts of workload. It is among others possible to specify the number of workers, which interact with the storage service, and the read/write ratio of the access operations. This way, the software implements a parallel operation mode. The complexity of COSBench is also a drawback, because the installation and configuration require much effort.

Bessani et al. [26] analyzed in 2013 among others the required time to upload and download objects of 100 kB, 1 MB, and 10 MB in size from Amazon S3, Microsoft Azure Blob Storage and Rackspace Cloud Files from clients that were located on different parts of the globe. In their work, the authors describe DepSky [27], a software solution that can be used to create a virtual storage cloud by using a combination of diverse cloud service offerings in order to achieve better levels of availability, security, privacy and prevent the situation of a vendor lock-in. While the DepSky software has been released to the public by the authors, they did not publish a tool to carry out performance measurements of storage services so far.

McFarland [28] implemented in 2013 in the programming language Python two applications, called S3-perf, which make use of the boto [29] library to measure the download and upload data rate of the Amazon S3 service offering for different file object sizes. Those solutions offer only little functionality. They do only provide the option to measure the required time to execute upload and download operations sequentially. Parallel operations are not supported and also fundamental operations other than the upload and download objects are not considered. Furthermore, the solution does not support the Swift API or the API of the Microsoft Azure Blob Storage service and the software license is unclear.

Jens Hadlich implemented in 2015 an utility, called the S3 Performance Test Tool [30] in Java, which can be used to measure the required time to create and erase buckets, as well as for uploading and downloading files into Amazon S3 or S3-compatible object storage systems. The tool allows to upload and download objects in parallel. It is free software and licensed according to the MIT License. A drawback of the S3 Performance Test Tool is that it does not support the Swift API or the Microsoft Azure Blob Storage API and it does not calculate the achieved bandwidth during upload and download operations.

Land [31] analyzed in 2015 the performance of the public cloud object-based storage services Amazon S3, Google Cloud Storage and Microsoft Azure Blob Storage with files of different sizes by using the command line tools of the service providers and by mounting buckets of the services as file systems in user-space. Attaching the storage of an object-based storage service via a file systems in user-space driver is a quite special application, which is hardly comparable to the typical sort of interaction with such services. Additionally, this work does not analyze the performance and functionality of private cloud scenarios. The author uploaded for his work a file of 100 MB in size and a local git repository with several small files into the evaluated storage services and afterwards erased these files, but in contrast to our work, he did not investigate the performance of the single storage service related operations in detail.

Bjornson [32] measured in 2015 the latency - time to first byte (TTFB) and the throughput of the storage service offerings Amazon S3, Google Cloud Storage, and Microsoft Azure Blob Storage for objects ranging from 16 KB to 32 MB. The author used the Object Storage Benchmark, which is a part of the PerfKit Benchmarker [33] test suite. This tool is free software and licensed according to the Apache 2.0 license. The software supports not only multiple storage service interfaces, but also sequential and parallel operations. The author discovered, that the different public cloud services have different performance characteristics, which depend heavily on the object size. Thus, it is important for users and administrators to benchmark storage services with objects, that are comparable in size with the objects their web applications use. The work does not consider the typical storage service related operations in detail and it does not analyze the performance and functionality of private cloud scenarios. The PerfKit Benchmarker suite is a set of different benchmark applications to investigate different performance aspects of cloud Infrastructure services. The complexity of this collection is also a drawback, because the configuration and proper handling requires much effort.

In contrast to the related works in this section (see Table 1), we wanted to develop and implement a lightweight solution to analyze the performance of the most important storage service operations and not only of upload and download operations or of the latency. Furthermore we wanted to develop a tool which can interact not only with the Amazon S3 API, but also with the Swift API and the API of the Microsoft Azure Blob Storage service. Last but not least, the development of the tool aimed to create a solution which can be deployed with minimum effort and is simple to use.

Development and implementation of OSSperf

The analysis of the existing benchmark testing solutions in "Related work" section resulted in the development of a new tool. The preconditions of developing a lightweight solution, which causes only little effort for installation and usage, led to the development of a bash script. In order to simplify the development and implementation task, already existing command line tools were selected to carry out all interaction with the used storage services. In practice, the users and administrators of storage services have already installed some of these command line tools and are trained in working with at least one of them.

Users of OSSperf have the freedom to choose between these command line tools for the interaction with the used storage services:

- `az` [34]. A python client for the Azure Blob Storage API.
- `gsutil` [35]. A python client for the Google Cloud Storage service offering, which uses the S3 API.
- `mc` [36]. The so called Minio Client. It is developed in the programming language `go` and can interact with storage services that (re-)implement the API of the S3.
- `s3cmd` [37]. A python client, that can interact with storage services that (re-)implement the API of the S3.
- `swift` [38]. A python client for the Swift API.

Access to the storage services need to be configured in the way the above mentioned command line tools expect it.

Per default, if no other client software is specified via a command line parameter, OSSperf will try to use `s3cmd`

Table 1 Software solutions to analyze the performance of object-based storage services

Name of the benchmark solution	Author(s)	Released to public	Software license	Parallel mode	Support for S3 API	Support for Swift API	Support for ABS[a] API
CloudCmp [21]	Li et al.	Yes	MIT License	Yes	Yes	No	Yes
COSBench [25]	Zheng et al.	Yes	Apache 2.0	Yes	Yes	Yes	No
Object Stor. Benchm. [33]	Multiple	Yes	Apache 2.0	Yes	Yes	Yes	No
OSSperf [52]	Baun et al.	Yes	GPLv3	Yes	Yes	Yes	Yes
S3 Perf. Test Tool [30]	Hadlich	Yes	MIT License	Yes	Yes	No	No
S3-perf [28]	McFarland	Yes	Unknown	No	Yes	No	No
Unknown [22]	Calder et al.	No	Unknown	Unknown	No	No	Yes
Unknown [17]	Garfinkel	No	Unknown	Unknown	Yes	No	No
Unknown [19]	Palankar et al.	No	Unknown	Unknown	Yes	No	No

[a] ABS = Azure Blob Storage

for the interaction with the desired storage service. Once a storage service is registered in the configuration file of `s3cmd`, OSSperf can interact with the S3-compatible REST API of the service.

REST is an architectural-style that relies on HTTP methods like `GET` or `PUT`. S3-compatible services use `PUT` to receive the list of buckets that are assigned to a user account, or a list of objects inside a bucket, or an object itself. Buckets and objects are created with `PUT` and `DELETE` is used to erase buckets and objects. `POST` can be used to upload objects and `HEAD` is used to retrieve meta-data from an account, bucket or object. Uploading files into S3-compatible services is done via `POST` or `PUT` directly from the client of the user. Table 2 gives an overview of methods used to interact with S3-compatible storage services [39].

If the Swift API or the Microsoft Azure Blob Storage API shall be used, OSSperf does not use the command line tool `s3cmd`, but the Swift client (`swift`) or the Azure client (`az`). In contrast to `s3cmd`, the `swift` tool uses no configuration file with user credentials. Users of `swift` need to specify the endpoint of the used storage service as well as username and password inside the environment variables `ST_AUTH`, `ST_USER` and `ST_KEY`. Users of `az` need to specify the username and password inside the environment variables `AZURE_STORAGE_ACCOUNT` and `AZURE_STORAGE_ACCESS_KEY`.

Users of OSSperf have the freedom to use the Minio Client (`mc`) or the python client for the Google Cloud Storage (`gsutil`), instead of `s3cmd`, for the interaction with storage services that implement the S3-API. In this case, it is necessary to specify the connection parameters of the storage service as the first step in the appropriate configuration files of theses clients. Afterwards, the alternative clients can be set via command line parameter and used by OSSperf.

One of the aims during the development of OSSperf was to create a tool, which tests the performance of the most common used bucket- and object-related operations. These are in detail the creation and erasure of buckets and the upload, download and erasure of objects, as well as the operation, which returns a list of the objects, that are assigned to a specific bucket. These operations are in other words the CRUD actions (see Table 3), which are the four basic functions of persistent storage and are mostly mentioned in the field of SQL databases and therefore mapped to basic SQL statements, but they can also be mapped to HTTP methods, which are the foundation of the interaction with object-based storage services.

To investigate the performance of the CRUD actions, when doing a performance evaluation with OSSperf, the tool executes these six steps for a specific number of objects of a specific size:

1. Create a bucket
2. Upload one or more objects into this bucket
3. Fetch the list of objects inside the bucket
4. Download the objects from the bucket
5. Erase the objects inside the bucket
6. Erase the bucket

The time, which is required to carry out these operations is individually measured and can be used to analyze the performance of these commonly used storage service operations.

Users of OSSperf have the freedom to specify the number of files via command-line parameters, which shall be created, uploaded and downloaded, as well as their individual size. The files are created via the command line tool `dd` and contain data from the pseudorandom number generator behind the special file `/dev/random`.

In order to be able to simulate different load scenarios, the OSSperf tool supports the parallel transfer of

Table 2 Description of the HTTP methods with request-URIs that are used to interact with storage services

Account-related operations		
`GET`	`/`	List buckets
`HEAD`	`/`	Retrieve metadata
Bucket-related operations		
`GET`	`/bucket`	List objects
`PUT`	`/bucket`	Create bucket
`DELETE`	`/bucket`	Delete bucket
`HEAD`	`/bucket`	Retrieve metadata
Object-related operations		
`GET`	`/bucket/object`	Retrieve object
`PUT`	`/bucket/object`	Upload object
`DELETE`	`/bucket/object`	Delete object
`HEAD`	`/bucket/object`	Retrieve metadata
`POST`	`/bucket/object`	Update object

Table 3 The CRUD actions and their matching HTTP methods

CRUD actions	HTTP methods	SQL statements
Create or replace a resource	`PUT/POST`	`INSERT`
Read a resource	`GET`	`SELECT`
Update (= modify) a resource	`PUT/POST`	`UPDATE`
Delete a resource	`DELETE`	`DELETE`

```
2017-08-31 03:35:26 10 8388608 1.199 151.014 0.873 31.137 2.192 1.296 187.711 4.443 21.552
2017-08-31 03:41:31 10 8388608 1.345 150.945 0.671 31.050 2.918 1.698 188.627 4.445 21.613
2017-08-31 03:47:35 10 8388608 1.291 151.194 0.674 30.563 2.197 1.266 187.185 4.438 21.957
2017-08-31 03:53:39 10 8388608 1.393 150.082 0.667 31.961 2.158 1.522 187.783 4.471 20.997
2017-08-31 04:00:28 10 8388608 1.298 193.751 0.660 30.219 4.713 1.932 232.573 3.463 22.207
```

Fig. 1 Example of output data of OSSperf

objects, as well as requesting delete operations in parallel. If the parallel flag is set, the steps 2, 4 and 5 are executed in parallel by using the command line tool `parallel`. The steps 1, 3 and 6 can not in principle be executed in parallel.

Setting the bucketname, that OSSperf shall use, can also be done via command line parameter. The default bucketname is `ossperf-testbucket`, but in case the performance of a public cloud service offering shall be investigated, it may be required for the user to specify another name for the bucket, because the buckets inside a storage service need to have unique names.

A common assumption, when using storage services, which implement the same API, is that all operations cause identical results. But during the development of OSSperf, some challenges caused by non-matching service behavior emerged. One issue is the encoding of the bucket names. In order to be conforming to the DNS requirements, bucket names should not contain capital letters. To comply with this rule, the free private cloud storage service solutions Minio, Riak CS, S3rver, and Scality S3, do not accept bucket names with upper-case letters.

Other services like Nimbus Cumulus and S3ninja only accept bucket names, which are encoded entirely in upper case letters. The service offerings Amazon S3 and Google Cloud Storage, as well as the private cloud solution Fake S3 are more generous in this case and accept bucket names, which are written in lower case and upper-case letters.

In order to be compatible with the different storage service solutions and offerings, OSSperf allows the user to specify the encoding of the bucket name with a command line parameter.

Every time, OSSperf is executed, the tool prints out a line of data, which informs the user about the date (column 1) and time (column 2) when the execution was finished, the number of created objects (column 3), the size of the single objects in bytes (column 4), as well as the required time in seconds to execute the single steps 1–6 (columns 5–10). Column number 11 contains the sum of all time values, which is calculated by using the command line tool `bc`. The final columns 12 and 13 present the bandwidth B_U during the upload (step 2) and the bandwidth B_D during the download (step 4) operations. These values are also calculated with the command line tool `bc` with Eqs. 1 and 2, where N is the number of files, S is the size of the single files, T_U is the required time to upload and T_D the required time to download the files. Figure 1 contains five lines of output data, generated by OSSperf. These lines are a part of the output data, which were used to generate Fig. 3 and Table 4.

$$\frac{S\left[Byte\right] \times N \times 8}{T_U\ [\text{s}]} \ /\ 1000\ /\ 1000\ = B_U\ \ [\text{Mbit/s}] \qquad (1)$$

$$\frac{S\left[Byte\right] \times N \times 8}{T_D\ [\text{s}]} \ /\ 1000\ /\ 1000\ = B_D\ \ [\text{Mbit/s}] \qquad (2)$$

Table 4 Bandwidth during upload and download when using a multi-node storage service implemented with Minio in a private context

Size of the files [kB]	Upload (parallel) [Mbit/s]	Upload (seq.) [Mbit/s]	Download (parallel) [Mbit/s]	Download (seq.) [Mbit/s]
0.5	0.02	0.02	0.03	0.03
1	0.05	0.04	0.06	0.07
2	0.10	0.08	0.12	0.14
4	0.19	0.17	0.25	0.28
8	0.38	0.32	0.47	0.56
16	0.76	0.67	0.95	1.14
32	1.50	1.27	2.06	2.34
64	2.88	2.38	4.21	4.31
128	5.27	4.34	7.80	7.73
256	9.17	6.97	13.95	13.44
512	14.48	10.00	21.35	19.32
1024	20.60	12.23	29.28	24.51
2048	22.69	15.49	37.69	27.01
4096	28.03	16.89	39.99	28.65
8192	30.15	17.95	42.35	29.64

The structure of the output simplifies the analysis of the performance measurements by using command line tools like `sed`, `awk` and `gnuplot`.

Analyzing the benchmark

An example for the presentation of benchmark results, gained with OSSperf, present Fig. 3 and Table 4. For this scenario, the free storage service solution Minio [11] was installed in an 8-node deployment on a cluster of Raspberry Pi 2 single board computers [40, 41]. The client, which executed OSSperf, was connected via Ethernet with the same network switch as the cluster nodes. Each Raspberry Pi node has a 100 MBit/s Ethernet interface.

The gained data shows that the parallel upload of files is beneficial for objects of all sizes, but the download of objects < 128 kB in size requires a longer time in parallel transfer compared to sequential mode. With a growing object size, the service performs better in parallel transfer compared with sequential mode.

The effect, that the bandwidth gets better with a growing file size, is caused by the protocol overhead[1] of the object-based storage services (see Fig. 2). This overhead exists for file transmissions of any size and its portion of the potential throughput shrinks with a growing file size. The overhead is caused among others by theses characteristics:

- For upload and download operations, one communication partner needs to wrap the files into HTTP messages, TCP segments, IP packages and Ethernet frames and the other communication partner needs to unwrap them.
- The transmission of the HTTP messages via a computer network requires time (= network latency).
- Each upload operation requires the client to wait for a reply of the server, that indicates the request was successful.

One of the reasons, why the upload performance in Fig. 3 and Table 4 is slower compared to the download performance, may be the effort to store files in minio multi-node deployments. Minio uses erasure code and checksums [42] to increase the availability of the stored data, which causes much effort on the single board computers, which were used in this scenario.

Another example of benchmark results, gained with OSSperf, present Fig. 4 and Table 5. In this scenario, OSSperf was used to investigate the performance of the Amazon S3 service offering. Also when using S3, the upload performance is slower, compared with the download performance. The measured bandwidth during upload is close to the maximum data rate, the used internet service provider offers for upstream. Equal to the private cloud scenario from Fig. 3 and Table 4, the parallel upload of the ten files is never faster compared with the sequential upload. And again, the parallel upload is only beneficial when downloading large files. In this scenario, the turnaround point is when downloading objects ≥ 512 kB in size.

When investigating the performance of a public cloud storage service like Amazon S3, it must always be kept in mind, that the performance of this service may vary over the time of the day and that it is among others influenced by the network path between

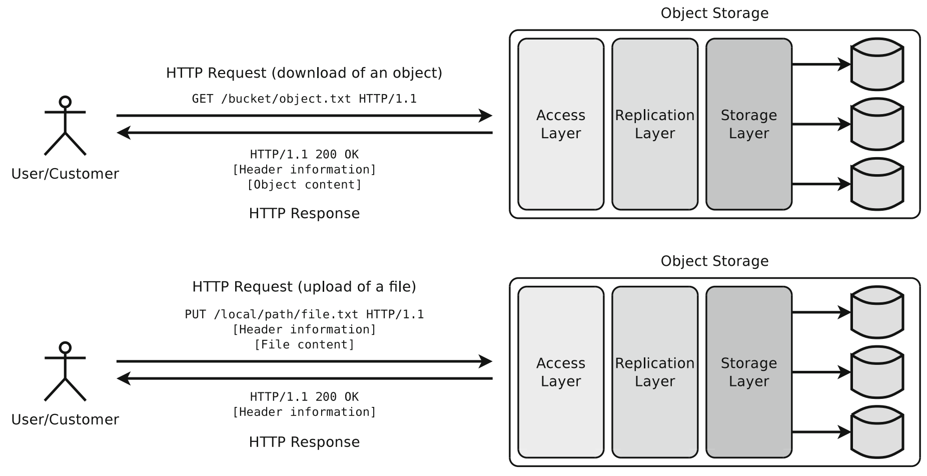

Fig. 2 Functioning of the object storage interface

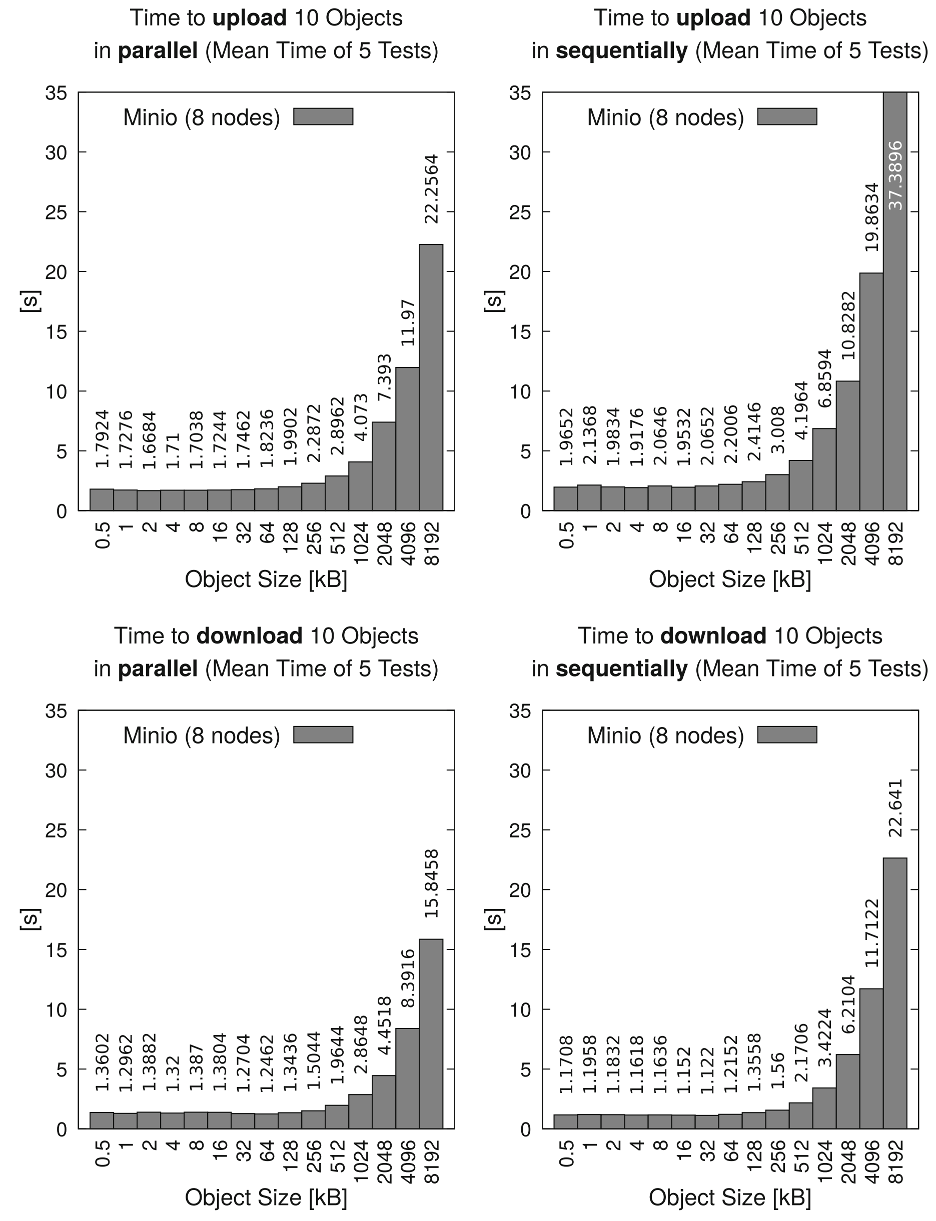

Fig. 3 Required time to upload and download sets of 10 files into a multi-node storage service implemented with Minio

client and service, as well as the current network utilization.

Conclusions

The development of OSSperf resulted in a solution to evaluate the performance of object-based cloud storage services, that is more lightweight than COSBench or the PerfKit Benchmarker test suite, because its installation and configuration does not require much effort and it has only few dependencies. OSSperf only requires the bash command line interpreter, `md5sum` for the creation and validation of the checksums, `bc` to carry out the calculations, `parallel` to implement the parallel mode and command line clients like `az`, `gsutil`, `mc`, `s3cmd` and `swift` for the interaction with the storage services.

In contrast to the works of Garfinkel and of Palankar et al., which both only consider `HTTP GET` requests, OSSperf carries out typical storage service operations and evaluates the time, which is required to execute these operations.

OSSperf supports the APIs of several different public and private cloud services by using clients that can

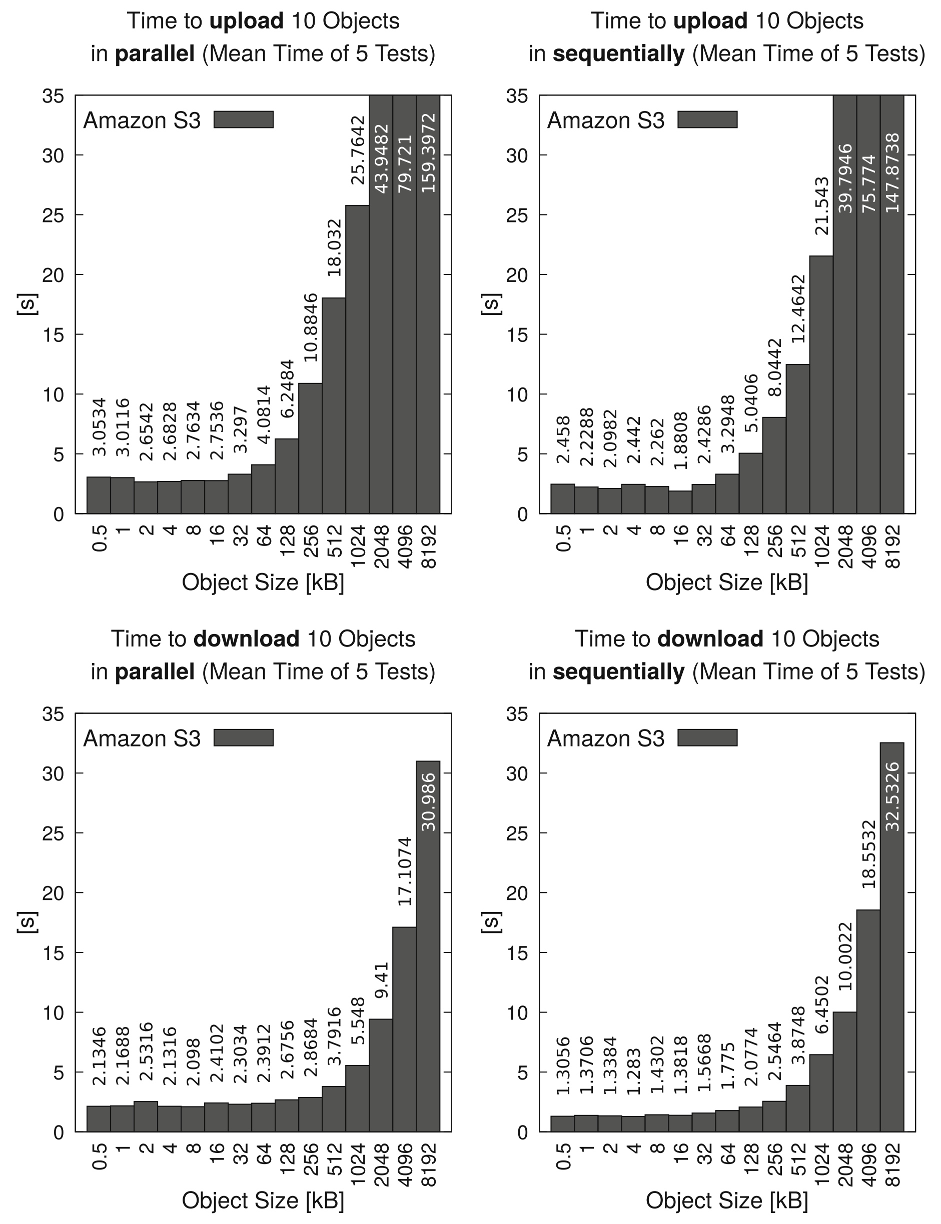

Fig. 4 Required time to upload and download sets of 10 files into the Amazon S3 public cloud storage service offering

interact via the S3-API, the Swift-API and the Azure Blob Storage API. This is a huge benefit in contrast to the other analyzed solutions, because the S3 Performance Test Tool, S3-perf, the work of Calder et al., the work of Garfinkel and the work of Palankar et al. support only a single API. CloudCmp, COSBench and the PerfKit Benchmarker test suite support only two APIs.

In contrast to S3-perf, the work of Calder et al., the work of Garfinkel and the work of Palankar et al., OSSperf is able to operate in sequential and parallel mode to create load situations of different sort.

The scope of functions sets OSSperf apart from the other solutions, we investigated in this work (see Table 1). With the performance information of OSSperf, users and administrators of storage services can find out which operations are most time consuming. This information is essential to optimize applications, which use storage services, as well as for optimizing storage service deployments.

OSSperf is free software and licensed under the terms of the GPLv3. The source code and documentation can be found inside the Git repository: https://github.com/christianbaun/ossperf/

Table 5 Bandwidth during upload and download when using the public cloud storage service offering Amazon S3

Size of the files [kB]	Upload (parallel) [Mbit/s]	Upload (seq.) [Mbit/s]	Download (parallel) [Mbit/s]	Download (seq.) [Mbit/s]
0.5	0.01	0.02	0.02	0.03
1	0.03	0.04	0.04	0.06
2	0.06	0.08	0.06	0.12
4	0.12	0.13	0.15	0.26
8	0.24	0.29	0.31	0.46
16	0.48	0.70	0.54	0.95
32	0.80	1.08	1.14	1.67
64	1.28	1.59	2.19	2.95
128	1.68	2.08	3.92	5.05
256	1.93	2.61	7.31	8.24
512	2.33	3.37	11.06	10.82
1024	3.26	3.89	15.12	13.01
2048	3.82	4.22	17.83	16.77
4096	4.21	4.43	19.61	18.09
8192	4.21	4.54	21.66	20.63

Next steps are the implementation for support of further command line clients to interact with object-based storage services like Rackspace Cloud Files [43] in order to allow a more detailed investigation of the way, the different clients influence the performance.

Endnote

[1] The network-level efficiency of object-based cloud storages is better compared with storage services like Google Drive [44], Microsoft OneDrive [45], Dropbox [46], Box [47] or Apple iCloud [48] – such services are sometimes called Personal Cloud Storage – because they all implement on top of object-based storage services an additional synchronization protocol in order to keep the data in a consistent state between client site and the service provide [49–51].

Abbreviations

ABS: Azure blob storage; API: Application programming interface; AWS: Amazon web services; Blob: Binary large object; CRUD: Create, read, update and delete; DNS: Domain name system; GCS: Google cloud storage; GPL: GNU general public license; HTTP: Hypertext transfer protocol; IaaS: Infrastructure as a service; IP: Internet protocol; kB: Kilobyte; MB: Megabyte; REST: Representational state transfer; S3: Simple storage service; TCP: Transmission control protocol

Acknowledgements
Many thanks to Katrin Baun for her assistance in improving the quality of this paper.

Funding
This work was funded by the Hessian Ministry for Science and the Arts (‘*Hessisches Ministerium für Wissenschaft und Kunst*’) in the framework of research for practice (‘*Forschung für die Praxis*’) and by the Faculty of Computer Science and Engineering Science of the Frankfurt University of Applied Sciences in the framework of ‘*Innovationsfonds Forschung*’ (IFOFO).

Availability of data and materials

The source code and documentation of the OSSperf software is stored inside the Git repository: https://github.com/christianbaun/ossperf/.

About the Authors

Dr. Christian Baun is a Professor at the Faculty of Computer Science and Engineering of the Frankfurt University of Applied Sciences in Frankfurt am Main, Germany. He earned his Diploma degree in Informatik (Computer Science) in 2005 and his Master degree in 2006 from the Mannheim University of Applied Sciences. In 2011, he earned his Doctor degree from the University of Hamburg. He is author of several books, articles and research papers. His research interest includes operating systems, distributed systems and computer networks.
Henry-Norbert Cocos studies computer science at the Frankfurt University of Applied Sciences. His research interest includes distributed systems and single board computers. Currently, he constructs a 256 node cluster of Raspberry Pi 3 nodes which shall be used to analyze different parallel computation tasks. For this work, he analyzes which administration tasks need to be carried out during the deployment and operation phase and how these tasks can be automated.
Rosa-Maria Spanou studies computer science at the Frankfurt University of Applied Sciences. Her research interest includes distributed systems and single board computers. Currently, she constructs and analyzes different multi-node object-based cloud storage solutions.

Authors’ contributions

CB and HNC carried out the literature review. CB and RMS developed the initial concept of the OSSperf benchmark solution and CB did most of the implementation. CB drafted the manuscript. HNC and RMS jointly provided useful remarks and critically reviewed the manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher’s Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 10 September 2017 Accepted: 20 November 2017
Published online: 02 December 2017

References

1. Amazon Simple Storage Service (S3). https://aws.amazon.com/s3/. Accessed 27 Nov 2017
2. Google Cloud Storage (GCS). https://cloud.google.com/storage/. Accessed 27 Nov 2017
3. Microsoft Azure Blob Storage. https://azure.microsoft.com/services/storage/blobs/. Accessed 27 Nov 2017
4. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D (2009) The Eucalyptus Open-source Cloud-computing System. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. IEEE. pp 124–131
5. Eucalyptus. https://github.com/eucalyptus/eucalyptus. Accessed 27 Nov 2017
6. Weil SA, Brandt SA, Miller EL, Long DD, Maltzahn C (2006) Ceph: A Scalable, High-Performance Distributed File System. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation. USENIX. pp 307–320
7. Ceph. https://github.com/ceph/ceph. Accessed 27 Nov 2017
8. Bresnahan J, Keahey K, LaBissoniere D, Freeman T (2011) Cumulus: An Open Source Storage Cloud for Science. In: Proceedings of the 2nd International Workshop on Scientific Cloud Computing. ACM. pp 25–32
9. Nimbus Cumulus. https://github.com/nimbusproject/nimbus. Accessed 27 Nov 2017
10. Fake S3. https://github.com/jubos/fake-s3. Accessed 27 Nov 2017

11. Minio. https://github.com/minio/minio. Accessed 27 Nov 2017
12. Riak CS. https://github.com/basho/riak_cs. Accessed 27 Nov 2017
13. S3ninja. https://github.com/scireum/s3ninja/. Accessed 27 Nov 2017
14. S3rver. https://github.com/jamhall/s3rver/. Accessed 27 Nov 2017
15. Scality S3. https://github.com/scality/S3. Accessed 27 Nov 2017
16. OpenStack Object Storage (Swift). https://github.com/openstack/swift. Accessed 27 Nov 2017
17. Garfinkel S (2007) An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS. Harvard Computer Science Group. Technical Report TR-08-07
18. Libcurl. https://curl.haxx.se/libcurl/. Accessed 27 Nov 2017
19. Palankar MR, Iamnitchi A, Ripeanu M, Garfinkel S (2008) Amazon S3 for Science Grids: a Viable Solution? In: Proceedings of the 2008 International Workshop on Data-aware Distributed Computing. ACM. pp 55–64
20. Li A, Yang X, Kandula S, Zhang M (2010) CloudCmp: Comparing Public Cloud Providers. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement. pp 1–14
21. CloudCmp. https://github.com/angl/cloudcmp. Accessed 27 Nov 2017
22. Calder B, Wang J, Ogus A, Nilakantan N, Skjolsvold A, McKelvie S, Xu Y, Srivastav S, Wu J, Simitci H, Haridas J, Uddaraju C, Khatri H, Edwards A, Bedekar V, Mainali S, Abbasi R, Agarwal A, Haq MFu, Haq MIu, Bhardwaj D, Dayanand S, Adusumilli A, McNett M, Sankaran S, Manivannan K, Rigas L (2011) Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP '11. pp 143–157
23. Zheng Q, Chen H, Wang Y, Duan J, Huang Z (2012) COSBench: A Benchmark Tool for Cloud Object Storage. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference On. pp 998–999
24. Zheng Q, Chen H, Wang Y, Zhang J, Duan J (2013) COSBench: Cloud Object Storage Benchmark. In: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering. pp 199–210
25. COSBench. https://github.com/intel-cloud/cosbench. Accessed 27 Nov 2017
26. Bessani A, Correia M, Quaresma B, André F, Sousa P (2013) DepSky: Dependable and Secure Storage in a Cloud-of-Clouds. ACM Trans Storage (TOS) 9(4):12
27. DepSky. https://github.com/cloud-of-clouds/depsky. Accessed 27 Nov 2017
28. S3-perf. https://github.com/ross/s3-perf. Accessed 27 Nov 2017
29. Boto. https://github.com/boto/boto. Accessed 27 Nov 2017
30. S3 Performance Test. https://github.com/jenshadlich/S3-Performance-Test. Accessed 27 Nov 2017
31. Real-world Benchmarking of Cloud Storage Providers: Amazon S3, Google Cloud Storage, and Azure Blob Storage. https://lg.io/2015/10/25/real-world-benchmarking-of-s3-azure-google-cloud-storage.html. Accessed 27 Nov 2017
32. AWS S3 Vs Google Cloud Vs Azure: Cloud Storage Performance. http://blog.zachbjornson.com/2015/12/29/cloud-storage-performance.html. Accessed 27 Nov 2017
33. Perfkit Benchmarker. https://github.com/GoogleCloudPlatform/PerfKitBenchmarker. Accessed 27 Nov 2017
34. Command-line Tools for Azure. https://github.com/Azure/azure-cli. Accessed 27 Nov 2017
35. Gsutil. https://github.com/GoogleCloudPlatform/gsutil. Accessed 27 Nov 2017
36. Minio Client. https://github.com/minio/mc. Accessed 27 Nov 2017
37. S3cmd. https://github.com/s3tools/s3cmd. Accessed 27 Nov 2017
38. Swift Client. https://github.com/openstack/python-swiftclient. Accessed 27 Nov 2017
39. Baun C, Kunze M, Schwab D, Kurze T (2013) Octopus-A Redundant Array of Independent Services (RAIS). In: CLOSER 2013: Proceedings of the 3rd International Conference on Cloud Computing and Services Science. pp 321–328
40. Baun C (2016) Mobile Clusters of Single Board Computers: an Option for Providing Resources to Student Projects and Researchers. SpringerPlus 5(1):360
41. Baun C, Cocos HN, Spanou RM (2017) Performance Aspects of Object-based Storage Services on Single Board Computers. Open J Cloud Comput (OJCC) 4(1):1–16
42. Minio Erasure Code Quickstart Guide. https://github.com/minio/minio/tree/master/docs/erasure. Accessed 27 Nov 2017
43. Rackspace Cloud Files. https://www.rackspace.com/cloud/files/. Accessed 27 Nov 2017
44. Google Drive. https://www.google.com/drive/. Accessed 27 Nov 2017
45. Microsoft OneDrive. https://onedrive.live.com. Accessed 27 Nov 2017
46. Dropbox. https://www.dropbox.com. Accessed 27 Nov 2017
47. Box. https://www.box.com. Accessed 27 Nov 2017
48. iCloud. https://www.icloud.com. Accessed 27 Nov 2017
49. Li Z, Jin C, Xu T, Wilson C, Liu Y, Cheng L, Liu Y, Dai Y, Zhang ZL (2014) Towards Network-level Efficiency for Cloud Storage Services. In: Proceedings of the 2014 ACM Conference on Internet Measurement Conference. IMC '14. pp 115–128
50. Drago I, Bocchi E, Mellia M, Slatman H, Pras A (2013) Benchmarking personal cloud storage. In: Proceedings of the 2013 ACM Conference on Internet Measurement Conference. IMC '13. pp 205–212
51. Drago I, Mellia M, M. Munafo M, Sperotto A, Sadre R, Pras A (2012) Inside Dropbox: Understanding Personal Cloud Storage Services. In: Proceedings of the 2012 ACM Internet Measurement Conference. IMC '12. pp 481–494
52. OSSperf. https://github.com/christianbaun/ossperf. Accessed 27 Nov 2017