Open Access

Open Journal of Cloud Computing (OJCC)
Volume 4, Issue 1, 2017
http://www.ronpub.com/ojcc
ISSN 2199-1987

Performance Aspects of Object-based Storage Services on Single Board Computers

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

Faculty of Computer Science and Engineering, Frankfurt University of Applied Sciences,
Nibelungenplatz 1, 60318 Frankfurt am Main, Germany
christianbaun@fb2.fra-uas.de, cocos@stud.fra-uas.de, spanou@stud.fra-uas.de

Abstract

When an object-based storage service is demanded and the cost for purchase and operation of servers, workstations or personal computers is a challenge, single board computers may be an option to build an inexpensive system. This paper describes the lessons learned from deploying different private cloud storage services, which implement the functionality and API of the Amazon Simple Storage Service on a single board computer, the development of a lightweight tool to investigate the performance and an analysis of the archived measurement data. The objective of the performance evaluation is to get an impression, if it is possible and useful to deploy object-based storage services on single board computers.

Type of Paper and Keywords

Regular research paper: *single board computers, private cloud, object-based storage services, simple storage service*

1 Introduction

A specific sort of cloud services, which belongs to the Infrastructure as a Service (IaaS) delivery model, is the object-based storage services. Examples for public cloud offerings, which belong to this sort of services, are the Amazon Simple Storage Service (S3) and Google Cloud Storage, which also implements the S3-API. Furthermore exist several free private cloud solutions, which implement the S3-functionality and API.

For research projects, educational purposes like student projects and in environments with limited space and/or energy resources, single board computers may be a useful alternative to commodity hardware servers, because they require lesser purchase costs and operating costs. In addition, such single board computer systems can be constructed in a way that they can easily be transported by the users [2] because of their low weight and compact design. Single board computer systems are not only well suited for many cloud computing applications, but also for Internet of Things (IoT) [19] [47] scenarios and more modern distributed systems architecture paradigms like Fog Computing [7] [25] and Dew Computing [39] [45].

The drawback of single board computers is the limited hardware resources, which cannot compete with the performance of higher-value systems [3]. For this work, the Raspberry Pi 3 single board computer was selected as hardware platform, because the purchase cost for this device is just approximately 40 € and operating system images of several different Linux distributions exist for its architecture. The computer provides four CPU cores (ARM Cortex A8), 1 GB of main memory and a 10/100 Mbit Ethernet interface. A microSD card is used as local storage.

In [2] we already investigated the characteristics, as well as the performance and energy-efficiency of a mobile cluster of eight Raspberry Pi 1 Computers and its single components. In this work we also mentioned useful applications for such a system. In [3] we analyzed and compared the performance and energy-efficiency of different clusters of Single Board Computers with the High Performance Linpack (HPL) [20] benchmark. This work covered clusters of Raspberry Pi 1, BananaPi and Raspberry Pi 2 nodes. For this work, several free private cloud S3-reimplementations have been evaluated and deployed on a Raspberry Pi 3 when it was possible. The performance and energy efficiency have been investigated among others with S3perf, a self-developed tool that investigates the required time to carry out typical storage service operations.

This paper is organized as follows. Section 2 contains a discussion of related work. In Section 3, the deployment of different private cloud storage services on the Raspberry Pi 3 is described. Section 4 presents the development and implementation of the tool S3perf. An analysis of the gained measurement data is done in Section 5. Section 6 presents conclusions and directions for future work.

2 Related Work

In the literature, several works cover the topic of measuring the performance of storage services with the S3 interface.

Garfinkel [18] evaluated in 2007 the throughput via `HTTP GET` operations which Amazon S3 can deliver with objects of different sizes over several days from several locations by using a self-written tool. The work is focused on download operations and does not analyze the performance of other storage service related operations in detail. 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. [32] evaluated in 2008 the ability of Amazon S3 to provide storage support to large-scale science projects from a cost, availability, and performance perspective. 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.

Zheng et al. [50] described in 2012 and 2013 the Cloud Object Storage Benchmark (COSBench) [12], which is able to measure the performance of different object-based storage services. It is written in Java and provides a web-based user interface and helpful documentation for users and developers. The tool supports not only the S3 API, but also the Swift API and it 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 [49]. The complexity of COSBench is also a drawback, because the installation and configuration requires some effort.

McFarland [43] implemented in 2013 two Python scripts, which make use of the boto [8] 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 just allow 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.

Land [26] analyzed in 2015 the performance of different public cloud object-based storage services 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. 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 the files, but in contrast to our work, he did not investigate the performance of the single storage service related operations in detail.

Bjornson [6] measured in 2015 the latency – time to first byte (TTFB) – and the throughput of different public cloud object-based storage services by using a self-written tool. Unfortunately, this tool has never been released by the author. The work does not consider the typical storage service related operations and it does not analyze the performance and functionality of private cloud scenarios.

In contrast to the related works in this section, we evaluate the performance of private cloud solutions on single board computers and not the performance of one or few public cloud service offerings. In addition, we developed and implemented a flexible and lightweight solution to analyze the performance of the most important storage service operations and not only of the `HTTP GET` operation or of the latency. We used this tool to analyze the performance of storage service solutions, which has also been released as free software.

3 Deployment of Private Cloud Storage Services with the S3-Interface on a Single Board Computer

For this work, several free private cloud storage services, which re-implement the functionality of the Amazon S3 and its API, have been taken into consideration. Table 1 presents an overview of the investigated storage services and their characteristics.

Some of the investigated services (Cumulus, Fake S3, S3ninja, S3rver, Scality S3 Server and Walrus) support only single-node deployments, which limits the reliability and scalability of these services in principle. These services focus on providing lightweight solutions, mainly for testing and development purposes. Ceph-RGW, Minio, Riak CS and Swift are designed to implement multi-node deployments.

The table among others highlights the programming languages, in which the different storage service solutions are implemented in. This information may be important for application scenarios where additional functions need to be implemented by the users. Furthermore, in practice, not all systems provide all compilers or runtime environments.

When implementing a single-node service, the durability of the stored data depends entirely on the storage system, which is used. It is possible to use these services as a front end and via the S3 interface to access

- a folder inside a POSIX file system, which is located on a local connected storage drive,
- a distributed file system, which spans via multiple physical servers,
- a network attached storage (NAS) or even
- a storage area network (SAN).

When using a multi-node deployment with Ceph-RGW, Minio, Riak CS or Swift, the data is replicated over several nodes by the storage service itself.

Whenever possible, the services, described in this work have been deployed on a Raspberry Pi 3 single board computer. The operating system used was Raspbian GNU/Linux 8.0 – which is a Debian Jessie derivate – with release date 2017-03-02. The used Linux kernel was revision 4.4.50. For this work, all deployed storage services used a local folder on the microSD card, which is connected to the Raspberry Pi 3, to store the buckets and objects. In order to investigate the performance of the services, the required time to carry out a set of operations was measured.

3.1 Ceph-RGW

The distributed file system Ceph [46] supports among others the Amazon S3 REST API [11]. This functionality of Ceph is also called Ceph Object Gateway or RADOS Gateway (RGW). Ceph is free software and licensed according to the GNU Lesser General Public License (LGPL).

Despite several success stories [1] [13] [14] [44] in literature, which describe the deployment of Ceph on Raspberry Pi 3 computers, our attempts to install and configure Ceph inside the used Raspbian revision did not result in a stable testbed. Therefore, for this work, Ceph was not further evaluated.

3.2 Cumulus

The storage service Cumulus [9] has been developed in the context of the Nimbus infrastructure project [24]. Cumulus is developed in Python and its source code is licensed according to the Apache License 2.0 [30]. The Cumulus service allows only single-node deployments. Because Cumulus does not rely on any higher level Nimbus libraries, it is possible to install it as a stand alone storage service without the entire Nimbus IaaS solution.

In order to install Cumulus, the operating system must provide the Python interpreter 2.5 or a more recent revision. The single installation steps are well documented in the Cumulus documentation inside the `QUICKSTART.txt` file, which can be found inside the Cumulus source code repository.

In case of a Cumulus only installation, the user management functionality of the Nimbus infrastructure cannot be used. For such scenarios, Cumulus provides a set of command line tools (`cumulus-add-user`, `cumulus-list-users` and `cumulus-remove-user`) to create, remove, and print out a list of current users.

After the first start of the service, all relevant configuration data is stored inside the file `~/cumulus/etc/cumulus.ini`. Among others, the following parameters are specified: The port number of Cumulus, the path of the folder, which is used to store the buckets and objects, the hostname, the path and file name of the logfile and if HTTPS shall be used.

3.3 Fake S3

The Fake S3 service is developed in the programming language Ruby and until revision v0.2.5, the source code is licensed according to the MIT License [17]. Later revisions are not licensed according to a software license which complies with the open source definition of the

Table 1: Storage services which implement the S3-API and their characteristics

Service	Tested revision	Multi-node deployment	Programming language	License
Ceph-RGW	v10.2.7	possible	C++	LGPL
Cumulus	v2.10.1	impossible	Python	Apache License 2.0
Fake S3	v1.0.0	impossible	Ruby	MIT License
Minio	v2017-03-16	possible	Go	Apache License 2.0
Riak CS	v2.1.1	possible	Erlang	Apache License 2.0
S3ninja	v2.7	impossible	Java	MIT License
S3rver	v0.0.7	impossible	Node.js	MIT License
Scality S3 Server	v7.0.0	impossible	Node.js	Apache License 2.0
Swift (with the swift3 middleware)	v2.13.0	possible	Python	Apache License 2.0
Walrus	v4.4.0	impossible	Java	GPLv3

Open Source Initiative (OSI). The service offers only single-node deployments and no graphical user interface.

In order to deploy Fake S3, the programming language Ruby and the package manager RubyGems need to be installed first. The installation of Fake S3 is done via the command `gem install fakes3`. Important service related parameters like the port number and the path of the S3 data folder are provided as command line parameters when starting the Fake S3 service.

During the evaluation of Fake S3, some bugs have been observed with this service. Buckets are still seen by the service, even if they have been erased prior. With the evaluated revisions, it was not possible to modify the access key and the secret access key via command line parameters or to specify them via environment variables.

3.4 Minio

The Minio service is developed in the programming language Go and its source code is licensed according to the Apache License 2.0 [28]. Minio provides a web user interface, which allows to carry out all object and bucket related tasks. The Minio project offers not only the source code of the service, but also binaries for the operating systems Mac OS X, Linux, FreeBSD and Microsoft Windows. Linux binaries for x86-compatible architectures and even ARM architectures are available.

When starting the service, the port number and a local folder for the object data, are provided as command line parameters. The user access key and secret access key can be specified via the environment variables `MINIO_ACCESS_KEY` and `MINIO_SECRET_KEY`.

During the first start of the service, all relevant configuration data is stored inside the file `~/.minio/config.json`. Inside this file, among others the following parameters are specified: The user access key and secret access key, if the web user interface shall be used, if the server shall print out messages on command line and the logging level and if the server shall write messages in a logfile and the file name as well as the logging level.

Since November 2016, Minio provides multi-disk support with internal replication on single node deployments and across multiple nodes. To avoid a single node being a bottleneck for requests, a tool like the light-weight web server software `nginx` can be used as a proxy [27].

3.5 Riak CS

Riak CS (Cloud Storage) implements a S3-like object storage with the S3-API on top of Riak KV [29], which is a distributed NoSQL key-value data store. Riak KV implements the characteristics of Amazon Dynamo [15] and has therefore a focus on multi-node scalability, fault-tolerance and data durability thanks to internal replication.

The evenly distribution of stored data is carried out by Riak KV fully automatic. To achieve this characteristic, Riak KV places the keys inside a Distributed Hash Table (DHT) [23]. The DHT is split into partitions, circularly distributed among the nodes, and replicas are always placed in contiguous partitions [10].

To enforce the global uniqueness of entities, a Riak CS storage system also requires the service Stanchion [41], which serializes the requests that involve creation and modification of the entities. When a user for instance tries to create a bucket with an already existing bucket name, this request is rejected by Stanchion. The same result is caused by the attempt to create a user with the Email address of an already existing user [22].

Riak KV, Stanchion and Riak CS are implemented in the programming language Erlang and are free software. The source code [34] is licensed according to the Apache License 2.0. The installation of Riak KV, Stanchion and Riak CS, as well as a compatible revision of the

Erlang compiler, requires a lot of effort, because no pre-compiled packages were available for Raspbian and the ARM architecture of the Raspberry Pi. Compiling all required components requires more than an hour on a Raspberry Pi 3. The services Riak KV, Stanchion and Riak CS have their own configuration files and need to be started in the correct order.

With Riak CS Control, a standalone web application for the user management of Riak CS systems exists. Equal to the other Riak components already described, Riak CS Control is free software and implemented in Erlang. Unfortunately, all attempts to deploy Riak CS Control on the Raspberry Pi 3 failed in this work.

3.6 S3ninja

The S3 ninja service is developed in the programming language Java and its source code is licensed according to the MIT License [36]. The service offers only single-node deployments. S3ninja provides a web user interface which allows to carry out all object and bucket related tasks.

From a perspective of operating system, S3ninja requires just an installed Java runtime environment. In effect, the deployment and start of this service solution require little effort. An unexpected issue arose from the interaction of S3ninja with `s3cmd`: The used `s3cmd` revision v2.7 forces the API to be accessible at path `/`, but S3ninja only permits the path `/s3`. This issue was solved by installing `nginx` and using it as a proxy.

In the configuration file of the service `~/s3ninja/app/application.conf`, the user access key and the secret access key can be specified. Also the path of the S3 data folder and the port number is specified inside this file.

3.7 S3rver

The service S3rver is developed in the programming language JavaScript and its source code is licensed according to the MIT License [37]. The software is executed on server-side by using Node.js. The service offers only single-node deployments and no graphical user interface.

In order to deploy S3rver, the Node.js run-time environment and the package manager npm need to be installed first. The installation of S3srver is done via the command `npm install s3rver -g`. Important service related parameters like the port number and the path of the S3 data folder are provided as command line parameters when starting the service. An unusual characteristic of S3rver is, that the IP address also needs to be specified as a command line parameter. If not specified, the port number of the service only accepts connections from localhost.

With the evaluated revisions, it was not possible to modify the access key and the secret access key via command line parameter or to specify them via environment variables.

3.8 Scality S3 Server

The Scality S3 Server, which offers only single-node deployments, is developed in the programming language JavaScript and its source code is licensed according to the Apache License 2.0 [38]. The software is executed on server-side by using the Node.js JavaScript runtime environment. If Node.js and the package manager npm are already installed, the service can be deployed via the command `npm install`.

One characteristic of Scality S3 Server is that its main memory consumption exceeds the physical main memory of the Raspberry Pi 3. In order to solve this issue, a Swap partition of at least 1 GB of size or a Swap file is required.

The user access key and secret access key can be specified via the environment variables `SCALITY_ACCESS_KEY_ID` and `SCALITY_SECRET_ACCESS_KEY` or inside the file `~/S3/conf/authdata.json`. Further relevant configuration parameters like the logging level and the port number of the service are specified inside the file `~/S3/config.json`.

3.9 Swift

Swift is a component of the IaaS solution OpenStack. It is written in Python and licensed according to the Apache License 2.0 [31]. The OpenStack project was initiated in 2010 by Rackspace and NASA with the objective to develop a scalable, durable and high-available object storage, which can be deployed across a large number of nodes. The swift service ensures data replication and integrity across the connected nodes.

Although being an object-storage service like the other services examined in this work, Swift implements an API, which is different to the S3-API. For interaction on command line, the python client [33] for the Swift API is a working solution. If the S3-API is mandatory, a middleware like swift3 [42] needs to be deployed.

The Swift service offers two options for authentication. The first option requires the OpenStack Keystone service installed and the second option makes use of the TempAuth functionality, which is implemented in Swift. The Keystone option was discarded, because a lightweight solution was wanted. The user credentials when using TempAuth are specified inside the file `/etc/swift/proxy-server.conf`. Further relevant configuration parameters, which are also specified inside this file, are among others the logging level and the port number of the swift service.

The deployment of Swift on the Raspberry Pi 3 needed much effort because of the several software dependencies and the complex configuration. The installation of the latest revision 2.14.0 was possible, but because of a software bug or a configuration issue, it was only possible to create buckets, but impossible to upload objects into the service. Therefore, revision 2.2.0 of the Swift service was installed, because packages of this revision for the Raspbian operating system do already exist.

3.10 Walrus

Walrus is a component of the IaaS solution Eucalyptus [16]. It is written in Java and licensed according to the GPLv3 license. The development is mainly driven by Hewlett Packard Enterprise (HPE), since HP acquired the company Eucalyptus Systems in 2014. Of all private cloud service solutions, mentioned in this work, Walrus has the longest development period. It is a part of Eucalyptus since revision 1.4 in 2009.

The developers support only the installation of Eucalyptus inside the operating systems CentOS 7 and Red Hat Enterprise Linux 7. Installations on different Linux distributions cause much effort because of multiple dependencies. Walrus uses a folder inside a POSIX file system to store the buckets as folders and the object data as files. The durability of the stored data depends entirely on the used storage system.

A stand alone installation of Walrus out of the box is impossible because Walrus does not utilize the POSIX file system for storing the metadata. These are stored in the database, which is managed by the Cloud Controller (CLC) – a different Eucalyptus component.

The service allows only single-node deployments. Instead of adding multi-node functionality into Walrus, the developers decided to support using Riak CS or Ceph-RGW as object storage in Eucalyptus instead of Walrus.

A stand-alone installation of Walrus is impossible and the installation of Eucalyptus was not successful on Raspbian GNU/Linux. Installation attempts with Ubuntu 16.04 LTS failed the same. Therefore, for this work, Walrus was not further evaluated.

4 Development of S3perf

An analysis of the existing testing solutions (see section 2) resulted in the development of a new tool, called S3perf [5]. Its focus was to be lightweight, be

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

compatible with the S3-API and the Swift API, have only few dependencies, cause only little effort for the installation and be simple to use.

One well-documented option to develop software, which shall interact with AWS-compatible services, is using the programming language Python together with the boto [8] library. In order to even simplify the implementation, the command line tools `s3cmd` [35] and the Swift client [33] were selected to carry out all interaction with the used storage services. These preconditions led to the development of a bash script.

Once a storage service is registered in the configuration file of `s3cmd`, S3perf can interact with its S3-compatible REST API. REST is an architectural-style that relies on HTTP methods like `GET` or `PUT`. S3-compatible services use `GET` to receive the list of buckets that are assigned to an 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` directly from the client of the user. Table 2 gives an overview of methods used to interact with S3-compatible storage services. [4]

If the Swift API shall be used for the communication with a storage service, S3perf does not use the command line tool `s3cmd`, but the Swift client. In contrast to `s3cmd`, the `swift` tool uses no configuration file to store user credentials. The user of S3perf needs to

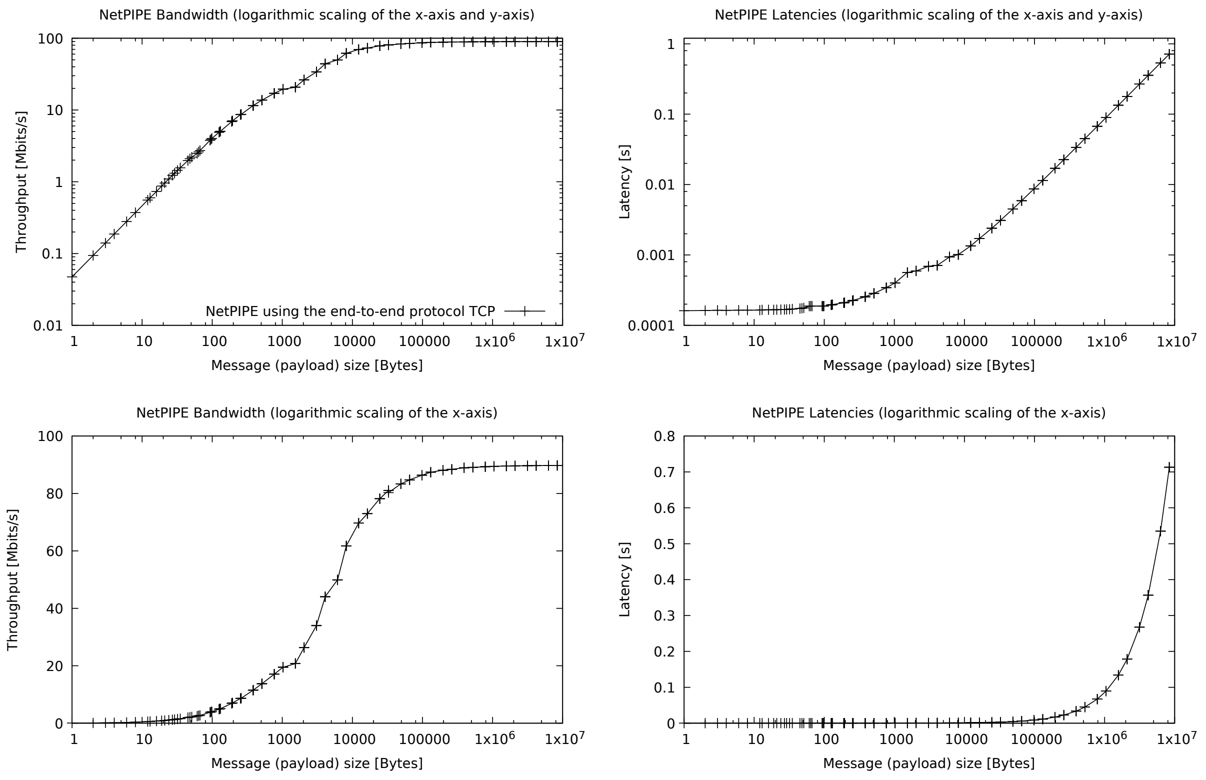


Figure 1: Analysis of the network performance of the Raspberry Pi 3 by using the NetPIPE benchmark

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`.

A common assumption, when using storage services, which implement the same API, is that all operations cause identical results. But during the development of S3perf, 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 services Minio, Riak CS, S3rver and Scality S3 do not accept bucket names with capital letters. Other services like Nimbus Cumulus and S3ninja only accept bucket names, which are encoded entirely in capital letters. The service offerings Amazon S3 and Google Cloud, as well as the private cloud solutions Fake S3 and OpenStack Swift are more generous in this case and accept bucket names, which are written in lowercase and capital letters. In order to be compatible with the different storage service solutions and offerings, S3perf allows the user to specify the encoding of the bucket names.

When doing a performance evaluation with S3perf, 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 S3perf 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 tool `dd` and contain pseudorandom data from `/dev/random`.

In order to be able to simulate different load scenarios, the S3perf tool supports the parallel transfer of 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`.

Every time when S3perf is executed, the tool prints out a line of data, which informs the user about the date and time when the execution was finished, the number of created objects, the size of the single objects, as well as the required time in seconds to execute the steps 1-6. The final column contains the sum of all time values, which is calculated by using the command line tool `bc`.

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

5 Performance Analysis of the Deployed Cloud Storage Services

To generate a sufficient number of measurement data, S3perf was executed for each storage service five times with ten objects of sizes 512 Byte, 1 kB, 2 kB, 4 kB, 8 kB, 16 kB, 32 kB, 64 kB, 128 kB, 256 kB, 512 kB, 1 MB, 2 MB, 4 kB and 8 MB. This caused up to 75 test runs for each storage service. Few test runs failed because the resources of the used single board computer were fully utilized or because the service did not allow to upload files of a specific size.

When analyzing the required time to upload or download files, it must be kept in mind that the Raspberry Pi 3 is equipped with a 10/100 Mbit Ethernet controller, that is internally connected with the USB hub. In practice, the network performance is less than 100 Mbit. The network performance between two Raspberry Pi 3 nodes was measured with the command-line tool `iperf` v2.0.5 [21] and with the NetPIPE v3.7.2 benchmark [40]. According to `iperf`, the network performance between the two nodes is approximately 94 Mbit per second.

A more detailed analysis of the network performance is possible with the NetPIPE benchmark. It tests the latency and throughput over a range of message sizes between two processes. The benchmark was executed between the two nodes by just using TCP as end-to-end (transport layer) protocol. The results in Figure 1 show that the smaller a message is, the more dominant the transfer time by the communication layer overhead is. For larger messages, the communication rate becomes bandwidth limited by a component in the communication subsystem. Examples are the data rate of the network link, the utilization of the transmission medium or a specific device between the sender and the receiver like the used network switch.

5.1 Step 1: Create a Bucket

The creation of a single bucket (Step 1) cannot be executed in parallel and it is not influenced by the size of the objects, which are (in later steps) created and transferred by S3perf. Therefore, the boxplot in Figure 2, which presents the statistical distribution of the required time [s], does not mention any object sizes. The horizontal line inside each box is the median (second quartile). The smallest 50% of the points are smaller than or equal to the median value. 25% of the points are smaller than or equal to the bottom box boundary (first quartile) and 25% of the points are bigger than or equal to the top box boundary (third quartile).

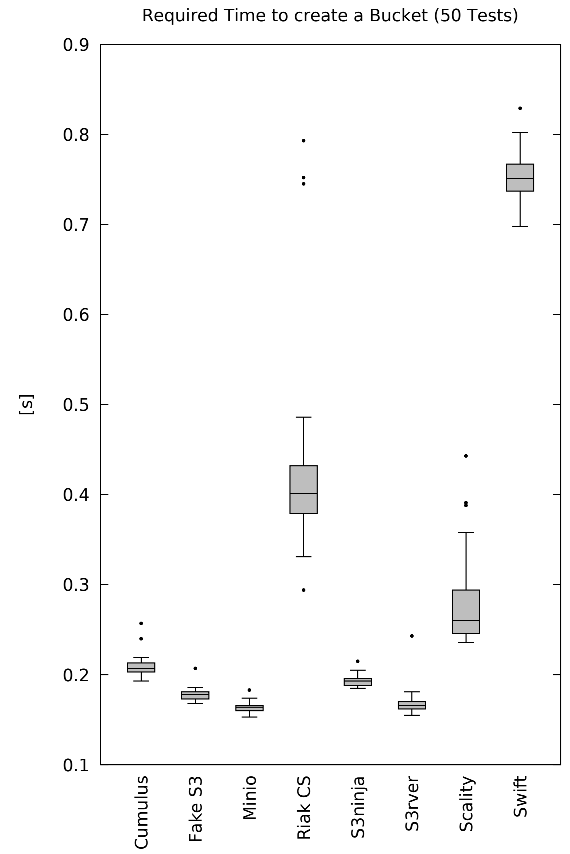


Figure 2: Time to create a bucket inside the tested storage services

The size of the box is also called interquartile range (IQR). It is calculated via ($IQR = Q3 - Q1$) and used to find outliers in the data. The whiskers extend from the ends of the box to the most distant point whose y-axis value has a maximum distant of 1.5 times the IQR. Figure 2 does not show all outliers. In few cases, creating a bucket with Riak CS and Swift required around 5 seconds.

Figure 2 shows that Riak CS and Swift require more time to create a bucket compared with the other service solutions. This is probably caused by their more complex way to store the data with replicas, which consumes additional resources, but in principle also offers the option to build systems with a better level of reliability.

5.2 Step 2: Upload the Objects into the Bucket

The required time to upload ten objects sequentially and in parallel into the tested storage services (Step 2) is presented in Figure 3. The measurements show that the parallel upload of the ten objects only in few cases lead to a significant acceleration compared with the sequential upload. In most cases, the period to upload the objects in a parallel way is longer compared to the sequential upload. The parallel upload is significant beneficial when using Fake S3 with objects $> 512\,\text{kB}$, Minio with objects $> 1024\,\text{kB}$ and $< 8192\,\text{kB}$, Riak CS with objects $> 256\,\text{kB}$ and S3ninja with objects $> 32\,\text{kB}$.

From the evaluated services, Riak CS is the most main memory consuming one and it does not free the main memory after an upload operation. After the upload of ten objects in parallel with a size of $2048\,\text{kB}$ each, only approximately 170 MB of main memory have been available on the Raspberry Pi 3. The attempt to upload ten objects in parallel with a size of $4096\,\text{kB}$ crashed the node. When trying to upload ten objects in serial with a size of $8192\,\text{kB}$, the main memory of the node gets entirely filled and also the Swap storage is used. In effect, the node also crashed during this attempt.

The upload of objects $> 1024\,\text{kB}$ into S3ninja failed because `nginx`, which was used as a proxy, did not permit the upload of such large files. Another observation with S3ninja was, that the service consumed one CPU core entirely at times, during the upload of ten objects (in parallel and sequential operation mode).

Also Swift consumes a lot of main memory because of the way, the objects, buckets and accounts are stored in rings with replicas. When deploying Swift on server resources, the replicas are distributed over multiple physical storage devices. On the Raspberry Pi 3 node, all replicas are stored on the microSD card, which is also used to store the operating system.

5.3 Step 3: Fetch the List of Objects

Fetching a list of objects (Step 3) cannot be executed in parallel and should not actually depend on the size of the objects, but as presented in Figure 4, when using S3ninja, the size of the objects has a strong influence on the required time. The root cause for this behavior may be the working method of the Java VM used.

It is also observed in this step that Riak CS and Swift provide a lesser performance compared with the other investigated solutions (except S3ninja) because Riak CS and Swift both store the objects with replicas.

5.4 Step 4: Download the Objects

The required time to download ten objects sequentially and in parallel into the tested storage services (Step 4) is presented in Figure 5. The measurements show that the parallel download of the ten objects does only in few cases lead to a significant acceleration compared with the sequential upload. In most cases, the period to download the objects in a parallel way is longer compared to the sequential download. The parallel upload is significant beneficial when using Cumulus with objects $> 1\,\text{kB}$ and $< 2048\,\text{kB}$ and S3ninja with objects $> 16\,\text{kB}$.

5.5 Step 5: Erase the Objects

The required time to erase ten objects sequentially and in parallel with the tested storage services (Step 5) is presented in Figure 6. The measurements show that erasing the ten objects in parallel does only in few cases lead to a significant acceleration compared with the sequential upload. In most cases, the period to erase the objects in a parallel way is longer compared to the sequential operation mode. Erasing in parallel is beneficial when using S3ninja with objects $> 8\,\text{kB}$.

Erasing objects should not actually depend on the size of the objects, but as presented in Figure 6, when using S3ninja, the size of the objects has a strong influence on the required time. The root cause for this observation may be the working method of the Java VM used.

5.6 Step 6: Erase the Bucket

Erasing a single bucket (Step 6) cannot be executed in parallel and because the bucket is already empty, the performance of this operation is not be influenced by the size of any objects. Therefore, the boxplot in Figure 7, which presents the statistical distribution of the required time [s], does not mention any object sizes.

The Figure does not show all outliers. In few cases, erasing a bucket with Riak CS required around 8 seconds and with Swift around 5 seconds.

6 Conclusions and Next Steps

The performance of single board computers like the Raspberry Pi 3 cannot compete with higher-value systems because the characteristics of their relevant hardware components, especially the CPU, main memory, storage and network interface. Regardless of the performance, single board computers are useful for academic purposes and research projects because of the lesser purchase costs and operating costs compared with commodity hardware server resources. The hardware resources of a single Raspberry PI 3 node are sufficient to deploy and use each one of the S3-API-compatible

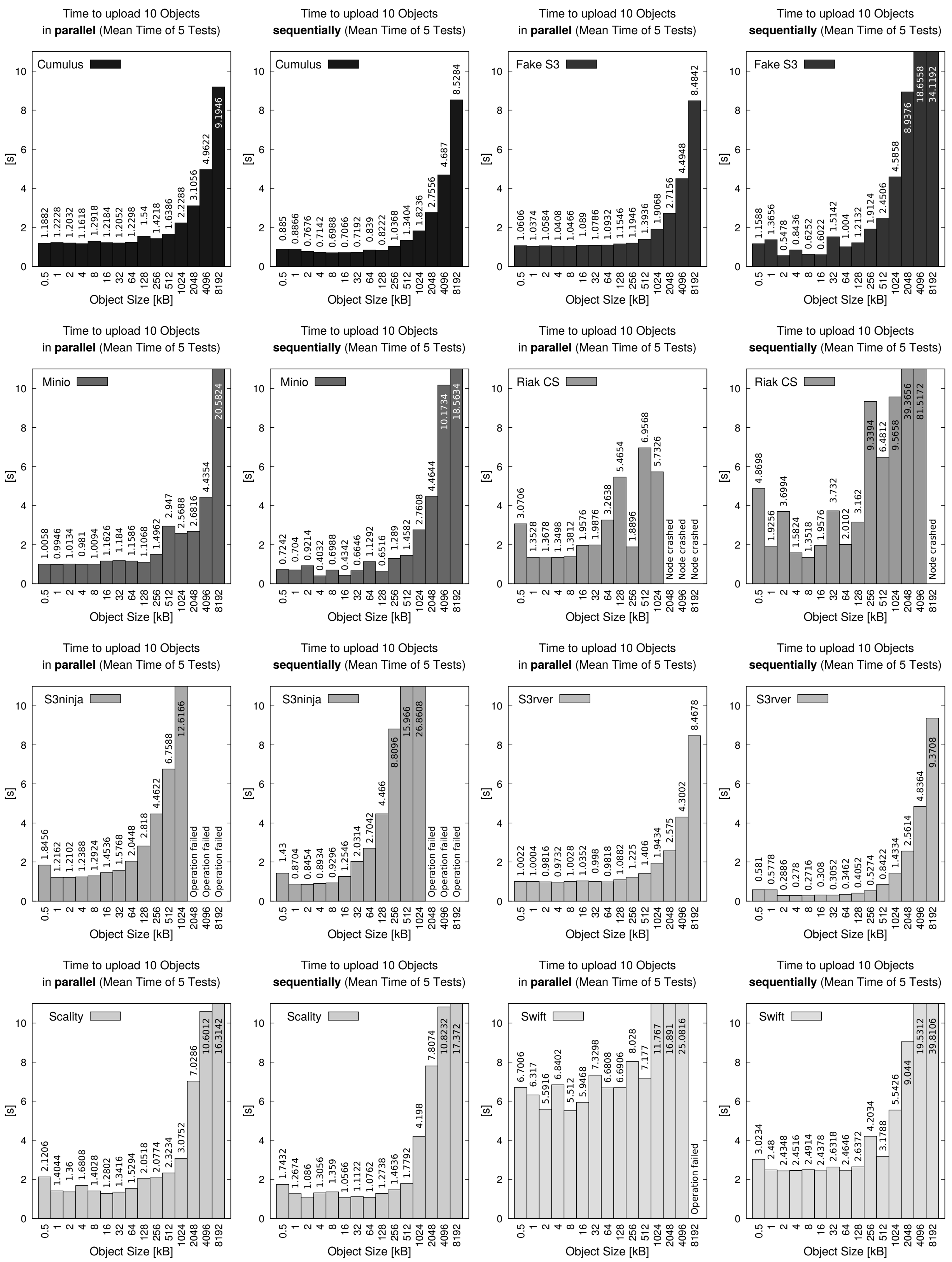


Figure 3: Time to upload ten objects sequentially and in parallel into the tested storage services

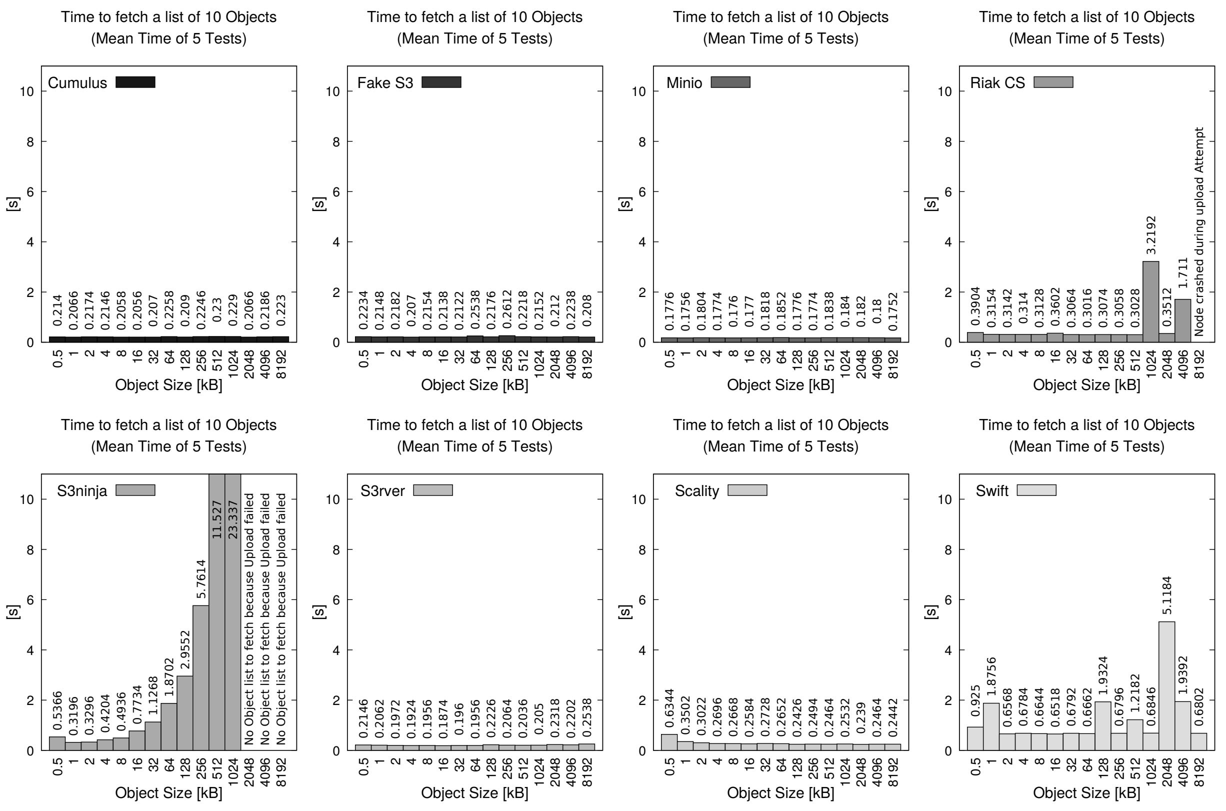


Figure 4: Time to fetch the list of objects (more detailed view)

storage service solutions, which have been examined in this work.

In order to investigate the performance of the analyzed storage services, a set of typical operations has been picked out and a lightweight tool, called S3perf has been developed and implemented. This tool was used to carry out the set of operations and measure the required time. Based on these measurements, the performance of the storage services on a single board computer has been analyzed.

Results are among others that the service solutions Cumulus, Fake S3, Minio, S3ninja, S3rver and Scality for most operations provide a better performance compared with Riak CS and Swift. The reason for this observation is probably caused by the way Riak CS and Swift store the objects and bucket data with replicas and organize them via Distributed Hash Tables. Riak CS and Swift both focus to work efficiently inside multi-node deployments and are not in first place designed to provide a strong performance inside single node scenarios. Another observation was that the parallel execution of multiple upload, download and erase operations just in few scenarios are beneficial for the overall performance. Especially, when using objects of 1 MB and more, the maximum throughput of the network interface of the Raspberry Pi 3 becomes a limiting factor. Furthermore, it has been observed that when using S3ninja, the performance of list and delete operations depend on the size of the objects.

Next steps are the deployment of Ceph-RGW, Minio, Riak CS and Swift inside multi-node systems of Raspberry Pi 3 nodes and an analysis of their performance and robustness.

For future work, it is also useful to analyze the total cost of ownership (TCO) of multi-node systems of single board computers which host storage services. As proven among others by Zhao et al. [48], the TCO of single board computer clusters can be better compared with higher-value systems.

Next steps also comprise an analysis of the maximum workload and storage capacity of the described systems in order to archive more information about the practical usability of storage services which are deployed on single board computer clusters.

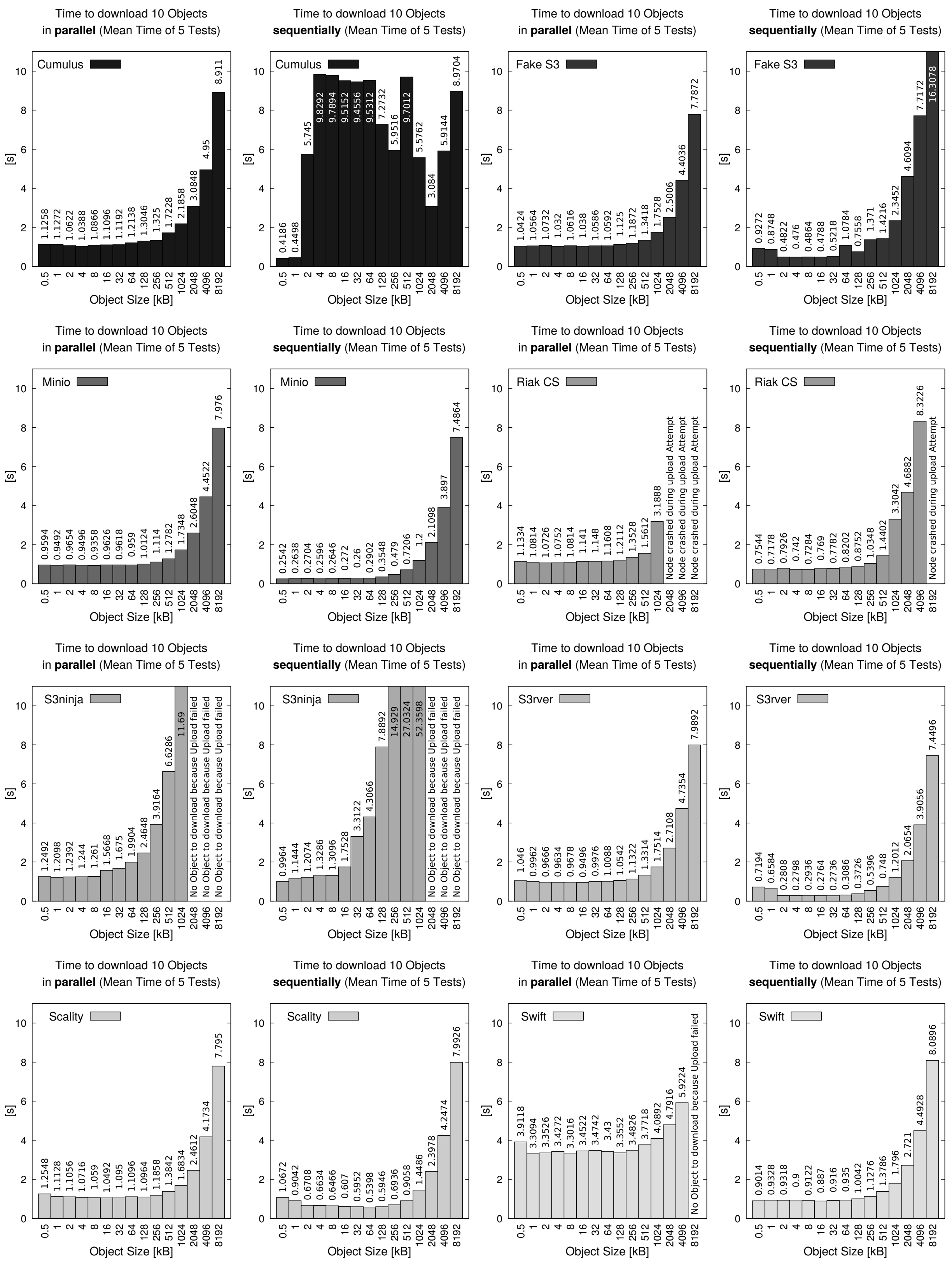


Figure 5: Time to download ten objects sequentially and in parallel into the tested storage services

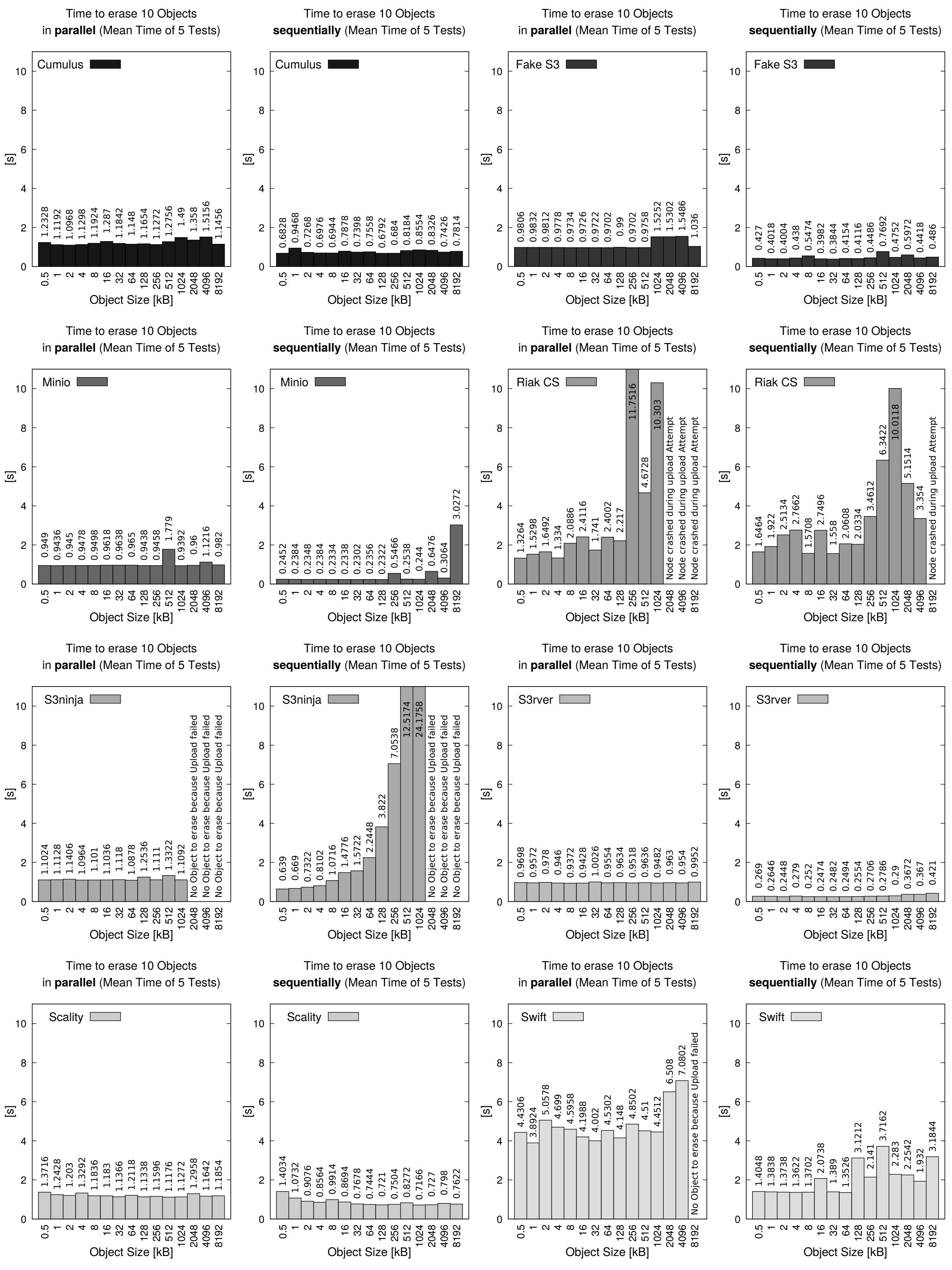


Figure 6: Time to erase ten objects sequentially and in parallel into the tested storage services

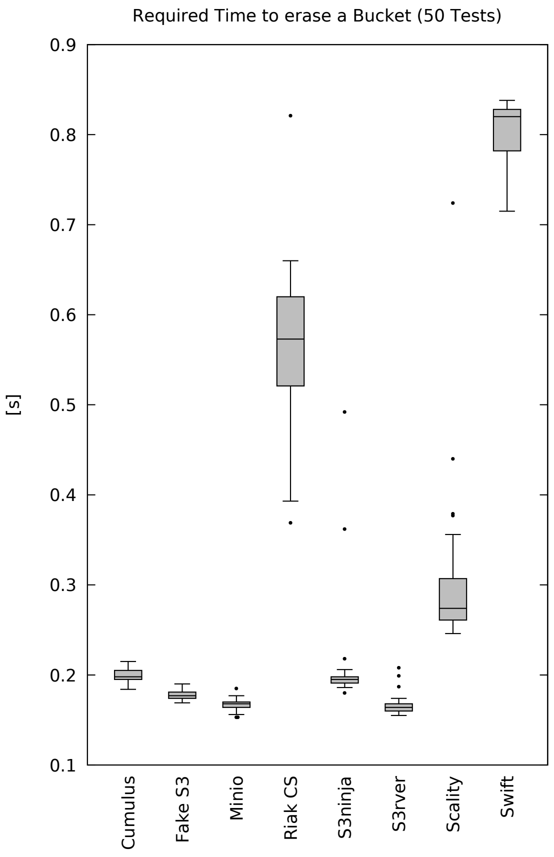


Figure 7: Time to erase the bucket

ACKNOWLEDGEMENTS

This work is funded by the Hessian Ministry of Higher Education, Research, and the Arts ('*Hessisches Ministerium für Wissenschaft und Kunst*') in the framework of research for practice ('*Forschung für die Praxis*').

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

REFERENCES

[1] B. Apperson, "Building a ceph cluster on raspberry pi," http://bryanapperson.com/blog/the-definitive-guide-ceph-cluster-on-raspberry-pi/, accessed 12th August 2017.

[2] C. Baun, "Mobile clusters of single board computers: an option for providing resources to student projects and researchers," *SpringerPlus*, vol. 5, no. 1, 2016.

[3] C. Baun, "Performance and energy-efficiency aspects of clusters of single board computers," *International Journal of Distributed and Parallel Systems*, vol. 7, no. 2/3/4, 2016.

[4] C. Baun, M. Kunze, D. Schwab, and T. Kurze, "Octopus-a redundant array of independent services (rais)." in *CLOSER 2013: Proceedings of the 3rd International Conference on Cloud Computing and Services Science*. SCITEPRESS, 2013, pp. 321–328.

[5] C. Baun and R.-M. Spanou, "s3-perf source code," https://github.com/christianbaun/s3perf, accessed 12th August 2017.

[6] Z. Bjornson, "Aws s3 vs google cloud vs azure: Cloud storage performance," http://blog.zachbjornson.com/2015/12/29/cloud-storage-performance.html, accessed 12th August 2017.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[8] Boto, "Boto source code and documentation," https://github.com/boto/boto, accessed 12th August 2017.

[9] J. Bresnahan, K. Keahey, D. LaBissoniere, and T. Freeman, "Cumulus: An Open Source Storage Cloud for Science," in *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM, 2011, pp. 25–32.

[10] N. Canceill, W. de Jong, and J. Saathof, "A study of scalable storage systems," SURF, Tech. Rep., 2014.

[11] Ceph, "Ceph Object Gateway," http://docs.ceph.com/docs/master/radosgw/, accessed 12th August 2017.

[12] COSBench, "Cloud object storage benchmark – source code and documentation," https://github.com/intel-cloud/cosbench, accessed 12th August 2017.

[13] J. Coyle, "Create a 3 node ceph storage cluster," https://www.jamescoyle.net/how-to/1244-create-a-3-node-ceph-storage-cluster, accessed 12th August 2017.

[14] J. Coyle, "Small scale ceph replicated storage," https://www.jamescoyle.net/how-to/2105-small-scale-ceph-replicated-storage, accessed 12th August 2017.

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin,

S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[16] Eucalyptus, "Eucalyptus source code and documentation," https://github.com/eucalyptus/eucalyptus, accessed 12th August 2017.

[17] Fake S3, "Fake S3 source code and documentation," https://github.com/jubos/fake-s3, accessed 12th August 2017.

[18] S. Garfinkel, "An evaluation of amazon's grid computing services: Ec2, s3, and sqs," *Harvard Computer Science Group Technical Report TR-08-07*, 2007.

[19] D. Guinard and V. Trifa, *Building the web of things: with examples in node. js and raspberry pi*. Manning Publications Co., 2016.

[20] HPL, "A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers," http://www.netlib.org/benchmark/hpl/, accessed 12th August 2017.

[21] Iperf, "Iperf source code and documentation," http://www.nwlab.net/art/iperf/, accessed 12th August 2017.

[22] P. Jain, A. Goel, and S. Gupta, "Monitoring of riak cs storage infrastructure," *Procedia Computer Science*, vol. 54, pp. 137–146, 2015.

[23] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 1997, pp. 654–663.

[24] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky computing," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 43–51, Sept 2009.

[25] Y. N. Krishnan, C. N. Bhagwat, and A. P. Utpat, "Fog computingnetwork based cloud computing," in *Electronics and Communication Systems (ICECS), 2015 2nd International Conference on*. IEEE, 2015, pp. 250–251.

[26] L. Land, "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 12th August 2017.

[27] R. McFarland, "s3-perf source code," https://github.com/ross/s3-perf, accessed 12th August 2017.

[28] Minio, "Minio source code and documentation," https://github.com/minio/minio, accessed 12th August 2017.

[29] A. Nayak, A. Poriya, and D. Poojary, "Type of nosql databases and its comparison with relational databases," *International Journal of Applied Information Systems*, vol. 5, no. 4, pp. 16–19, 2013.

[30] Nimbus, "Nimbus source code and documentation," https://github.com/nimbusproject/nimbus, accessed 12th August 2017.

[31] OpenStack Storage (Swift), "Swift source code and documentation," https://github.com/openstack/swift, accessed 12th August 2017.

[32] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in *Proceedings of the 2008 international workshop on Data-aware distributed computing*. ACM, 2008, pp. 55–64.

[33] Python client for the Swift API, "Source code and documentation," https://github.com/openstack/python-swiftclient, accessed 12th August 2017.

[34] Riak CS, "Riak CS source code and documentation," https://github.com/basho/riak_cs, accessed 12th August 2017.

[35] s3cmd, "s3cmd source code and documentation," https://github.com/s3tools/s3cmd, accessed 12th August 2017.

[36] S3ninja, "S3ninja source code and documentation," https://github.com/scireum/s3ninja, accessed 12th August 2017.

[37] S3rver, "S3rver source code and documentation," https://github.com/jamhall/s3rver, accessed 12th August 2017.

[38] Scality S3 Server, "S3 Server source code and documentation," https://github.com/scality/S3, accessed 12th August 2017.

[39] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing," *Open Journal of Cloud Computing (OJCC)*, vol. 2, no. 1, pp. 16–24, 2015. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201705194519

[40] Q. O. Snell, A. R. Mikler, and J. L. Gustafson, "Netpipe: A network protocol independent performance evaluator," in *IASTED International*

Conference on Intelligent Information Management and Systems, vol. 6. USA, 1996.

[41] Stanchion, "Stanchion source code and documentation," https://github.com/basho/stanchion, accessed 12th August 2017.

[42] Swift3, "Swift3 middleware for openstack swift - source code and documentation."

[43] N. Tiwari, "Enterprise-grade cloud storage with nginx plus and minio," https://www.nginx.com/blog/enterprise-grade-cloud-storage-nginx-plus-minio/, accessed 24th June 2017.

[44] R. Vijayan, "Ceph on a raspberry pi," https://www.linkedin.com/pulse/ceph-raspberry-pi-rahul-vijayan, accessed 12th August 2017.

[45] Y. Wang, "Definition and categorization of dew computing," *Open Journal of Cloud Computing (OJCC)*, vol. 3, no. 1, pp. 1–7, 2016. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201705194546

[46] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.

[47] C. W. Zhao, J. Jegatheesan, and S. C. Loon, "Exploring iot application using raspberry pi," *International Journal of Computer Networks and Applications*, vol. 2, no. 1, pp. 27–34, 2015.

[48] Y. Zhao, S. Li, S. Hu, H. Wang, S. Yao, H. Shao, and T. Abdelzaher, "An experimental evaluation of datacenter workloads on low-power embedded micro servers," *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 696–707, 2016.

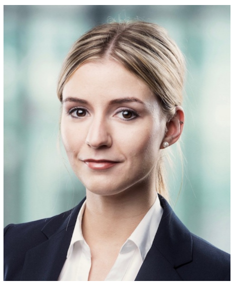
[49] Q. Zheng, H. Chen, Y. Wang, J. Duan, and Z. Huang, "Cosbench: A benchmark tool for cloud object storage services," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 998–999.

[50] Q. Zheng, H. Chen, Y. Wang, J. Zhang, and J. Duan, "Cosbench: cloud object storage benchmark," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 199–210.

Author Biographies

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.