ALMA MATER STUDIORUM – UINVERSITY OF BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

*DISI*

*COMPUTER ENGINEERING*

**MASTER THESIS**

In

Distributed Systems M

# HIGH-PERFORMANCE PERSISTENT CACHING IN MULTI- AND HYBRID- CLOUD ENVIRONMENTS

CANDIDATE
Luca Chiossi

SUPERVISIOR
Prof. Paolo Bellavista

CO-SUPERVISOR
Dr. Christian Pinto

Academic Year 2018/19

Session III

# ABSTRACT (Italian)

Il modello di lavoro noto come Multi Cloud sta emergendo come una naturale evoluzione del Cloud Computing per rispondere alle nuove esigenze di business delle aziende. Un tipico esempio è il modello noto come Cloud Ibrido dove si ha un Cloud Privato connesso ad un Cloud Pubblico per consentire alle applicazioni di scalare al bisogno e contemporaneamente rispondere ai bisogni di privacy, costi e sicurezza.

Data la distribuzione dei dati su diverse strutture, quando delle applicazioni in esecuzione su un centro di calcolo devono utilizzare dati memorizzati remotamente, diventa necessario accedere alla rete che connette le diverse infrastrutture. Questo ha grossi impatti negativi su carichi di lavoro che consumano dati in modo intensivo e che di conseguenza vengono influenzati da ritardi dovuti alla bassa banda e latenza tipici delle connessioni di rete.

Applicazioni di Intelligenza Artificiale e Calcolo Scientifico sono esempi di questo tipo di carichi di lavoro che, grazie all'uso sempre maggiore di acceleratori come GPU e FPGA, diventano capaci di consumare dati ad una velocità maggiore di quella con cui diventano disponibili.

Implementare un livello di cache che fornisce e memorizza i dati di calcolo dal dispositivo di memorizzazione lento (remoto) a quello più veloce (ma costoso) dove i calcoli sono eseguiti, sembra essere la migliore soluzione per trovare il compromesso ottimale tra il costo dei dispositivi di memorizzazione offerti come servizi Cloud e la grande velocità di calcolo delle moderne applicazioni.

Il sistema cache presentato in questo lavoro è stato sviluppato tenendo conto di tutte le peculiarità dei servizi di memorizzazione Cloud che fanno uso di API S3 per comunicare con i clienti. La soluzione proposta è stata ottenuta lavorando con il sistema di memorizzazione distribuito Ceph che implementa molti dei servizi caratterizzanti la semantica S3 ed inoltre, essendo pensato per lavorare su ambienti Cloud si inserisce bene in scenari Multi Cloud.

# INDEX

**PREFAX** .......... 1
**INTRODUCTION** .......... 2
**1 THE CLOUD COMPUTING** .......... 4
**1.1 AN INTRODUCTION TO DISTRIBUTED SYSTEMS** .......... 4
**1.2 CLOUD COMPUTING CONCEPTS** .......... 7
**1.3 A GENERAL CLOUD COMPUTING ARCHITECTURE AND THE NEED FOR STANDARDS** .......... 9
**1.4 CROSS CLOUD COMPUTING ENVIROMENTS** .......... 13
**1.5 EXECUTION OF ARTIFICIAL INTELLIGENCE AND SCIENTIFIC COMPUTING WORKLOADS IN HYBRID- AND MULTI- CLOUD COMPUTING ENVIRONMENTS** .......... 15
**2 DATA STORAGE SYSTEMS** .......... 17
**2.1 POSIX STANDARD AND PORTABILITY** .......... 17
**2.2 OVERVIEW OF FILE SYSTEMS KEY CONCEPTS** .......... 19
**2.3 FILE SYSTEMS INTERACTION AND POSIX** .......... 22
**2.4 STORAGE NETWORKING TECHNOLOGIES** .......... 27
**2.5 NETWORK FILE SHARING AND MAIN FACTORS AFFECTING PERFORMANCE USING IP NETWORKS** .......... 30
**2.6 THE OBJECT STORAGE** .......... 33
**2.7 A UNIFIED VIEW: BLOCKS, FILES AND OBJECTS** .......... 35
**3 STORAGE TECHNOLOGIES AT THE STATE OF THE ART** .......... 38
**3.1 THE CACHE MEMORY IN INTELLIGENT STORAGE SYSTEMS** .......... 38
**3.2 FILE SYSTEMS AT THE STATE OF THE ART** .......... 43
**3.3 THE S3 CLOUD OBJECT STORAGE** .......... 48
**3.4 S3 REST API, SERVICES PRICING AND BEST PRACTICES** .......... 53
**3.5 THE POWER OF FUSE – AN INTERESTING EXAMPLE WITH S3** .......... 59
**4 CEPH AND CACHING** .......... 63
**4.1 CEPH – AN OPEN SOURCE AND DISTRIBUTED UNIFIED STORAGE SYSTEM** .......... 63
**4.2 CEPHFS DEPLOYMENT AND LIBCEPHFS OVERVIEW** .......... 69

4.3 PLUG IN DEVELOPMENT – THE POWER OF FUSE AND POSIX ....... 71
4.4 CEPH'S RADOSGW IN DETAIL ....... 75
5 S3 CACHE LAYER FOR HYBRID- AND MULTI- CLOUD ENVIRONMENTS ....... 80
5.1 RGW FOR S3 CACHING – WHY? HOW? ....... 80
5.2 S3 CACHE AUTHENTICATION MANAGEMENT AND PRE-IMPLEMENTATION CONSIDERATIONS ....... 86
5.3 GETOBJECT ....... 91
5.4 PUTOBJECT ....... 94
5.5 LISTOBJECTS ....... 98
5.6 POST-IMPLEMENTATION CONSIDERATIONS ....... 100
6 EXPERIMENTS AND RESULTS ....... 103
6.1 BENCHMARKS AND EXPERIMENTS' INFRASTRUCTURE ....... 103
6.2 CACHE GETOBJECT PERFORMANCE WITH THE S3 CUSTOM BENCHMARK ....... 106
6.3 CACHE PUTOBJECT PERFORMANCE WITH THE S3 CUSTOM BENCHMARK ....... 111
6.4 CACHE GETOBJECT PERFORMANCE DURING TRAINING SESSIONS OF TENSORFLOW'S DNN RESNET OVER THE IMAGENET DATASET ....... 116
CONCLUSIONS ....... 123
REFERENCES ....... 126
IMAGES ....... 130
TABLES AND CHARTS ....... 132
ACKNOWLEDGMENTS ....... 133

# PREFAX

Computers are the most relevant innovation of our times. We can find them in different forms and we interact with them in almost every situation and aspect of our lives. This document for example has been written on a normal laptop and it wouldn't be surprising if in the pocket of the reader we could find a modern smartphone. Computers are the result of the combination of many different innovations which have been held by incredible minds and visionaries during the past two centuries.

The birth of modern computers and how our life changed with them is just the last step of a process started during the first half of the $19^{th}$ century when Ada Lovelace spoke for the first time about programming and computation. Many other brilliant innovators had an important role such as Vannevar Bush, Alan Turing, John von Neuman, J.C.R Licklider, Doug Engelbart, Robert Noyce, Bill Gates, Steve Wozniak, Steve Jobs, Tim Berners-Lee and Larry Page just to mention some of them.

The unicity in the invention of computers in my opinion is the fact that they are not the result of a single idea but instead the combination of many intuitions of scientists from a wide and heterogeneous range of disciplines. They express the continuous seek of human kind to go beyond its limits to improve people lives or for the simple please of curiosity. Research does not come from the necessity to satisfy specific needs but from the human nature of seeking knowledge. It is thanks to this knowledge if we now have what we have and we must not forget that many of the technologies we depend on nowadays have been possible only thanks to the result of research.

With this short introduction I want to thank all the academic sector, the scientific community and researchers from all around the World for the effort they put on their work every day.

With next sections it is presented the work of my Master Degree Thesis, for the course in Computer Engineering taken at University of Bologna, which I had the pleasure to work on at the laboratories of IBM Research Ireland.

# INTRODUCTION

The Multi Cloud paradigm is emerging as a natural evolution of Cloud Computing to respond to the needs of scaling workloads beyond the boundaries of a single Cloud while satisfying constraints of cost, performance, security and privacy. In Multi- Clouds two or more Cloud data centers are interconnected and their resources used in conjunction for storage and computation. A typical case is that of a Public Cloud connecting to private data center of an enterprise (Private Cloud) whose applications are running in synergy across the two sites: this scenario is commonly referred to as Hybrid Cloud.

In such scenarios data are stored across the different sites depending on business constraints, such as cost of storage and privacy constraints. When workloads from a different site need to access remote data, those need to be fetched across the wide-area network (WAN) connecting the two Clouds before they can be utilized in the destination Cloud. This has a severe negative impact on data-intensive workloads that are remarkably affected by poor data access bandwidth and latency but need to access data in distributed Multi Cloud storages.

Artificial intelligence (AI) and Scientific Computing are examples of classes of these kind of workloads. With their increasing use of accelerators (such as GPUs and FPGAs), applications like training of Deep Learning models or scientific simulations are capable of consuming quantities of data at a rate that is an order of magnitude faster than what typical WAN links can provide.

Implementing a cache layer that transparently fetches and caches data from slow (remote) cheap storage to fast (but expensive) storage close to the computation represents a solution to seek the perfect trade-off between cost of long term data storage versus computational performance. However, there are many factors that must be investigated in order to achieve an efficient solution such as for example cache write backs in high performance Multi Cloud environments.

The proposed solution is the result of an in depth study of at the state of the art storage technologies and main characteristics and needs of just mentioned types of workloads during reading and writing procedures.

Distributed file systems allow the storage and availability of data across different localities thanks to the coordination of computational nodes which efficiency depends therefore on the specific implemented policies. Thanks to this kind of storage systems data can be stored over different nodes making possible to obtain very good levels of dynamicity, flexibility and reliability. Because of these reasons the choice of performing caching operation using the distributed storage technology Ceph has been made.

Data information can be transferred in different ways according to the specific type of the adopted storage technology. The POSIX API represents the most diffused file interaction as it is the standard for the most modern file systems but it is not the only one. There are new emerging prominent storage technologies with different types of interaction such as for example those defining the S3 API. They allow data transmission to/from Cloud storage services provided as IaaS over the Internet with the HTTP protocol. This is mainly due to the new application domains introduced by the Cloud Computing paradigm.

Ceph, in addition to the previously mentioned characteristics, allows data I/O interactions with different semantics such as POSIX and S3 which are standards for the most Cloud services subject of this study.

In the first part an overview of previously mentioned technologies and how they ca be implied in the realization of caching solutions in the context of Hybrid- and Multi- Clouds will be presented.

In the second part the design and implementation choices made during the development of a cache layer for S3 objects with the Ceph technology will be presented.

In the last section a detailed analysis of cache's performance with different workloads will be provided.

# 1 THE CLOUD COMPUTING

## 1.1 AN INTRODUCTION TO DISTRIBUTED SYSTEMS

Computers are Hardware and Software components that are able to compute and process algorithms. In terms of computational power, every computer has many different characteristics which impact performance. Computer Networks are a set of different interconnected computers capable of communicating between them. The main example is the Internet and the associated World Wide Web [1].

Distributed systems are systems in which the components (Hardware and Software) of many networked computers can communicate and coordinate their actions via the only exchange of messages. They introduce some concerns like concurrency, absence of a global clock and single point of failure. There are indeed new challenges that must be faced when working with Distributed Systems which can therefore lead to powerful, efficient and reliable computation solutions if appropriately deployed. There are many examples of these systems with which we interact every day like web search, email services, online gaming, social networks, etc.

These systems are very common in modern technology trends and industrial scenarios. During last years the most important innovations on computational technologies are related to them and because of this it is very important to understand the theories behind them and how they work to have a complete view of modern computational paradigms.

Examples of the main factors and trends leading nowadays research in Distributed Systems are for instance the emergence of pervasive networking technologies, the emergence of ubiquitous computing and mobile systems, the increasing demand for multimedia services and the view of Distributed Systems as a utility. The last mentioned factor is very important and has a central role when we speak about Cloud Computing.

The modern Internet is a global collection of many different interconnected Computer Networks which are distinguished by their locality, type of communication and number of

devices. As a result, we have many different kind of networks, including a wide range that uses wireless communication such as Wi-Fi, Bluetooth or new generation mobile phone networks. The need for an efficient way to address all these systems and devices is just a natural consequence of their heterogeneity as shown in the following pictures.

*Image 1.1 - Heterogeneous devices interacting with the Internet*

The continuous technology development in electronic engineering has made it possible to produce devices of very small dimensions. This, along with the improvements in wireless communications, has made it possible to introduce mobile devices into Distributed Systems. Examples of these devices are laptops but also smartphones, wearable devices like smartwatches and embedded devices that make Mobile Computing possible.

The term Mobile Computing represents the possibility to perform computational tasks with devices which are not constrained to one single physical location but capable of working while moving. In such a scenario the user has access to many different Hardware and Software resources while fair away from its local intranet. An extension of this paradigm is the Ubiquitous Computing where the user interacts with many different devices available in a specific environment. These devices are so pervasive in people everyday life that become transparent to the users that, stop to focus on the usage of these devices as they become a natural extension of what they are doing as much as it can be a pen when they want to handwrite.

Multimedia Systems are very important and of particular interest in Distributed Systems. It isn't the goal of this work to deepen in these particular technologies but it is interesting

to see that they are capable of making multimedia resources available to remote devices real time and on-demand taking advantage of Distributed Systems properties.

Thanks to the maturity reached by Distributed Systems companies are now selling computing and storage capacities as services in the same way as it is normally done for other services like water or electricity. Distributed Systems can therefore be seen as a utility offered by companies to clients who do not buy products but rent resources (Software and Hardware) of company's remote data centers. Physical resources like computation, storage or network are made available to customers that do not have to buy Hardware on their premises anymore to take advantage of their computational power. Users may therefore decide to use a remote storage to save their files or backups as well as they now may decide to perform workloads tasks on remote computational nodes while accessing sophisticated data centers which also allows them to perform Distributed Computation.

Operating System virtualization techniques have a key role in this kind of services. Clients have access to virtual resources rather than physical nodes making Cloud Providers able to perform an optimized management of data centers resources and as a consequence to offer the best service as possible to customers.

Following the same approach Software Services can be offered to clients in Distributed environments enhancing performance of execution. With applications and infrastructures already available to be used, companies may also decide to redirect their effort on the usage of these applications rather than to their development, with relevant impacts in development times and costs.

## 1.2 CLOUD COMPUTING CONCEPTS

With the term of Cloud Computing we refer to the possibility to see computation as a service that can be offered through the Internet. Cloud systems typically offer sets of storage, computing and internet-based applications as services with the aim to go beyond clients on premises resources requirements and limitations. Cloud users can now benefit of the power of Cloud infrastructures with the only need of a Web interface to access them. This is very powerful from the prospective of Operating Systems as the abstraction level offered by Web Interfaces makes it possible to access Cloud services independently from the specific types of on-premises resources and remote data centers infrastructures.

Computers Clusters provide High Performance Computing (HPC) capability as they are the aggregation of many interconnected computers which cooperate together as a single point of computation. They represent the typical implementation of data centers which are the physical abstraction of Cloud Computing Systems. The final goal of Computers Clusters is to implement these kinds of Cloud Services in order to offer them in the market over the Internet.

There are indeed many technologies involved in the creation of this new computational paradigm. Some of the state-of-the-art techniques are [2]:

- Virtualization technologies: They partition Hardware making computing platforms flexible and scalable with the opportunity to share resources between different and heterogeneous services. Moreover, many tenants can access Cloud resources simultaneously increasing infrastructures usage rate and reducing services costs.
- Orchestration of service flows and workflows: Clouds should be able to orchestrate services from different sources of different types to form services flows and workflows transparently for users.
- Web service and SOA (Service Oriented Architecture): Services are exposed on the internet through the usage of web interfaces while the internal orchestration and organization can be managed as SOA.

- The Web 2.0: It improves interconnectivity and interactivity of Web Applications.
- The worldwide distribution of storage systems: Network storage services offered on the Internet like Google File System or Amazon S3 are typically backed on distributed storage providers (data centers for example). The way data is stored and managed over data centers' infrastructures is transparent from the user perspective.
- Programming models: Some Cloud models should be modified to adapt better to Cloud infrastructures. Map Reduce is an example of computational paradigm which takes advantage of the distributed characteristic of data centers in order to execute Big Data applications.

It is possible to identify many benefits from the usage of Cloud Computing inside a company. The business model change as the effort can be redirected to the usage of already available resources rather than to the infrastructure implementation [3]. This also allows owners to save on system-administration costs as it is not needed anymore to manage local resources.

## 1.3 A GENERAL CLOUD COMPUTING ARCHITECTURE AND THE NEED FOR STANDARDS

During the past years many Distributed Computing technologies have been proposed by the scientific community but only Cloud Computing have made it possible to systems-integrators and mash-up technologies to undertake their business with a minimal investment on the infrastructure development [4]. For a new Cloud user, it can be difficult to navigate between all the Cloud technologies offered over the Internet but some architectural guidelines may help. It is here presented for this reason a possible general architectural view of a Cloud System.

The taxonomy which better helps to describe how the Cloud is composed is the Everything-as-a-Service (XaaS) which characterizes it as a stack of different kinds of services as shown in the following picture. More precisely the principal groups of services are the so called: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS).

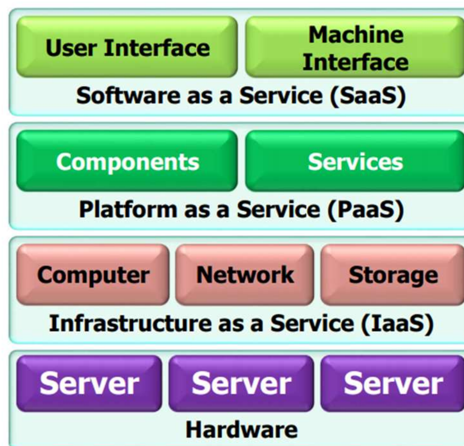



*Image 1.2 - Representation of the Cloud Services and Cloud infrastructures Stack*

On the lowest level of this stack we have the IaaS whose services can be characterized into two main groups: Physical Resources or Virtual Resources. As mentioned before, virtualization technologies offer many benefits to vendors from a management point of view but there are also some reasons why clients may decide to rent entire physical resources. An example of these scenarios may be the one where clients decide to take

advantage of Cloud Services but do not will to pay the overhead introduced by virtualization hypervisors or even more because they do not want to share the same infrastructures with other tenants for security reasons. On the top of the IaaS layer we can instead differentiate three different categories of applications: computational, storage and network services, all of them relying on virtualization.

The term PaaS refers to the middle level of the Cloud Stack and it is used to identify the services into Programming Environments and Execution Environments. The latter type typically includes services of the former. Examples for these categories are Google's App Engine and Microsoft's Azure.

The last layer we mentioned is the highest, the so called SaaS that includes all the applications running on the Cloud and that provides a direct real-time service to clients.

There are also some other possible kinds of services worth of noticing that are not meant to directly serve clients needs but to combine different available applications for business purposes. They do not find a collocation into the described stack as they are rather set beside it. There are some business cases where the solution to a specific problem needs resources coming from different layers and not just one. This makes it necessary to provide clients with appropriate administrative and business support in terms of resources management and costs optimization.

The scheme presented above in this section shows only a general model we can refer to while speaking about Cloud Computing. It can represent the general structure of many different Cloud vendors but this does not mean that the models of the many systems available in the market will refer to it. There are many efforts going on at the moment to makes standards for Cloud Technologies.

The NIST definition (National Institute of Standards and Technologies) of Cloud Computing tries to identify a baseline with the main aspects of Cloud services and deployments to make it easier the comparisons between already existing systems and new coming technologies. Thanks to these key concepts further discussions will be clearer [5].

The essential characteristics of a Cloud System identified by the NIST definition are:

- On-demand self-service: Clients can take advantage of computing capabilities when required without human interaction with the provider.
- Broad network access: Services are provided by heterogeneous platforms to different clients via standard mechanisms.
- Resource pooling: Resources like storage, processing, memory or network bandwidth are pooled and managed in a multi-tenant model to serve customers' needs on-demand. Remote resources are typically offered with such an abstraction that hide the real location of physical infrastructures to Cloud clients.
- Rapid elasticity: Clients can increase or decrease the number of services and rented resources at any time. This makes it possible to adjust payments to real-time needs without any waste.
- Measured service: The abstraction layer between users and Cloud infrastructures allow services providers to manage their resources doing optimizations. Both clients and providers should be in condition to transparently control and monitor resources usage.

The document includes also a service model which contains the IaaS, PaaS and SaaS that have been already discussed.

The deployment models are very important to have a view of possible Cloud applications environments. According to the NIST definition there are four main models:

- Private Cloud: The Cloud infrastructure is of private usage by a single organization, possibly composed by many different users. It can be managed by third parts or the organization itself.
- Community Cloud: The Cloud infrastructure is meant to be used by a specific community of consumers from organizations with same concerns. It can be managed by third parts or the organizations belonging to the specific community.

- Public Cloud: The Cloud infrastructure can be used by the general public. It exists on the premises of the Cloud provider which is also typically in charge of its management.
- Hybrid Cloud: The Cloud infrastructure is composed by two or more different ones that maintain their unicity while being bound together by standardized or proprietary technologies which allow application and data portability.

## 1.4 CROSS CLOUD COMPUTING ENVIROMENTS

Cloud Computing is capable of bringing many benefits from a computational point of view and because of this it is becoming more and more central in company's business even though, there are still many concerns about services portability because of the dynamicity of the Cloud market.

There are contracts like SLA (Service Level Agreement) where users and providers agree on the quality requirements that must be satisfied by the provisioned services. Despite this, one of the principal barriers of the adoption of Cloud solutions is the so called locking from long term commitments to a specific vendor.

This fear comes from the fact that Cloud vendors offer sets of APIs (typically not standard) to allow clients applications to interact with their services. During business lifetime it may happen that some services become obsolete or that a competitor becomes able to offer the same service with markedly differences in performance. Because of these reasons clients may opt to change the Cloud Provider to take advantage of different services with therefore the need to change the way of interaction as well. These changes have huge impacts on customers' business, especially in terms of costs and time because to adapt applications to a different set of API typically results in very expensive refactoring.

The need for standards in the way of interacting with Cloud services and the bound to specific vendors' solutions are the core of Cross Clouds challenges [6]. The principal categories may be summarized in four different groups:

- Hybrid Clouds: As described by the NIST they are compositions of many Clouds. In this scenario system developers aggregate different Cloud parts to build appropriate solutions for their applications which will then have to interact with various sets of API. This structure implies that it must be defined a logic to determine which Cloud part should be used and when. This logic is coupled with the application at a certain extent and can be eventually implemented as a proxy between the different systems.

A typical example is the one where a Private Cloud is connected to a Public Cloud to go beyond on premises infrastructure computational limits with the opportunity of scaling as needed and satisfying business constraints like privacy and security.

- Multi Clouds: Such as Hybrid Clouds they are made combining different autonomous Cloud systems. But in contrast, this model management can be achieved with some abstraction as it introduces a certain level of portability. Usually there is a common denominator set of APIs between the different systems which reduces the number of specialized services.
- Meta Clouds: They offer both abstraction and delegation pushing even far away the responsibility of application developers for the system management. They are typically deployed by third party brokers which offer loosely-coupled interaction as a managed service. It is brokers duty to find available resources to serve applications needs into the context of the Meta Cloud.
- Cloud Federations: In contrast to the previously mentioned models, Federated Clouds achieve distribution through prior agreements in the form of common interfaces or data formats. These efforts have been made to allow customers to work with resources and services across different vendors.

## 1.5 EXECUTION OF ARTIFICIAL INTELLIGENCE AND SCIENTIFIC COMPUTING WORKLOADS IN HYBRID- AND MULTI- CLOUD COMPUTING ENVIRONMENTS

In the previous chapters Cloud Computing key concepts and aspects have been discussed thus, at this point, its models, concerns and benefits should be clear to the reader. With this section it is now presented the specific scenario where this Thesis project work has been set, with references to Artificial Intelligence and Scientific Computing applications, Hybrid- and Multi- Cloud paradigms which are very prominent study areas for the scientific community.

Because of the computational power offered by Cloud vendors, adopting Cloud infrastructures to perform heavy workloads of applications such as of Artificial Intelligence or Scientific Computing seems a logical solution. Private Clouds are constrained to their own resources but still capable of providing efficient solutions with some peculiar benefits to companies. Public Clouds are able to serve on-demand clients' needs but there are many concerns about privacy and security as users have no direct control over their information that is managed by a third party instead. Thus, while the former allow companies to take advantage of Cloud infrastructures while not sharing sensitive information, the latter are ideally capable of offering unlimited amounts of resources and services allowing systems to scale as needed. Hybrid and Multi Clouds become therefore a natural extensions of single Clouds to overcome their limits while taking advantage of the singular benefits offered by any of them.

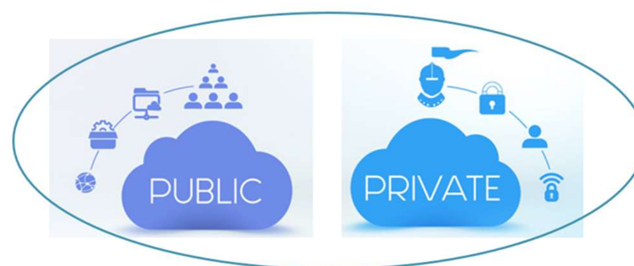



*Image 1.3 - Hybrid Cloud scenario where a Private Cloud interacts with a Public one to answer new companies' needs of privacy and elastic scaling*

As shown in image 1.3 there may be situations where a Private Cloud is connected to a Public one and their resources work in conjunction to achieve business goals. In such a scenario, which is the one of interest for this study, information is stored across different sites on distributed computational nodes. In order to be computed, data needs to be transferred using the communication channels of the (WAN) wide-area-network. This has a sever negative impacts on data-intensive workloads that are in this way limited by poor data access bandwidth and latency but need to access data in distributed Multi Cloud storages.

Artificial Intelligence and Scientific Computing applications are examples of classes of this kind of workloads. Many accelerators (like GPUs and FPGAs) make applications such as training of Deep Learning models or scientific simulations capable of consuming data at a rate which is substantially faster than what WAN links can typically provide. Because of these reasons data availability and transfer cost become the prevalent bottleneck of computational performance.

The implementation of a cache layer which transparently fetches data from a remote (slow) storage to a faster and more expensive one, closer to computational nodes seems thus to be a good solution to achieve the perfect trade-off between cost of long term data storage and computational performance. There are many different mechanisms and technologies which enable data storage and retrieval as presented in the next section.

# 2 DATA STORAGE SYSTEMS

In this section it will be given a summary of what data storage means with a particular interest for those technologies meant for Distributed Systems, the so called Distributed Information Systems. A detailed discussion of specific technologies faced during the execution of some experiments that have been performed will be better analyzed in section number 3.

## 2.1 POSIX STANDARD AND PORTABILITY

To build a cache layer for a Cross Cloud system it is necessary to have a complete and deep understanding of what storage technologies are and how they work in order to store and to retrieve data. It is very important to keep in mind that any kind of storage system has peculiar characteristics with their benefits and drawbacks and may better adapt to some cases rather than others. Moreover, the type of communication is not always the same. Different technologies typically have different sets of interaction API, especially in Cloud environments.

As it has been already said many times, standards are very important for the intercommunication and portability of different systems and applications. To build a cache layer, the first thing that must be clear is how the applications of interest are going to communicate with the storage support. For this purpose, it is here presented the POSIX standard (Portable Operating System Interfaces for Computer Environments) which describes how a POSIX compliant application or file system interacts with files. File systems which are the basic support for data storage, management and retrieval.

First computers in the history of Information Technology were characterized by different Operating Systems and programming architectures [7]. As a result, one application could not be moved from a system to another unless it was rewritten to be compatible with the different supporting infrastructure. The first real attempt in the direction of program

portability was made by IBM when it started to adopt one single architecture across many different machines. In this way programs could be executed over the different computational nodes conforming with this structure. Another step forward has been made by the Bell Labs when scientists started to work on the creation of the UNIX, an operating system capable of running over different machine of different vendors.

Nowadays there are still many battles on standardization across different operating systems but one thing is agreed by the most of them: the POSIX standards which are a set of assertions that help developers to make applications compatible between many different operating systems and architectures. A POSIX compliant application can move between different heterogeneous systems with a very low maintenance.

It is not the goal of this work to analyze the singular sections of the POSIX standard documents but, as commonly accepted by the most prominent tech vendors, to understand how it works and what it involves for data storage systems. It describes a contract between the application and the operating system. More precisely, it doesn't give guidelines for the production of the application itself or of the lower support but the way they will interact. This interaction is represented by the interfaces of the library called by the applications and by the interfaces offered to the library by the operating system. Vendors must only adapt their architectures to the POSIX library interfaces to be highly compatible and the applications thus become automatically portable and easily movable as they can work with no knowledge about the lower supports.

Another important characteristics of POSIX semantics is that it is extensible and not locking. It may happen in fact, that for some specific cases one technology will need to go beyond the POSIX guidelines. This can be done by simply adding special purposes modules, keeping in mind that they will not be compliant with all systems.

## 2.2 OVERVIEW OF FILE SYSTEMS KEY CONCEPTS

The file systems are the most basic storage technology. They are that part of an operating system which provides access and memorization mechanisms for the information (programs and data) stored on disk drives, disk partitions and logical volumes. File systems are a hierarchical organization of files which is the abstraction of the collection of information records residing on the computer memory [8]. It is their goal to provide functions, to allow users to work on files, such as creation, deletion, access, permission check or modification. They also organize and manage all the data structures that work along with files like directories. Directories are containers for storing pointers to maps of files and allow the hierarchical and structured view of stored data.

An important characteristic of file systems as type of storages is that they also hold some other records of information, the so called metadata such as (for the UNIX operating system as an example) superblocks, i-nodes and lists of free and occupied data blocks on a specific system/device.

A super block contains information about the file system such as its type or layout while an i-node maintains information related to any file and directory. A file system block is the smallest unit allocated on the physical support to store the data and may deeply influence operations' performance, especially those of reading and writing. There are cases where files are stored in many different blocks and others where they cannot be stored at all according to specific data blocks' size and number.

Some examples of file systems are:

- FAT 32 (File Allocation Table) – Microsoft Windows
- NTFS (NT File System) – Microsoft Windows
- UFS (UNIX File System) – Unix
- ExtX (Extended File System) – Linux distributions

An important aspect of file systems (as basic storage systems) worth of noticing is their capacity of abstraction between physical and logical memory. Users can think about the

files as a continuous chain of logical blocks while the physical ones are managed by modules of the operating system. This is done thanks to the evolution of Logical Volumes Managers (LVMs) that enabled the extension of file systems capacity and allowed efficient client storage management over physical infrastructures. Hard disks can be partitioned or concatenated to build a logical volume that is the abstraction of the storage memory with which the file system works. In this way once installed, a file system can work on different computational nodes at the same time while seen as a single unit of memorization.

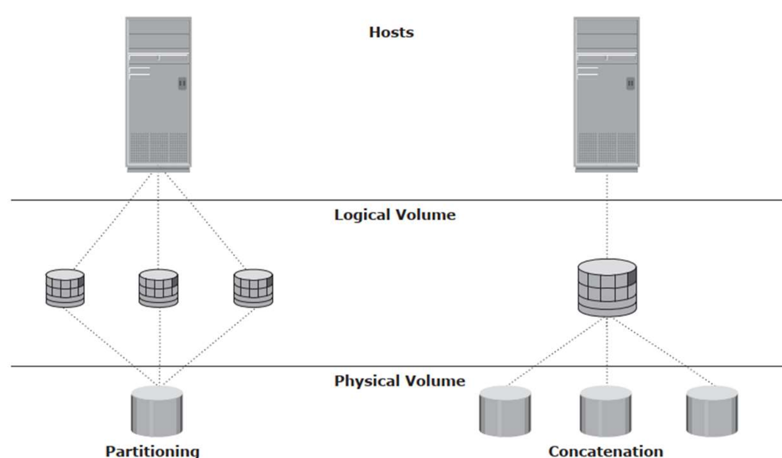



*Image 2.1 – Example of management of physical storages as virtual units*

To make a certain disk drive, disk partition or logical volume available to the operating system and so part of the general file system, the mount operation must be executed. This command indicates that the specified file system is ready to be used, associates it an address (the mount point) and sets the desired access options. This operation therefore makes the file system and its associated information such as files, directories and special files available to be accessed by the users.

It is important to keep in mind that along with the mount operation there is one more command which is very important when operating with file systems, the so called unmount operation. The managed data is not immediately written to the device when operations are called because of efficiency reasons. Files and directories are pooled and then stored all together in order to reduce the number of I/O operations with a certain device. Because of this it becomes very important to perform the unmount operation that will notify our

intentions to the operating system. It will therefore start all those procedures which deal with buffered data and metadata and no information will be lost.

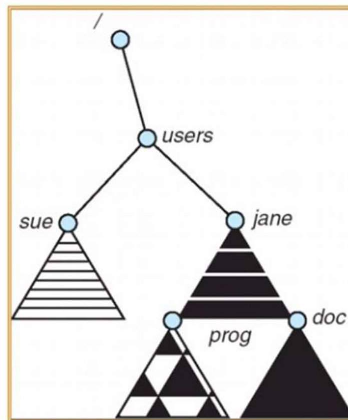



*Image 2.2 – General file system hierarchy – each triangle represents a different file system which has been made available through a call to the operating system mount operation*

As showed, file systems are an important part of operating systems as they represent the basement for storing information on physical devices and allowing users to manage and work with files. For the purpose of this work it is presented in the next section how POSIX standard characterizes file systems making them compliant between different computer systems. This is very important to seek systems interconnection.

## 2.3 FILE SYSTEMS INTERACTION AND POSIX

Whenever a certain application workload is executed there are always files involved as they are the representation of the information that is processed. The real computation is performed inside the main memory of a computational node thus, needed files (and their relative data structures) must be copied into it, on memory areas known as buffers. It is noticeable how POSIX defined standards for I/O functions. These functions must be executed in order to transfer data from a device to another.

When building portable POSIX applications, it is important to remember that the only thing that really matters is calling POSIX libraries and not knowing libraries implementation. How libraries have been implemented is just an operating system matter and may affect performances but there is not that much that developers can do about it [7].

An important thing that must be kept in mind is that file structures members (containing references to files metadata information) should never be directly accessed in portable applications. POSIX does not make any assumption about their content indeed. This is of course another matter for file systems which are meant to work directly with files and their data structures.

It is possible to work with files at high or low level. Low level functions give more control over a file but it is not always desirable to work with them as they may sometimes represent an element of possible incompatibilities in contrast to the high level ones. The key difference between their approach is the usage of a file stream or a file descriptor. These concepts will be clearer after few paragraphs.

Further discussions will be general but a precision should be made at this point. It is possible to speak about POSIX without referring to any programming language but in order to build applications it is necessary to reference one in particular. The chosen language to explain these theories is C as the main language for the most operating systems and because POSIX supports two main programming environments which both work with the C language.

Some important high level functions are: fopen(), fclose(), fwrite(), fread(), fseek(), frewind(), fscanf(), fprintf(), fflush(), setbuf() and fflush().

Fopen() is very important because it starts the interaction with a file returning a FILE data structure that represents an associated stream. Fclose() on the other side deletes a link between a file and a certain stream releasing all associated system resources and forcing the execution of all pending requests. Fwrite() and fread() execute basic operations of write and read from/to a pointed buffer to/from a certain file pointed by its relative stream. There are then functions which allow the user to switch the file position pointed by the stream but fseek() typically is the best choice and frewind() set the pointer to the first position. Fscanf() and fprintf() are useful to work with formatted data. Setbuf() associate a FILE stream to a memory buffer (if does not exist it will create a new one). Fflush() is very important because it forces pending output data of streams to be written into files.

Before to proceed in the discussion and explain the characteristics of low level functions it is important to have an overview of those operations which manage and give access to files. POSIX defines all the procedures that make the abstractions of files and directories portable between systems. These functionalities perform all the operating systems routines that deal with data and metadata creation, deletion and modification.

The POSIX file system is based on the UNIX operating system and defines common interfaces to files within the motto “less is better”. UNIX principles are important but it must not ever be forgotten that different systems are not constrained to them and it may happen sometimes that things are done in different ways (with different set of functions for example). These cornerstones are: any I/O is done using files, a file is a sequence of bytes and a directory is a list of files.

The POSIX file systems characteristics are here presented as an overview of the most important ones.

- Portable file names: no more than 14 characters composed by only letters, numbers, under score, hyphen and point with lower and upper case making some file names possibly similar but different.
- Directory tree: any file system starts with a directory called root with file name “/”. It is a list of files, some of which may be directories. It is the same for general directories apart for the name. A file can be named calling the chain of directories starting from “/” with their names separated by a “/” (this is the so called absolute path). The chain of directories before a file name is the path prefix. A file can also be referenced with a relative name starting from a specific directory known as the working directory instead of the root one. The functions to work with directories are getcwd() which returns the current working directory while chdir() allows to switch it to another one.
- Making and removing directories: it is possible to create a new directory calling the mkdir() function while rmdir() is capable of deleting an existing one.
- Directory structure: each file in the file system has a unique number (the so called i-number which references the file i-node data structure with all its associated metadata). Every element inside a directory points a serial number and many different path can reference the same files. (NB: memory addresses of data records of files are contained into the i-nodes structures).
- Linking to a file: the function link() associates a certain path with the file specified.
- Removing a file: the unlink() in contrast does the opposite of link() and when a file has no more associated links is simply deleted.
- Renaming a file: the outcome of a call to the rename() function is the creation of a new link for a file and the deletion of the old one. While renaming directories and files is portable and safe within the same system, this operation may corrupt files if performed between different systems. In these cases a copy() operation and a following unlink() or rmdir() would be required.

- File characteristics: as said many times a file system maintain information of any files. These metadata can be accessed through the call of the stat() function. POSIX does not specify the exact implementation of this kind of information but define the general structure to allow flexibility.
- Changing file accessibility and owner: with chmod() it is possible to change file's permissions while with chown() it can be changed its owner in order to implement different management policies.
- Reading directories: to allow different implementations of directories, POSIX defines only the functions that allow users to retrieve directories entries. Similarly to high level functions for files and streams the opendir() returns a directory stream which can be used to access directory information. Readdir() returns a structure containing information related to a specified directory and closedir() notify the system about the no interest on working anymore with a certain directory. Rewinddir() reset the position of the directory stream to its beginning.

Now that all the file systems structures and its functions are clear it is possible to proceed with the description of low level functionalities that, in contrast to what it could be thought, are not very well specified between systems in terms of general behavior. In previous C language implementations, they were the routines called when high level functions executed. Now the POSIX standard (more conformed with the standard C libraries definition) defines interfaces also at a low level but because of this it is good practice to call high level functions if seeking very high portability and low level functions if more control over files is needed.

Low level primitives work with file descriptors which are integers identifying opened files to access their data structures which are loaded into main memory during open() calls and saved back during close() ones.

Read() if called, copies data from an opened file and save it into a buffer similarly as for write(). An example of possible incompatibility is the case when an integer saved into a system where integers have a 32 bits size is then read on a system where integers

are of 16 bits. In this case with only one call to the read() function, the entire file will not be read.

Fcntl() is an interesting multi-purpose function that performs operations over file descriptors of opened files. This is another interesting example of possible incompatibility as POSIX does not define what it will happen if someone attempts to modify the flags that this function work with.

The lseek() function has a similar behavior of the flseek() mentioned before. The substantial difference is that instead of working with a file stream it works with the file descriptor.

Finally, it is worth of noticing that high and low level functions can be mixed. For example calling fdopen() a file stream pointing to the file associated with the specified file descriptor will be returned while fileno() does the opposite. It is important to keep in mind that working simultaneously on the same file with both file descriptors and file streams may cause incongruence as the behavior of the formers may vary from system to system.

All the things said in this chapter will be very important when it will be presented the implementation of a plug in to make an existing cache, built with POSIX standards, capable of storing information on a certain storage as a back-end. It will be shown the power of working with standards in the area of Information Systems.

Now that the concepts of file systems are clear it is possible to go further with the discussion introducing Distributed Information Systems as the final goal is the implementation of a cache working in Cloud Computing environments.

## 2.4 STORAGE NETWORKING TECHNOLOGIES

First of all, it is important to have a well understanding of why companies are so interested on storing data over Distributed Systems. This will justify the reasons for adopting Cloud storage solutions.

There is a common trend showing an increasingly growth in the amount of data produced by companies that however still need to be stored, fast accessible, protected and managed efficiently. In order to obtain these properties, it becomes necessary to switch from a centralized to a distributed storage system to overcome the limits of single machines.

To be more precise, a valid information solution should be able to [8]:

- provide data availability to users when they need it
- integrate the information infrastructure with business processes
- provide a flexible and resilient storage infrastructure

The first interesting system that will be mentioned is the DAS (Direct-Attached-Storage) which is nothing else than a storage environment where memorization devices are all directly connected. It represents a system where storage devices are isolated on their own and because of this it becomes hard to share information between users.

The natural evolution of DAS is the so called SAN (Storage-Area-Network) that is a dedicated network of storage resources. It is not important to deepen in these concepts but it is interesting to see how the paradigm is switching from a centralized to an always more distributed scenario. The SAN is capable of overcoming the limits of its predecessor making sharing files in distributed environments possible and more efficient, with better economies of scale and management in terms of data protection and maintenance. Moreover, thanks to virtualization techniques, in SAN environments it becomes possible to enhance utilization and collaboration among distributed resources over different sites. For this reason, the utilization rate of storages is improved compared to the direct-attached-storage because the information is now sharable. This is very important as it has considerable impacts on companies needs of infrastructures.

DAS and SAN represent good solutions to interconnect the different resources of storage infrastructures to answer the growth of companies' data but have shown some limits. In network-based file sharing systems, file servers use client-server technologies to provide required data and this resulted in the appearance of over and lower utilized storage resources.

A step forward has been made with the introduction of NAS, the so called Network Attached Storage, a dedicated storage device that can provide high-performance for file-sharing eliminating the need of many general purpose machines. It enables client to share files over an IP network via the introduction of network and file-sharing protocols such as TCP/IP for data transfer and NFS (Network File System) for network file services. The NAS utilizes a specific operating system that is optimized for I/O operations to serve specific file services' needs, making it better performing than a general purpose machine. This results in the number of clients that can be served simultaneously.

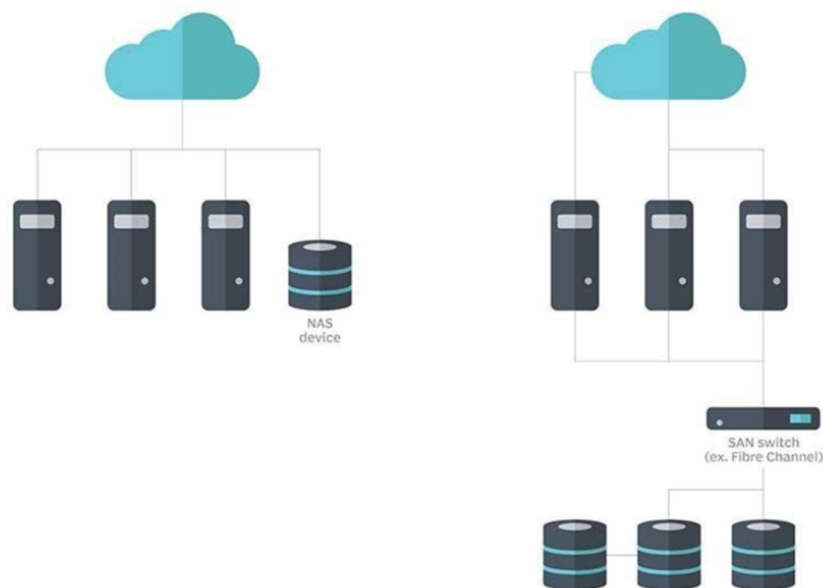



*Image 2.3 – Examples of NAS and SAN in the internet scenario*

The benefits introduced by the NAS are:

- It enables efficient file sharing

- It supports many-to-one and one-to-may configurations
- It provides better performance and ease of deployment
- It is flexible as compatible with UNIX and Windows systems
- It centralizes data storage and simplifies file systems management
- It provides better scalability and high availability
- It ensures security and reduces costs

Network-based file-sharing environments become therefore composed by many general purpose servers and NASs that may need to transfer files between them. Because of this, a file-level virtualization that enables files mobility across different and heterogeneous devices has been implemented.

As shown there are many reasons to switch to Distributed Information Systems to share data. In the next paragraph it will be now presented how these environments effectively work with also a deeper explanation of some of the just mentioned key concepts.

## 2.5 NETWORK FILE SHARING AND MAIN FACTORS AFFECTING PERFORMANCE USING IP NETWORKS

In a file-sharing environment users are capable of storing and retrieving information across multiple nodes within distributed deployments adopting protocols made for these kind of tasks.

Some interesting examples of the methods utilized for file sharing are:

- FTP (File Transfer Protocol): It is a client-server protocol that allows the transferring of data over the network using TCP. (NB): SFTP (SSH FTP) is its secure version.
- Distributed File System: It is a file system in which data is distributed over many computational nodes. It ensures efficient management and security while users can access data with a unified view of all its files.
- NFS file-sharing protocol: It enables files owners to define their specific type of access. With its utilization users can mount a remote file system therefore making its files locally available. It provides routines for: searching, opening, reading from, writing to and closing files, changing file attributes and modifying directories and file links.
- DNS (Domain Name System): It is a service that helps users to identify and access resources of a certain network.
- LDAP (Lightweight Directory Access Protocol): It is an example of service protocol that creates a namespace and helps to identify resources of a network.
- Peer-to-Peer model: It represents a paradigm where machines can share files within a network. The discovery of files is done by a software appositely built. Even if it will not be further explored, it is important to mention this last model as it represents a valid alternative to the more classic client-server one.

As discussed in the previous section the NAS represents an important element in Distributed Information Systems. Its main types of implementation are presented below as

they involve concepts that can be generalized as models which are very useful to proceed in this study:

- Unified: It provides data access within a unified storage platform with a central management system.
- Gateway: Contrary to the unified model it utilizes external storage for data memorization and retrieval with a consequent need for separated managements.
- Scale out: This implementation is very important for Cloud environments as it represents the structure of data centers' clusters organization.

Models are important because they help us to better understand Distributed Systems making the analysis of those elements that have important impacts over performance. In chapter 5 the cache developed during this project work will be presented. The cache works within an IP network environment since it was meant to work with S3 objects.

The main factors that impact distributed storages in this kind of networks are presented below [8]. It is important to consider these factors even if some of them may appear obvious:

- Number of hops: a large number of hops may considerably increase the data latency as each one of them requires IP processing.
- Authentication service: there must be enough available resources dedicated to this tasks in order to avoid congestions and latency increase.
- Retransmission: it is important to set this parameter appropriately as one of the most affecting network traffic jams. It also may bring up unexpected errors.
- Overutilization of routers and switchers: additional devices should be added if some of them become over utilized.
- File system lookup and metadata requests: The processing required to access files or directories at the appropriate locations is typically the main bottleneck in Distributed Information Systems. An intricate directories structure may cause important delays. Because of this it is typically flattened to favor fast and efficient

data retrieval. This concept is very important and must be mastered to understand the scenario of this Thesis work.

- Over utilized devices: The execution of multiple and simultaneous data access operations may cause the overutilization of some system's devices. This would negatively affect performance as a result of a bad data distribution.
- Overutilization of clients: clients can use protocols for network file sharing such as the NFS. If a client becomes over utilized the processing of all its relative requests and responses may cause delays on data retrievals.

Distributed storages at this point should be more clear in terms of benefits and performance impacting concepts. Even if with some performance limitations during data transfer and data retrieval procedures, storage paradigms defined by SAN and NAS have shown increasing improvements over data storage techniques with respect to blocks and files abstractions. They still represent the base for information systems and this is the reason why it is important to keep their model in mind.

However, there is another kind of storage that is the object based storage. It is a prominent storage paradigm with no boundaries in terms of performance enhancing. It has also the incredible capacity of making stored information simultaneously accessible within the abstractions of blocks, files and objects. For this reason, it can be considered an incredibly portable type of storage.

The object storage technology is presented in the next section. It has a central role in this work as the storage back-end of the implemented cache system follows this model.

## 2.6 THE OBJECT STORAGE

To better describe the object storage model and why it is so powerful it is good to step back in order to understand what are the limits of hierarchical file systems and how they can be overcome.

The main ways users have been interacting with data over the years can be summarized in the two categories of databases and file systems [9]. The former is capable of efficient management of huge amounts of data thanks to the structures that can be defined from the characteristics of the information kept in memory. An analysis of the properties of databases will not presented here but it is worth mentioning how their paradigm has influenced new storage technologies. They are good solutions to process huge amount of data but they also become inadequate when more control over data is needed. File systems help in this direction giving more management power to users.

Traditional file systems however present some limitations. Users' needs are changing and new paradigms must be investigated in order to achieve better efficiency in storages management. Moreover, people are now working more and more with bigger amounts of data. This, with the change in data retrieval paradigm that is now more focused on file characteristics rather than data organization, has made the hierarchical structure of directories and files useless.

The problem of the hierarchical namespace, as mentioned in the previous section, may impact systems performance introducing overhead as it makes file localization more expensive in terms of computational costs. In addition to that, users are now accessing data that is typically unstructured which means that it does not only make NAS more inefficient but it also becomes useless.

To overcome these limitations object-based storages represent the perfect choice as they are capable of managing files according to their content and characteristics rather than their location and organization. Because of this, it is important to understand what are their main properties and benefits.

The main element in an object storage system is the OSD (Object-Based Storage Device) which is a device meant to organize and store unstructured data in the form of objects. OSDs do not keep a directories hierarchy but maintain the address space flat. Objects are identified through a unique ID number that is generated by appropriate functions (hash functions for example) or specific algorithms. Objects are capable of storing information such as user data or metadata along with them enhancing the compatibility of heterogeneous storage systems.

An OSD system is typically composed by many servers interconnected within an internal network. They run the OSD service environment that provides functionalities to access and manage the stored data. The main services are the metadata service and the storage service. The former is in charge of providing ID keys to objects while the latter works in contact with the disks where users' information is maintained.

An important characteristic of OSDs is that they perform very well with many low-cost disks that are less expensive than a single powerful one. Finally, the principal benefits introduced by object-based storages may be summarized in:

- Security and Reliability: Data integrity and authenticity are guaranteed by the storage that has the responsibility of performing user authentication and encryption procedures.
- Platform independence: As objects are no more than containers of data, metadata and attributes it becomes possible to store them over different distributed, heterogeneous and remote devices. This property is very important especially for Cloud Computing environments which are the subject of this work.
- Scalability: Thanks to the flat address it becomes possible to store huge amounts of data without impacting performance.
- Manageability: OSDs are capable of coordinating their operation autonomously according to user defined policies. This makes them capable of self-management that is very important in complex systems environments.

## 2.7 A UNIFIED VIEW: BLOCKS, FILES AND OBJECTS

To complete this analysis of storage technologies it is vital to discuss another relevant paradigm of storage systems that is the unified storage system. It can provide a unified view of the data in all the forms of blocks, files and objects at the same time. This is very important in Cloud Systems where stored data and devices are typically heterogeneous. At the end of this section, to conclude this chapter, an interesting comparison of the different types of storage technologies will be presented.

A unified storage system consists of four main components:

- The storage controller: Provides block-level access to the application servers and manages the back-end storage pool of the storage system.
- The NAS head: Provides access to NAS clients acting like a file server. It interacts with the storage thanks to the virtualization of physical devices offered by the storage controller. It is in charge of the configuration of the file systems installed on the disks, to undertake the NFS and to share the data with the clients.
- The OSD: Interacts with the storage via the storage controller and provides web communication to application servers with REST, SOAP and dedicated API interfaces.
- The storage: Is the physical storage. It is composed of many different interconnected devices which maintain users' information.

In the following picture it is shown a scheme of this specific storage system architecture.

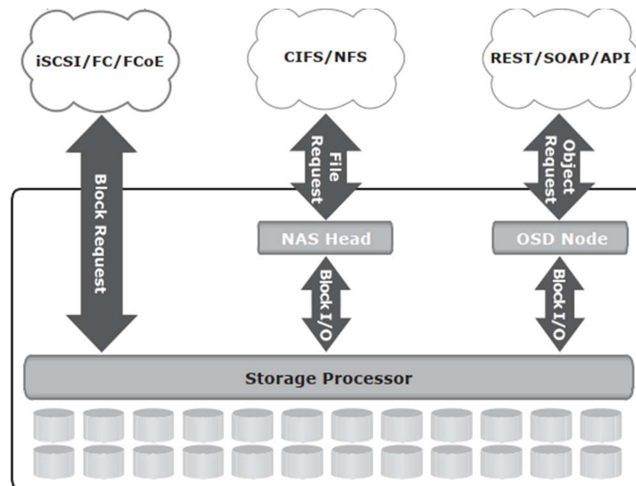



*Image 2.4 – Unified information storage architecture*

Unified storage systems are a prominent solution nowadays because of their incredible abstraction power. There are therefore many reasons that justify their adoption and one cannot deny that this paradigm appears to be the most suitable for many industrial scenarios.

However, many systems still interact only with the protocols and API defined by blocks, files and objects data paradigms. Because of this it is important to keep in mind what are their main differences and most suitable scenarios.

The following table presents a summary of what has been discussed and concludes this chapter.

| | |
|---|---|
| Block Storages | Meant to work closer to the hardware, blocks are able to chop amount of data in different part that can be stored on different machines independently from the specific operating system. They provide more control over the data becoming more efficient eliminating the infrastructure around files but becoming therefore less user friendly. |
| File Storages | Meant to allow users to work with their data they are very suitable for high level applications. However, they implement all the infrastructure characterizing files that is a penalty for systems performance and also a possible problem for compatibility. |

| Object Storages | Meant to overcome all the limits of file storages they are very well performing in those scenarios where data is unstructured as they maintain a flat namespace. They are very good to achieve good performance at a high abstraction level and compatibility between heterogeneous systems. However, objects cannot be rewritten and in system where many write operations are required they are not well suited such as it happens with databases. |
|---|---|

*Table 2.1 – Storage paradigms' concepts summary*

The following picture shows a simple model of these distinct paradigms.

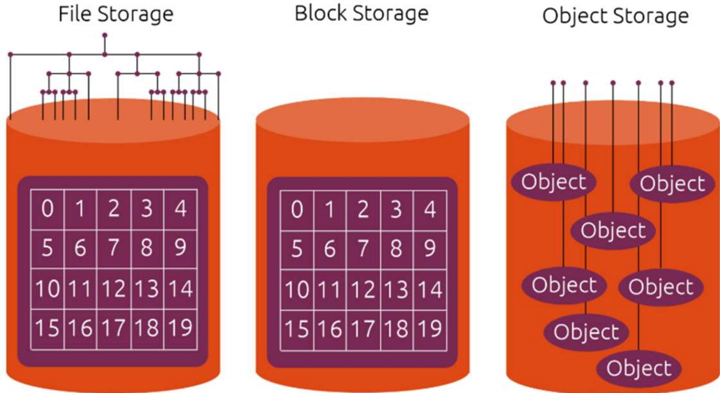



*Image 2.5 – The storage paradigms of file, block and object memories*

In this chapter a deep overview of storage systems has been presented. Now that these fundamental concepts are clear it will be presented in the next section the problem of cache memories and a summary of what is the state of the art of some interesting storage technologies that have been particularly important for this project work.

# 3 STORAGE TECHNOLOGIES AT THE STATE OF THE ART

At this point it is very important to have clear in mind the final goal of this project work that is the development of a cache layer capable of working in Multi Cloud environments to serve computational nodes communicating with S3 object-storages to retrieve and store data.

Within this chapter the analysis of storage technologies will become less abstract as some important real cases will be briefly presented in order to give a good idea of what is the current scenario of this scientific area. Initially the concepts of caching paradigms will be given in order to introduce the problems that involves. Then some technologies at the state of the art will be summarized to extract the key concepts behind them, introducing in this way the first work activity that has been done, that is the development of a plug-in to extend an already existing cache program. Finally, the S3 Cloud Object store will be presented as the most adopted storage service of Cloud solution.

## 3.1 THE CACHE MEMORY IN INTELLIGENT STORAGE SYSTEMS

Storage systems are more complex than what we can think. They are meant to provide data access, management and retrieval to users by using simple instructions and APIs. Unfortunately, the internal architecture of information systems is not as simple as the user interaction process.

If good management and fast retrieval are characteristic to seek in order to provide a good service, some assumptions must be done. Typically, many requests simultaneously reach the computational nodes where the information is kept, as a result monolithic structures would represent a bottleneck during responses processing. For these reasons the structure of modern and well performing systems is broken down into four different main elements: the front-end, the cache memory, the back-end and the physical disks [8].

The front-end is defined as that component of an intelligent storage system that specifies the interfaces through which the users can communicate. It also interacts on the storage side with the cache memory to perform I/O operations.

The cache layer is very important as it is the component that abstracts the storage making it possible to achieve higher performance. It is a volatile memory that stores temporary data. It is in charge of providing the information requested by the front-end and of storing into the physical memory what the front-end receives from users.

The back-end instead provides an interface between the cache and the physical memory. It performs I/O operations directly communicating with the disks to make data persistent and/or available to the cache. It also represents an additional temporary memory as well as the cache acts for the front-end.

Now that the general guidelines of a performing storage architecture have been defined it will be better explained what problems and challenges are introduced to developers by the cache memory in order to achieve the expected benefits from its adoption.

Physical disks are adopted because of their high storage capacity and low costs but they bring with themselves also an important drawback in terms of I/O speed. Cache memories are more expensive and volatile but also capable of providing high rate responses. Caches are meant to overcome the time limits presented by disks in order to seek the best solution in terms of both performance and costs. However, because of their capacity limitation it is important to plan an appropriate memory management in order to enhance read and write operations speed without introducing a bottleneck to the system.

Cache memories communicate in both users and physical memory directions while interacting with the storage's front-end and back-end.

During read operations, before communicating with the slower disks, it is checked if the cache memory already contains the necessary data. If this happens the cache memory responses back to the front-end sending the requested information. This case commonly known as cache HIT represents the best scenario which makes cache memories particularly

advantageous as they bypass the problem of slow communication between front-end and physical disks. The opposite situation is the so called cache MISS. In this case the cache memory needs first to retrieve the requested data from the disks before being capable of answering users' needs.

The performance of reading operations is as much better as much higher it is the number of HIT compared to MISS that is known as hit ratio. Because of this reason being able to predict the data that will be requested may have huge impacts on performance. An interesting policy in this sense is the so called "data pre-fetch" that tries to read information from the disks before it is actually requested by users. This is particularly helpful to serve sequential reads operations.

Writing operations instead take advantage by the presence of cache memories in two distinct scenarios:

- Write-back cache: Data is written on the cache memory and a commit is immediately sent back to the user that can therefore continue with its operations without having to wait data to be physically written to disks. This is very useful to make write operations faster but it may happen that system failures will cause data loss as cache memories store volatile data.
- Write-through cache: When data is stored on the cache memory it is immediately written to the physical support. The commit will be sent back to the user only at the end of the write operation. This eliminates the possibility of data loss but has the drawback of bounding write operations speed to the physical support's I/O capacity.

As previously mentioned an important concern of cache memories is their limited storage capacity. They are implemented as fixed size blocks (known as pages) vectors of data. The bottleneck of information systems is represented by I/O operations which make it important to reduce their number as much as possible. Data is so maintained in cache to enhance efficiency but it cannot be kept for long times because otherwise it would introduce congestions problems that globally affect the performance of requests processing.

Apart from reading in advance or keeping modified information to reduce the number of I/O operations it becomes thus important to plan an appropriate management to reduce the number of the data maintained by the cache layer.

Some important policies are:

- Last Recently Used: Data that has not been accessed by the longest time is removed from the cache memory and eventually written to the disk if not updated.
- Most Recently Used: Based on the assumption that data when accessed has a low probability to be accessed in short times, this policy frees cache's memory from the most recently accessed data.

The operation of writing not aligned data to the disks is called flush. With this operation information that has been modified inside the cache memory will be written to the disks making so users' updates persistent. Without entering too much in the details it is interesting to see the three main flush operations:

- Flush idle: Flushing operations are performed during average cache memory usage. This is the most desirable situation as it does not have impacts on global request processing speed.
- High watermark flush: Flushing operations enter in execution when cache usage reaches a certain threshold. In this situation some parallel I/O operations may be affected on performance because of flush operations cost.
- Forced flushing: Flushing are performed when the cache memory cannot store anything else because it has reached its maximum usage. This scenario must be avoided because it is the principal cause of bad I/O rates.

Cache memories thanks to all these measures represent the best choice to achieve good performance while adopting inexpensive devices. Moreover, some industrial solutions have demonstrated that the adoption of multi-layer cache systems may enhance performance even more. Therefore, it is important to keep in mind that there are scenarios where these principles can be extended.

Finally, in cache architectures the back-end device interacting with the storage memory has a central role as its performance will determine how a certain cache solution will perform. For this reason, here in the next section it will be presented a summary of some prominent technologies that can be adopted as a back-end in the Multi Cloud environments caching systems.

## 3.2 FILE SYSTEMS AT THE STATE OF THE ART

File systems are the basic mechanism involved in data access and retrieval procedures. An exhaustive explanation of their key concepts has been presented in sections 2.2 and 2.3. Now it is important to understand how they can be introduced in specific scenarios and how they are evolving to serve new needs introduced by new computational paradigms such as by Cloud Computing environments.

Many examples of file systems meant to serve different cases are available in literature. Some of them have been implemented to test new possible ways of interactions with specific storage solutions while some others have just been a natural evolution in data access mechanisms to go beyond the limits presented by traditional computational paradigms ([12] and [13]).

For instance, an interesting experiment has been done to provide POSIX interaction to databases in the development of a file system with a database as back-end storage [12]. Databases are very powerful storage systems for structured data but are also characterized by specific interaction paradigms. This is a good example that shows how important the POSIX semantic is in allowing program portability over heterogeneous systems. This project is worth mentioning because it shows also how for performance matters sometimes, POSIX standards must be abandoned even if is very powerful. This is the reason why this project was not very successful.

Another interesting example is represented by the development of a file system capable of interacting directly with GPUs and thus allowing programs to bypass CPU support that was causing important delays in computation and program development [13]. New computational paradigms are becoming more and more prominent. An important case is the parallel computation offered by GPUs. Without going into too much detail it is interesting to highlight that new paradigms such as GPU computation may be negatively affected by the already existing model making therefore programs development more intricate and complex in order to overcome these differences. This experiment has been

successful and opened new opportunities to enhance the performance of the systems taking advantage of GPU's hardware properties.

The computational model of Cloud Computing is characterized by Distributed Systems' architectures which are made of many inexpensive and heterogeneous devices. Because of this it is important to understand what are the best solutions at the state of the art that can be a valid choice as a storage back-end.

Traditionally many file systems solutions have already been proposed for this type of scenarios. Traditional Distributed File Systems [18] such as NFS or NAS had the common goal of giving a unified view of a distributed storage environment composed by many nodes deployed across multiple sites. They represent a good alternative for this purpose, however, because of their characteristics they do not fit very well in Cloud scenarios as they do not meet the requirements of performance, reliability and level of automation demanded and also they introduce geographical limitations. Moreover, while giving a unified view of a set of storage nodes they had many concerns about devices failure which could represent a single point of failure for the entire system.

In order to overcome all these problems a new kind of storage technology for distributed scenarios has been developed which is the so called Cloud Distributed File System that is instead very suitable for Cloud environments. These new solutions are a new generation of Distributed File Systems capable of making users in condition to share data in a simple way such as it would be in centralized systems. They are capable of managing the storage of several nodes with replication in order to achieve reliability qualities that could not be provided by Traditional Distributed File System. This along with better performance and an improved level of automation makes this specific type of technology the most suitable for this project work.

In the following picture it is possible to appreciate their typical architecture in Cloud Computing environments.

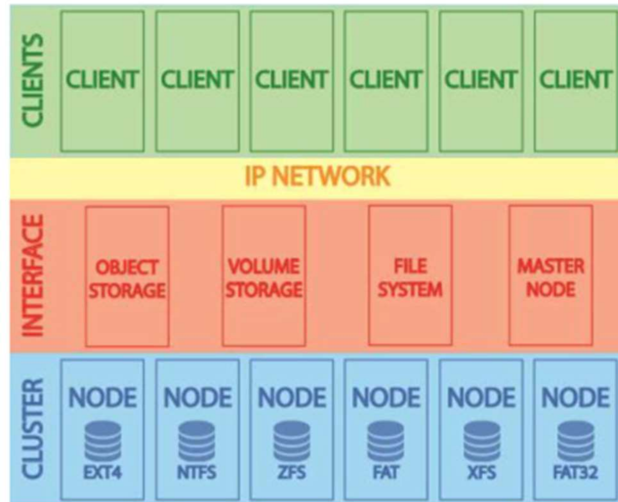



*Image 3.1 – An architectural view of Cloud Distributed File Systems*

Some successful and famous Cloud Distributed File Systems are here presented with their main different characteristics. It is of fundamental importance bearing in mind that even if following a common scheme each of these specific solutions are characterized by different designs, which affect systems' overall performance and properties.

- Hadoop File System (HFS) [10]: It provides a good choice to manage data distributed over different nodes storing files' data and metadata separately. Its architecture is composed of one Node Master (Name Node) which interacts with many Data Nodes. The communication between the different servers is made with the TCP protocol and data replication is executed across different Data Nodes in order to increase data locality probability. The HDFS namespace is kept as a hierarchy of files and directories which is maintained by the Node Master along with the file system's files metadata. When a client reads or writes with the HDFS it first interrogates the Node Master to retrieve the information needed to appropriately retrieve and store data over the Data Nodes. Each cluster is composed of only one Name Node which must be able to hold the communication with many clients simultaneously. The HDFS is also capable of providing advanced services like journaling and backup with the deployment of Checkpoint Nodes and Backup

Nodes. Finally, even if built as an extension of the UNIX file system paradigm its API interface has sacrificed POSIX compatibility to seek better performance.

- Google File System (GFS) [11]: It is a Distributed File System meant to serve large, distributed data-intensive applications. Its main goal is the provisioning of good performance for many simultaneous sequential read operations and atomic append write operations within an environment composed by many inexpensive devices. The design has therefore been characterized by the presence of failures, not as an occasional event, but as the normality. Because of this, constant monitoring, error detection and fault tolerance must be central elements of the system. Also, it has been relaxed consistency of write operations to favor performance. The GFS keeps a hierarchical structure of files and directories and its interface provides all normal file operations such as create, delete, open, close, read and write while however not conforming to the POSIX standard. The architectural design is similar to the one of HDFS and it is characterized by the presence of a single Master holding metadata information and multiple Chunk-Servers where data is effectively stored. Files are divided in many different pieces which are then stored across different servers as Linux files.

- Lustre [15]: It is a good example of Distributed File System that tries to provide the highest portability as possible. In fact, it leverages the power and flexibility of the open source Linux operating system in order to provide a modern POSIX compliant file system capable of satisfying modern data center's clusters' needs. Moreover, Lustre's configuration relies on the XML, LDAP and SNMPS protocols making its management and monitoring able to be easily integrated with other third party components. It eliminates single point of failure problems and distributes both metadata and data across many different nodes. The so called Metadata Servers (MDS) are in charge of managing the file system's metadata while the actual I/O operations are performed on the Object Storage Targets (OST) which are also in charge of interfacing with physical storage devices. The OST are particularly

interesting in this kind of architecture because they abstract the interaction with the real storages hardware, the underlying Object-Based Disks (OBD) which can be improved and changed and still be compatible with the system. Clients can interact with MDSs and OSTs at any time as high availability is granted by LDAP servers that are always aware of the system status and deployment.

- Ceph [14]: It follows the same principle of separating data and metadata but also try to go beyond. It maximizes the separation thanks to the introduction of the CRUSH function which allows random distribution of the information over the nodes of the cluster. In this way objects can be assigned and retrieved on specific and dynamic locations within the different heterogeneous machines of the system. It also supports data replication to enhance overall reliability and failure tolerance. Moreover, metadata management becomes more dynamic as metadata can adapt to the particular usage statistics and Meta-Data Servers (MDSs) deployment. This is possible thanks to the adoption of the Dynamic Subtree Partitioning policy that aims to distribute metadata closer to those nodes where users request data or where the storage devices are underused in order to favor data locality and load balance. Finally, along with these two main characteristics Ceph improves system performance utilizing a specific file system specifically designed to store and manage objects over the Object Storage Devices (OSDs). Thus, instead of relying on different machines' mounted file systems it adopts the so called Extended and B-tree based Object File System (EBOFS) that have shown great performance during the experiments.

Because of its properties that have just been explained, Ceph appeared to be a good choice to satisfy all the needs of the Multi Cloud scenario. Moreover, it provides a unified view of the main storage paradigms such as it has been presented in section number 2.7. This, along with the fact that it is an open source project, makes it the best candidate as the storage back-end for the development of a cache layer for S3 objects. This will become clearer with the next chapters.

## 3.3 THE S3 CLOUD OBJECT STORAGE

Now that caches key concepts have been clarified and also storage solutions have been deeply discussed, what it remains to explain before starting with the presentation of the project work is the Remote Cloud Storage for which the cache has been meant that is the S3 (Simple Storage Service) COS (Cloud Object Storage). For this purpose, the AWS documentation will be used as reference but it must be clear since the beginning that all the reasoning that will be done will not be specifically meant for this vendor but with more general implications.

The S3 is a protocol by which it is possible to interact with a Cloud Object Storage that can be provided as a service by many different vendors. Amazon is the company that first implemented this type of protocol that has been then adopted by many prominent Cloud companies thus making it a standard de facto. This is very important because the adoption of a standard solution frees users by their biggest perceived fear that is the long term vendor locking as it has been mentioned in section 1.4.

To give an initial definition, the Amazon Simple Storage Service can be defined as an object storage service that offers industry-leading scalability, data availability, security and performance [19]. Many type of customers can use it to store and protect any amount of data for their specific business tasks. It is capable of providing easy-to-use management features so that clients can tune the service in order to meet their specific needs. Finally, it is designed to provide nearly 100% durability and to store data for millions of applications all around the world.

Some of the main characteristics of an S3 COS [20] are here presented:

- They provide durable infrastructures to enable customers to store their data as they are designed to satisfy important durability needs. Also, they store data across multiple facilities and devices while giving a single and unified view of the entire service.
- They are low costs and enable users to pay only for the usage (on demand charging).

- They are high available and designed to make the 99.99999% of objects available over an entire year. Amazon defines this as a SLA highlighting its service reliability.
- They can be optimized across different regions. It is possible to specify specific geographical areas in order to define in which facilities data should be effectively stored. This allows to reduce latency and costs.
- They implement the communication protocol over SSL (Secure Socket Layer) to meet security needs with also the opportunity of specifying data encryption mechanisms to protect the information during transferring procedures. They also make it possible to implement access policies to define which users are allowed to work with certain categories of data.
- They allow high performance through the usage of multi-part uploads in order to optimize bandwidth usage during upload operations. Also download operations performance is enhanced, thanks to the support provided for the access of high amount of volumes residing on the storage facilities.

There are many possible use cases where the S3 service is very well suitable [21]. Some few interesting examples are:

- Backup and archiving: It can be used to store backups of clients' information or just as a storage support for business tasks.
- Software delivery: It can be used to store programs that can therefore be easily downloaded by third parties. This can be done via tools like the bitTorrent service. It also grants data access control.
- Big data analytics: Big Data information can be stored on the S3 COS and then analyzed. It is possible also to work directly on Cloud systems without the need of downloading huge amounts of data.
- Media hosting: It is very suitable to store unstructured data such as multimedia files as it is implemented as a true object storage.
- Cloud-native application data: It is compatible with many different Cloud solutions as it adopts a standard protocol. This opens the possibility to implement

new applications without the need of on premises infrastructures as it has already been better explained in section 1.2.

In order to understand how this kind of services works it is important to deepen over its key concepts. Firstly, buckets are containers for the objects that will be stored on the COS. They enable the organization of the namespace at high level. They typically refer to one owner account and can be accessed by many different users according to the policies that the owner has specified. They can so be seen as a unit of aggregation of the information that will be stored.

Objects are the fundamental entities that will be physically stored on the service. They are composed of data and metadata where the former is managed by the COS as just a collection of information and the latter is a set of name-value pairs associated to each specific object. Very important is the fact that objects are identified by a unique name within the bucket they are stored in and that each of them has a version ID number.

Keys are the unique identifier associated to each object within a bucket. Because of this each object is uniquely identified by the triple [key, bucket name, version ID]. Typically, a bucket is accessed by specifying the string name "S3://bucket_name" while for an object it must be specified "S3://bucket_name/object_key" with eventually the version ID.

As mentioned before regions are very important. It is possible to specify a certain region in order to indicate in which specific facility certain data must be stored. In this way it is possible to achieve better performance and save on transmission costs.

Therefore, Amazon S3 is a high available and durable web storage that can be accessed through its specific APIs [23] and not a file system. The main ways of interaction with an S3 COS are:

- The AWS (or other Cloud vendor) Management Console: typically has a web interface and provides a graphical management tool for common users.

- The AWS Command Line Interface (CLI): it is a bit more technical and provides a unified tool by which it is possible to manage and interact with many different and heterogeneous remote S3 COSs.
- The AWS Software Development Kit (SDK): it is a set of libraries written in many programming languages that allow programmers and developers to build solutions for their business tasks and needs. This specific tool had a central role in this project work as a good portion of the cache system has been developed with the AWS SDK for C++ [24].

It is important to keep in mind that it is not a file system because web storages are meant for different purposes. More specifically, an S3 Cloud Storage is very suitable for those scenarios where the information is written once and read many times. This is due to the fact that high availability and durability must be provided. In order to guarantee it, data will be written in many copies that will also be distributed across multiple sites. This of course enhance services' qualities but it also makes write operations more expensive computationally speaking.

In addition, the data stored is considered eventually consistent. This means that there is a small probability that errors may occur during operations execution. This is a Cloud scenario and therefore the service is strictly dependent on the HTTP protocol and the infrastructure of the Internet for data transmission. To be more precise:

- New objects upload: strong consistency as the commit will be returned only after the data has been successfully written across multiple facilities.
- Updates:
    - write then read: could report keys that do not exist
    - write then list: might not include keys in list
    - overwrite then list: old data could be returned
- Deletes:
    - delete then read: could still get old data
    - delete then list: deleted key could still be added to the list

It is not mandatory to deepen on the security techniques offered by this service for the aim of this work. However, it is important to understand the basic mechanisms by which it is possible to obtain access to certain data. This will have important implications on the development of the S3 cache. As mentioned before a user, owner of a certain bucket, may specify access policies such as ACLs or IAMs. When the remote COS receives a request it first checks user's credentials that are specified by the Account Access Keys [22]. These important keys are: The Access Key ID which uniquely identifies an account within the system and the Secret Access Key that is instead meant to specify the access permissions. It is also possible to provide temporary security credentials to users in order to limit services usage, it can be useful in certain circumstances. Finally, through data encryption it is possible to protect the information sent within the body of HTTP packets.

Versioning is another important concept as it allows users to roll back to prior versions of the stored objects. Many versions of the same object can be maintained even if the object has been deleted from the bucket. This enhance system persistency and makes client operations safer from the eventually consistent characteristic of S3 storage services.

## 3.4 S3 REST API, SERVICES PRICING AND BEST PRACTICES

The S3 storage service defines a specific set of API. Because of this it is here presented a summary of the main operations that can be performed on a remote COS that adopts this type of protocol. This is fundamental for proceeding in this study because the S3 cache must be capable of interacting with both an S3 client and a remote S3 Clout Object Storage.

Amazon S3 supports the REST (Representational State Transfer) API [23] which is a common and successful interface to allow the interaction with web services [26]. Without entering in its architectural details, it is important to note that the REST communication relies on the HTTP protocol thus making the service that use it "RESTful".

REST API calls to the S3 interface can be authenticated or anonymous. In order to make an authenticated access to the remote COS the credential keys are required (as it has been presented in the previous chapter) in order to assign an authenticating signature to the HTTP request that will be sent. To obtain this signature from the users' keys some computation should be performed. Because of this it is good practice to utilize the AWS CLI or the AWS SDK which will calculate the signature to be included with the packet by specifying the keys values. Key values can be stored into a file at "/.aws/credentials" or just passed to the functions of the SDK in the programming language that is the more appropriate for the specific computational needs.

All the different requests that can be sent with the S3 belong to the same set of API however, to better describe them they are divided into two main sets.

The Simple Storage Service includes all those operations related to buckets and objects:

*AbortMultipartUpload, CompleteMultipartUpload, CopyObject, CreateBucket, CreateMultipartUpload, DeleteBucket, DeleteBucketAnalyticsConfiguration, DeleteBucketCors, DeleteBucketEncryption, DeleteBucketInventoryConfiguration, DeleteBucketLifecycle, DeleteBucketMetricsConfiguration, DeleteBucketPolicy, DeleteBucketReplication, DeleteBucketTagging, DeleteBucketWebsite, DeleteObject, DeleteObjects, DeleteObjectTagging, DeletePublicAccessBlock, GetBucketAccelerateConfiguration, GetBucketAcl, GetBucketAnalyticsConfiguration, GetBucketCors, GetBucketEncryption, GetBucketInventoryConfiguration, GetBucketLifecycle, GetBucketLifecycleConfiguration, GetBucketLocation, GetBucketLogging, GetBucketMetricsConfiguration, GetBucketNotification, GetBucketNotificationConfiguration, GetBucketPolicy, GetBucketPolicyStatus, GetBucketReplication, GetBucketRequestPayment, GetBucketTagging, GetBucketVersioning,*

*GetBucketWebsite, GetObject, GetObjectAcl, GetObjectLegalHold, GetObjectLockConfiguration, GetObjectRetention, GetObjectTagging, GetObjectTorrent, GetPublicAccessBlock, HeadBucket, HeadObject, ListBucketAnalyticsConfigurations, ListBucketInventoryConfigurations, ListBucketMetricsConfigurations, ListBuckets, ListMultipartUploads, ListObjects, ListObjectsV2, ListObjectVersions, ListParts, PutBucketAccelerateConfiguration, PutBucketAcl, PutBucketAnalyticsConfiguration, PutBucketCors, PutBucketEncryption, PutBucketInventoryConfiguration, PutBucketLifecycle, PutBucketLifecycleConfiguration, PutBucketLogging, PutBucketMetricsConfiguration, PutBucketNotification, PutBucketNotificationConfiguration, PutBucketPolicy, PutBucketReplication, PutBucketRequestPayment, PutBucketTagging, PutBucketVersioning, PutBucketWebsite, PutObject, PutObjectAcl, PutObjectLegalHold, PutObjectLockConfiguration, PutObjectRetention, PutObjectTagging, PutPublicAccessBlock, RestoreObject, SelectObjectContent, UploadPart, UploadPartCopy.*

The AWS Control Set instead includes all the actions at the account level:

*CreateAccessPoint, CreateJob, DeleteAccessPoint, DeleteAccessPointPolicy, DeletePublicAccessBlock, DescribeJob, GetAccessPoint, GetAccessPointPolicy, GetAccessPointPolicyStatus, GetPublicAccessBlock, ListAccessPoints, ListJobs, PutAccessPointPolicy, PutPublicAccessBlock, UpdateJobPriority, UpdateJobStatus.*

This work is focused only on a subset of functions defined in the first set. The operations of interest are those concerning reading and writing objects as the final goal is to make data closer to computational nodes than where actually stored in order to reduce the bottleneck presented by WAN connections for intensive computational tasks.

The HTTP packets that will be sent are all different depending on the specific type of operation. They can specify the GET, POST, HEAD, PUT, DELETE or OPTION request methods with or without query parameters. However, it is possible to identify some common request and response headers that can be used by various types of S3 REST commands. Here below it is presented a summary of them. It is very important to understand these parameters because they show how this communication mechanism effectively works.

Common request headers:

- Authorization: Information required for request authentication.
- Content-Length: Length of the message without the header, useful to check data loss during packets transmission.
- Content-Type: Type of the eventual information contained in the body of the HTTP packets.

- Content-MD5: Information that can be used to verify that the data is the same as the one that has been originally sent.
- Date: Date and time according to the requester.
- Expect: Information that can be used to request an acknowledgement before effectively send the body containing the data.
- Host: Specifies the path of a certain resource inside the storage service.
- x-amz-content-sha256: Hash of the request payload used when signature version 4 is used for authentication.
- x-amz-date: Date and time according to the requester, has priority over the Date head parameter if both specified.
- x-amz-security-token: Can be used when paying operations are performed within an S3 service or to provide a security token when using temporary credentials.

Common response headers:

- Content-Length: Length in bytes of the response body.
- Content-Type: Type of the information contained in the response body.
- Connection: Specify if the connection to the server is actually open or closed.
- Date: Date and time according to the responder.
- ETag: Hash of an object used to reflect changes that have been made on it.
- Server: Name of the server that created the response.
- x-amz-delete-marker: Boolean used to specify if the returned object was true or false.
- x-amz-id-2: Special token specified for troubleshoot problems.
- x-amz-request-id: Value that uniquely identify the request.
- x-amz-version-id: Version of the object.

In addition, it is not the aim of this work to deepen on the HTTP protocol's architecture but it is worth of giving a brief overview of the possible response and error codes.

Successful codes are in the range of 200 while error codes 400 refer to request exceptions and 500 to internal service errors.

When an error message is sent back as an S3 HTTP response it typically includes also the following parameters that can be used to manage the specific error that occurred:

- Code: The error code that identify a specific error condition.
- Error: Container for all error elements.
- Message: Contains a generic description of the error.
- RequestId: ID of the request associated with the error.
- Resource: The bucket or the object involved in the error.

As it has been discussed in the previous section, S3 storages are offered as a IaaS over the Internet and are characterized by a standard communication interface making vendors competing on service's quality and pricing. This is a very interesting aspect that must be considered when implementing a cache layer because the approach that will be taken may well or badly impact on monetary expenses. In the next table it is possible to see some pricing policies offered by two main Cloud companies. It is only presented as an overview but more details are available on vendors' websites.

| **S3 pricing policy (EU London)** | | | |
|---|---|---|---|
| *S3 Amazon Web Service [27]* | | *IBM Cloud Object Storage [28]* | |
| **S3 standard** | | **COS for active data** | |
| First 50 TB / Month | $0.024 per GB | 0-499.9 TB / Month | $0.0235 per GB |
| Next 450 TB / Month | $0.023 per GB | 500+ TB / Month | $0.0214 per GB |
| Over 500 TB / Month | $0.022 per GB | | |
| PUT, COPY, POST, and LIST | $0.0053 (per 1.000) | PUT, COPY, POST, and LIST | $0.005 (per 1.000) |
| GET, SELECT and all others | $0.00042 (per 1.000) | GET and all others | $0.004 (per 10.000) |
| Data retrieval | No charge | Data retrieval | No charge |

| S3 standard infrequent access | | COS for less active data | |
|---|---|---|---|
| All Storage / Month | $0.0131 per GB | 0-499.9 TB / Month | $0.0128 per GB |
| | | 500+ TB / Month | $0.0107 per GB |
| PUT, COPY, POST, and LIST | $0.01 (per 1.000) | PUT, COPY, POST, and LIST | $0.01 (per 1.000) |
| GET, SELECT and all others | $0.001 (per 1.000) | GET and all others | $0.01 (per 10.000) |
| Data retrieval | $0.01 per GB | Data retrieval | $0.01 per GB |

*Table 3.1 – S3 AWS and IBM COS pricing policies*

As we can see pricing is affected by two main factors which are the size of the data stored and transferred and the number of operations executed. Bearing in mind these concepts will make it possible to implement policies that will surely make an application working faster with an S3 service but also capable of reducing the number of operations and data transferred thus reducing clients' expenses.

Finally, Amazon defines some best practices [25] that must be taken into consideration while working with an S3 storage service. These guidelines have very important implications on the management of the requests by the S3 cache layer that will be implemented.

Applications can easily reach thousands of simultaneous requests of uploading and retrieving objects per second as the S3 service is designed to automatically scale in order to reach high request rates. It is therefore important to properly manage how these requests will be sent in order to optimize performance and data bandwidth usage. Because of this it is good practice to scale storage connections horizontally. This can be done by parallelizing requests and spreading them over many different connections. Moreover, it is possible to take advantage of the header byte-range parameter that can be defined within an HTTP request packet. In this way only a certain portion of an object is transferred by the HTTP

response reducing latency and incrementing bandwidth usage if many requests of this type are sent simultaneously to download an object.

For latency-sensitive applications it is also good practice to adopt an incremental retry-policy with timeout and retry parameters. In fact, it may happen that a certain response does not choose the best path to reach the destination node because of the distributed nature of this kind of service. In the most of these cases it is better to restart the operation processing as a bad path may take longer to answer than an operation that started after it. Also, it may happen that a packet is lost over the Internet infrastructure and retransmitting the request is the only effective way to complete the operation in execution.

There are some vendor-specific techniques offered in the market like the Amazon S3 Transfer Acceleration that can improve the service performance but it is not very interesting for this work. It is instead important to learn how to optimize an S3 client as the cache layer will be communicating in both the directions of clients (as a server) and storage back-end (as a client). For this purpose, it is important to adopt the latest available version of AWS SDKs that are regularly update to follow best practices of both S3 and REST API paradigms. For example, SDKs operations automatically retry requests on 500 errors. Moreover, latest packages versions allow developer to perform objects upload and also download with the Transfer Manager entity that parallelizes requests to automate horizontally scaling.

In this section all the practical aspects that are useful for proceeding with the development of the S3 cache have been presented. However before starting the new chapter it will be shown a last technology that has shown a great potential in the implementation of new file systems.

## 3.5 THE POWER OF FUSE – AN INTERESTING EXAMPLE WITH S3

To conclude this study over the state of the art of file systems and storage technologies it is now presented a prominent framework that is FUSE. It is worth of noticing because it is interesting to see how it enables developers to create new file systems at the user space level. This has important implications on ease of development and product performance.

File systems in history have been typically implemented as part of the Operating System's kernel [16]. This fact is mainly due to the high performance demanded by programs while interacting with the information stored on a computing machine. However, many developers have started to implement new file systems at the user level as it makes production way simpler. Initially this procedure was utilized only for prototyping because of the many concerns related to the performance of file systems implemented in this way.

The FUSE (File system in User space) framework brings many advantages to the production:

- User space code is easier to develop, port and maintain.
- Kernel bugs can crash the all system while user space bugs only affect the program imitating thus the impacts.
- Many programming languages and libraries are available at this level.
- New high performance interfaces avoid expensive copies of data between user and kernel space.

The last point, along with the fact that the most of the time the bottleneck of file systems is on the program logic and coordination rather than on data movements, has made this framework a good alternative for the production of new file systems. There are indeed many new products that have been developed with these technologies that have also been adopted by many IT vendors.

FUSE is the most prominent framework for developing file systems at the user space and is available for many OSs. Because of this, it has been used as a reference of this new

paradigm for the purpose of this study. The following picture shows the structure of its architectural design.

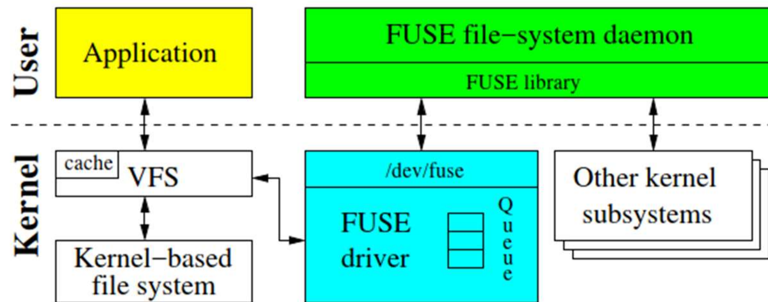



*Image 3.2 – FUSE architecture*

As we can see for user applications nothing changed as they still interact with the kernel's Virtual File System (VFS) that is in charge to redirect the requests to the appropriate low level file system. Therefore, requests are redirected to the FUSE driver that thanks to a queue system manages and sends them to the FUSE implemented file system program at the user space through the FUSE library. It is not important to go into the details of the architecture itself but it must be understood the power and potential offered by such a technology.

The FUSE file system daemon is a program that serves the requests when received. Its behavior is defined by the code written using the FUSE library. With this type of technology, it becomes even possible to define stacks of file systems where at each level one file system interacts with some others.

Some interesting implementation key points are:

- User-kernel protocol: It is in charge of enabling the communication between the FUSE driver and the daemon. The driver when receives requests from the VFS creates a FUSE request structure and sends it to the daemon which will then send back a FUSE response according to the logic implemented by the user space level file system.

- Queues: As previously mentioned the requests sent by the VFS and received by the FUSE driver are managed with different queues that define the order they will be served.
- Splicing and FUSE buffers: Many data transfer optimizations have been made in order to reduce the number of write and read operation between the user and the kernel spaces.
- Multi-threading: Support to parallelisms to enhance computational performance is done by the framework.
- Write back cache and max writes: FUSE operations are mainly synchronous. A write back policy has been implemented in order to improve data transfer performance of large files moved between user and kernel spaces.

The implementation details are not really important for the aim of this work but they are worth of mentioning as an interesting example of storage system technology and techniques that can be adopted to enhance the system's performance.

It is possible to find some related works in literature that have tried to analyze the conduct of this new type of file systems [16]. The experiments have shown interesting results. There are cases where, independently from the adopted hardware, user space file systems behave with close if not better performance than those developed at the kernel level. This is a very good manifesto for the adoption of these frameworks in the development of new file systems as they make production way easier. However, some workloads have demonstrated a performance degradation of the 80%. Because of this, it is important to take into consideration the target for which a new file system is meant before making the decision of adopting a FUSE-like framework in the development of business products. Also, it must always be remembered their impact on CPU usage which is increased of an average of the 30% during the execution of user space file systems in comparison to OS level ones.

However, thanks to this framework many new paradigms have been prototyped but also many new interesting products have been made. An interesting example that shows the power of FUSE is represented by the s3fs file system [17].

In the sections 2.1 and 2.3 it has been deeply discussed about the importance of POSIX standards, especially for file systems. An S3 Cloud storage, even if adopting a standard de-facto protocol does not show a POSIX interface making it not compatible with a normal program-file type of interaction.

S3fs is a file system that has been developed thanks to the implementation power offered by the FUSE framework. It provides a complete POSIX compatible interface to interact with it while it uses an S3 storage as a back-end. This makes it possible to interact with an S3 COS while using normal file systems interfaces.

This project is a new study and still in development. Because of this it is not ready to be adopted as a tool for industrial tasks. However, it is a very interesting study case to take into consideration while working with file systems and S3 Cloud Object Storages.

In the following image a scheme is presented that summarizes all the concepts that have been discussed so far.

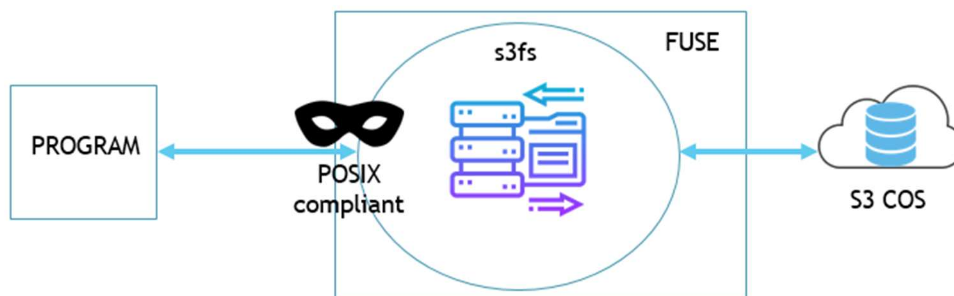



*Image 3.3 – the FUSE s3fs study case*

Starting with the next chapter the analysis will be principally based on the unified storage system Ceph as it has been the central support for the development of this Thesis project work. All the concepts that have been discussed are very important and will help to make clearer further discussions.

# 4 CEPH AND CACHING

In section 3.2 a brief overview of some interesting Distributed File Systems has been shown. In particular, the Ceph storage system has been introduced at a very high level. With the next sections some more details about this technology will be given as it has been central for the design and the implementation of caching solutions for the enhancement of computing workloads' performance in Cloud scenarios.

In order to proceed, Ceph's architecture will be first summarized. Then, the creation of a plug-in that allows an existing FUSE cache program to work with this storage system will be presented. Finally, it will be done a detailed analysis of the part of Ceph concerning the Cloud Object Storage thus introducing chapter 5 where the S3 cache details will be explained.

## 4.1 CEPH – AN OPEN SOURCE AND DISTRIBUTED UNIFIED STORAGE SYSTEM

In section 2.7 the model of unified storage systems which are capable of providing access to stored data in all the forms of blocks, files and objects, has been discussed. In section 2.6 an overview of object storages has also been given while in section 3.5 the concepts related to S3 and more specifically to Cloud Object Storages have been clarified. Now all these concepts will be concretized within the Ceph storage technology.

Ceph is a distributed storage system. This means that it has been meant for data distribution across different nodes and that it has been optimized for the management, sharing and retrieval of information in distributed and heterogeneous scenarios. An important aspect that must be taken into consideration is that it is an open source project which makes it particularly suitable to be extended or exploited for innovation purposes. It can run on commodity hardware like commodity servers, IP networks, and storage devices such as HDDs, SSDs or NVMe [30] making thus a single cluster capable of serving different data paradigms (blocks, files and objects) as a unified storage.

The first important characteristic to be mentioned is that Ceph is reliable, in the sense that it overcomes the single point of failure problem. Data durability is guaranteed through replication and erasure coding. In addition, no interruption of the service is needed, even in situations of system upgrading or cluster deployment changing thanks to the design choice of favoring consistency and correctness over performance.

Ceph is very interesting because it is a technology designed to seek scalability properties. Compared to other solutions like the Hadoop File System or the Google File System it is scalable for data bat also for metadata, as mentioned in section 3.2. It is a complete elastic storage infrastructure that allows clusters to change without the need of interrupting the service in execution. Hardware can be added or removed at any time while the system is online and all the main cluster's management policies can be undertaken, such as:

- Scaling out: Adding more components to a cluster to improve overall capacity and performance.
- Scaling up: Adding bigger and faster hardware components.
- Federating: Deploying multiple clusters across different sites with data replication in order to be capable of disaster recovering.

The following picture summarizes the structure of Ceph's architecture as a unified storage.

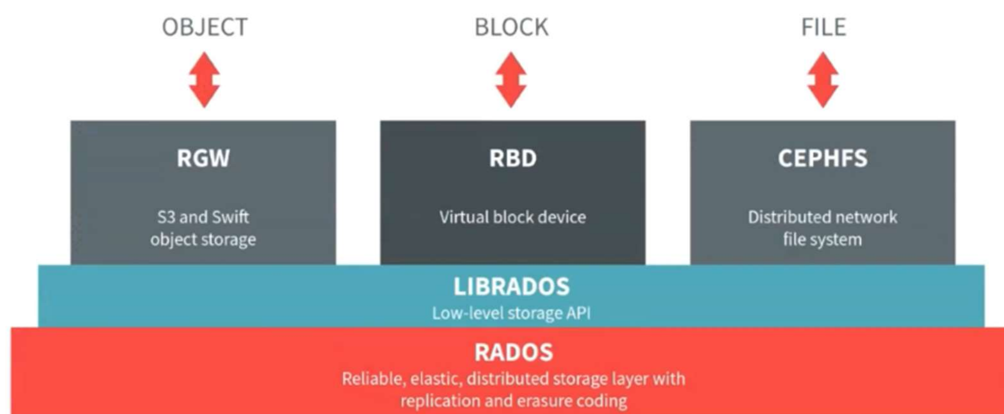



*Image 4.1 – The architecture of Ceph as a unified storage system.*

The RADOS layer (Reliable Autonomic Distributed Object Storage) represents the main component from which all Ceph modules are based. It is the abstraction of the underlying

distributed storage layer that is in charge of ensuring data replication. It handles hardware components and rebalances data after migrations in order to achieve high reliability and availability properties. On top of it there is the librados module that is a set of low-level API by which it is possible to communicate with a Ceph cluster. Finally, another step above there are the three main modules of Ceph. They create the abstractions of blocks, files and objects and provide sets of interfaces by which users can access and manage stored data with the most appropriate syntax according to their specific needs. They serve users requests while executing I/O operations with RADOS thanks to the API offered by the librados module.

These three main modules are:

- RGW (Rados GateWay): Is the component that provides S3 and Swift Cloud object type of interactions within a Ceph system.
- RBD (Rados Block Device): Is the component that enables the data block paradigm.
- CephFS (Ceph File System): Is the component that exposes files and POSIX syntax interfaces.

The RADOS layer therefore is the component that automates data management thus providing strong consistency. According to the CAP theorem it implements a CP system which means that data Availability is sacrificed to guarantee Consistency and Persistency.

Ceph and thus RADOS are a software system that is composed of some different elements. They can be summarized as:

- Monitor: It is the principal module. It is in charge of data replication, data placement and management policies. Within a cluster there are usually between three and seven of them. They are in charge of coordinating all cluster's components.
- Manager: It collects real time metrics that are useful to keep monitored system's statistics. It can also contain some pluggable management functions. Typically, there is only one manager active and one or more in stand-by.

- OSD (Object Storage Daemon): As mentioned in section 3.2, Ceph has improved data retrieval and storage performance thanks to the introduction of the EBOFS (Extent and B-tree Object File System) which is a file system based on b-tree concepts that allows to bypass the Linux VFS and page caching workloads. Along with OSDs that mount EBOFS the data is stored within HDDs or SSDs devices. They serve users requests while effectively managing stored information. As they are the real storage component, it is possible to find a very huge amount of them within a single cluster.
- MDS (MetaData Server): In contrast to previous modules an MDS is not mandatory within a RADOS cluster. It is the daemon that manages a file system namespace within a Ceph system. Because of this, if a program-file interaction is not demanded there is no need for it. However, if the CephFS module is adopted it provides many benefits like for example metadata management but also distribution and balancing thanks to the Dynamic Subtree Partitioning policy that has been introduced by design. It coordinates files access between clients while managing file consistency, locks and leases. Finally, it enables all these services by saving files and directories metadata over Ceph objects (this will be clearer after next paragraphs).

In the following picture it is given a representational view of the components that have just been mentioned.

*Image 4.2 – RADOS software components*

Ceph's architecture and its main software components should be clear at this point but one last important thing still has to be discussed that is how data is effectively stored inside a

RADOS cluster. Ceph externally provides a unified view for blocks, files and (Cloud) objects while internally data is saved as internal objects over the OSDs components. This is where the CRUSH placement function acts and performs all previously mentioned distributed procedures.

In section 2.6 the concepts of object storages have been presented and, as it should be clear at this point, in this kind of technologies what happens is that objects are stored within a container along with some metadata information. Therefore, Ceph objects are stored and retrieved only thanks to their name bypassing heavy lookups procedures by using the placement function. Moreover, they are conceptually contained within pools (Ceph container abstractions) along with "omaps" which are the objects' metadata in the form of sets of key-value couples.

The following picture shows how the placement calculation works.

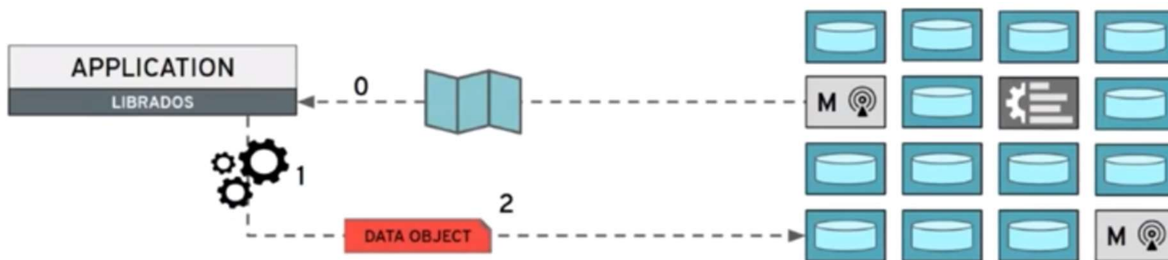



*Image 4.3 – CRASH calculated placement procedure*

An application when interacting with a Ceph's cluster in order to store or retrieve information must follow these steps:

0. Getting cluster's OSDs map while first interacting with a ceph-mon daemon.
1. Calculating correct object location based on its name.
2. Performing I/O operation with the appropriate ceph-osd.

This specific model is the key for the achievement of high availability. In facts, in case a certain OSD that hosts the data of interest got corrupted what it would happen is that the application during map retrieval would get an updated map and while recalculating object

location the address of the OSD containing the copy of the object and not the original one would be returned. This is how replication makes a Ceph system very powerful in terms of reliability.

Without entering in details any further it is just important to mention that pools, which are the container abstraction within a Ceph cluster, can be split into many pieces that are the so called placement groups. Different placement groups are then spread across different OSDs and thanks to erasure coding and replication different data management policies can be implemented. Placement groups are fundamental during placement calculation as they are one of the CRUSH input parameters. This is exactly what makes Ceph dynamic and powerful: it always takes into consideration both cluster's updated configuration and the adopted data replication's policy [29].

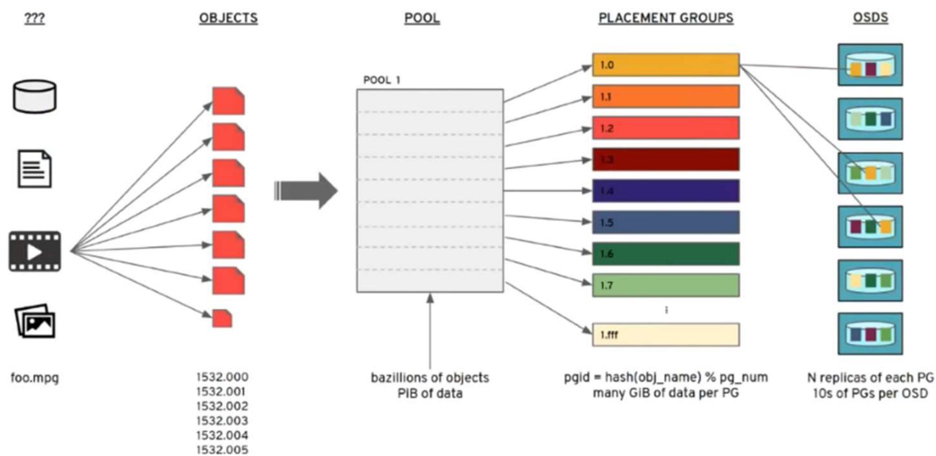



*Image 4.4 – Representation of unstructured multimedia file data distributed over Ceph's objects, pools, placement groups and OSDs*

## 4.2 CEPHFS DEPLOYMENT AND LIBCEPHFS OVERVIEW

The plug-in program that is going to be presented in section 4.3 has been meant for a POSIX compliant cache system. In order to implement this module thus making a Ceph cluster a possible compatible back-end it has been used the CephFS module. Because of this it is here presented a summary of its main concepts as they will make the development explanation easier to the reader.

The next picture shows how clients interacts with RADOS via POSIX semantic thanks to the CephFS module.

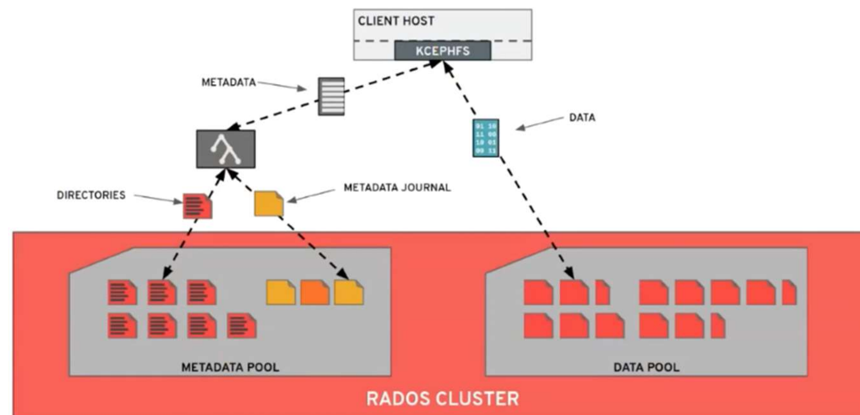



*Image 4.5 – Program-file interaction with CephFS and RADOS*

It should not be surprising at this point that a client must deal with both metadata and data in order to interact with the files stored in a distributed environment. As it is shown in the Image 4.5 a client program first interacts with a ceph-mds daemon to execute the lookup procedure which is meant to retrieve the file location according to the given path. MDSs maintain the file hierarchy structure saved on objects within a Ceph pool that is the so called metadata pool. Moreover, additional file systems' advanced services such as journaling are implemented. In fact, along with the directory hierarchy's objects it is possible to see also journaling objects which contain log records useful for recovering operations in case of files corruption.

Once the lookup operation has been performed, a MDS service return to the client the requested files' metadata and all the information needed to perform data retrieval. Data is

so obtained by a following request to the data pool by which data objects are given back. Finally, the file is composed and returned to client as a response for the file request.

All these procedures are transparent from the client point of view. If clients were forced to adopt specific APIs with the HDFS or the GFS, when interacting with a Ceph file system they can just interact by using standard POSIX file systems commands. In the following picture it is shown how this is made possible by the Ceph's libraries structure.

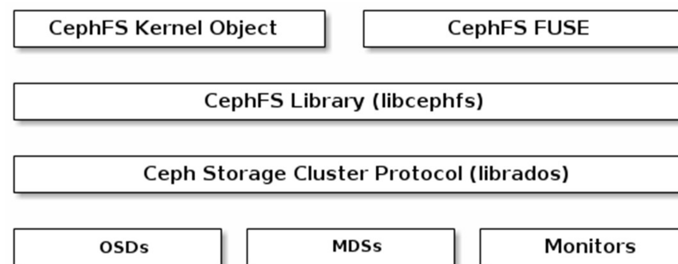



*Image 4.6 – View of the CephFS module within Ceph's software hierarchy*

A client can just mount the Distributed File System provided by a Ceph cluster thanks to the CephFS Kernel Object or the CephFS FUSE modules [29]. The former allows to mount the file system within the kernel driver of the operating system while the latter uses the user space file system framework FUSE that has been explained in the previous chapter. Once mounted, clients only have to interact with the file system within the mount point address that has been provided. These two modules are based on the libcephfs library that implements the POSIX functions by which it is possible to interact with RADOS. Libcephfs is therefore the layer that is in charge of overcoming differences between POSIX-like file systems paradigm and the Ceph's RADOS storage abstraction while using librados to communicate with the cluster's storage devices.

## 4.3 PLUG IN DEVELOPMENT – THE POWER OF FUSE AND POSIX

Cloud environments are capable of providing computing infrastructures as a service over the internet. An important characteristic that must always be remembered regarding these scenarios is that they are based on the hardware and the environment of vendors' data centers. It has been said many times in chapter 1 how virtualization techniques are important for providing this kind of service. Moreover, there is a prominent trend nowadays that shows an ever-increasing abstraction level of software and hardware virtualization techniques. The main example of this is represented by the ever-growing adoption of containerization solutions. This new paradigm has introduced new prominent technologies in the Cloud scenario such as Docker [31], Kubernetes [32] and OpenShift [33]. This has made it more dynamic and easier data centers and on premises infrastructures management but has also introduced new concepts that must be taken into consideration while working with Cloud systems.

For instance, it is particularly important for this work, as related to containers and file systems, to avoid the usage of mount points as much as possible. This is principally due to the fact that in Cloud environments machines are shared between different applications and only a small amount of control over the hardware is given to the single programs in execution. Because of this reason the solution presented will not use the top level modules of CephFS Kernel Object and CephFS FUSE but will work instead directly with the functions offered by libcephfs in order to make the application capable of working even if the file system provided by a Ceph cluster is not effectively mounted within the OS of a data center's machine.

The internal caching logic details of the caching system that has been extended are not really important for the aim of this discussion. What is instead very interesting is how the FUSE framework has been adopted to make the cache POSIX compatible and easy to be extended in terms of heterogeneous back-end systems.

The cache's core module follows the normal cache paradigm where the data is maintained into the cache memory and managed in the form of fixed-size blocks. When a byte range of a certain file is requested by the user, the cache memory reads the blocks that contain the requested data which is stored into the persistent memory of the storage device. It may happen that more bytes than needed are read during I/O operations according to the blocks' size defined when the cache system has been started. Because of this it is important to choose the block size that fits the expected data transfer behavior between users' applications and storage devices. If the size specified is too big it may cause delays and the cache memory will become full faster. However, if the block size is too small, it could cause a huge increase in the I/O operation number which may cause congestions and high delays for big byte range data transfer.

The cache program allows the user to specify three different mount points addresses:

- Mount point of the file system to be cached.
- Mount point of the storage back-end where to store cached data blocks.
- Mount point of where to mount the cache itself.

Thanks to this paradigm for example, by specifying the mount point where it has been mounted an s3fs [17] file system, it would be possible to perform the caching of an S3 Cloud Object Storage. It is possible to cache any kind of file system within this system, the only thing that really matters, is that they must be POSIX compliant.

To be more precise, not all these mount points must be specified in order to interact with this caching program. If an appropriate plug in has been developed for a certain storage technology thus making the cache program compatible with it, the only mount point that has to be specified is the one of the cache. Caches are meant to work on the same locality of application programs so that it is not really a problem to mount them.

In order to use a Ceph cluster, as a cache back-end storage system, the mount point of a Ceph File system (mounted with the kernel module or the FUSE one) could just be passed

to the cache. However, as previously mentioned, it is important to limit the number of mount points because of data centers' machine accessibility permissions.

When working with the FUSE framework the key concept in developing new products is the "fuse_operations" data structure defined by libfuse [37]. Within this structure the behavior of the file system during processes execution must be defined by specifying the functions that have to be called. In practice, along with the parameters of this structure the behavior of the FUSE cache is implemented.

These functions in addition have been structured in such a way to interact with the back-end storage by calling the procedures defined in another data structure that specifies the functions by which it is possible to interact with a specific technology. During the cache startup procedure, along with the previously mentioned parameters, the back-end system that should be used must then also be specified along with its specific configuration parameters. In this way the structure that is loaded is the one defined within the plug-in of the specified back-end solution.

The implementation of a plug-in that makes a Ceph cluster a possible storage back-end for this cache system has simply been a mapping between the functions defined by the "libcephfs" module and the back-end procedures data structure, thanks to the FUSE paradigm and the standard defined by POSIX. Here below the functions mapping is presented:

| FUSE cache back-end procedures structure | libcephfs |
|---|---|
| • .open | • ceph_open |
| • .close | • ceph_close |
| • .mkdir | • ceph_mkdir |
| • .remove | • ceph_remove |
| • .rename | • ceph_rename |
| • .stat | • ceph_stat |
| • .lseek | • ceph_lseek |
| • .read | • ceph_read |
| • .write | • ceph_write |

| | |
|---|---|
| • .chown | • ceph_chown |
| • .access | • ceph_access |
| • .opendir | • ceph_opendir |
| • .closedir | • ceph_closedir |
| • .readdir | • ceph_readdir |
| • .setxattr | • ceph_setxattr |
| • .getxattr | • ceph_getxattr |
| • .statvfs | • ceph_statvfs |

*Table 4.1 – FUSE cache back-end – libcephfs POSIX mapping*

The reason why this work has been presented is to show how taking advantage of these technology paradigms, along with a smart design, makes the work of developers very simple. This is very important in industrial cases as it may have important consequences in terms of time and costs. In addition, as portability is an important characteristic to seek in Distributed Systems scenarios, these concepts may also determine the diffusion rate of a product.

This case shows a practical case where FUSE and POSIX made the file system implementation very easy. Thanks to the FUSE framework it has been possible to implement the cache as a file system by simply specifying the functions to be called during the execution. In addition, the POSIX standard has made the plug-in implementation a simply mapping between the functions defined within the cache data structure and the API offered by the libcephfs library. If Ceph did not support a POSIX compliant module it would have required to manually implement these procedures in order to create this plug-in which would have been more challenging thus requiring more time.

Finally, as discussed in section 3.6, when working with FUSE there are performance concerns that must always be taken into consideration when developing industrial products.

## 4.4 CEPH'S RADOSGW IN DETAIL

The example that has been presented in the previous section is particularly interesting to show how FUSE and the POSIX standards are very powerful tools for software compatibility and thus software diffusion. Also, it has been important during the work of this Master Thesis for getting used with the Ceph technology and all the paradigms related to distributed storages. In this section another important part of Ceph is going to be explained which is the one concerning Cloud Object Services as the final goal is the development of a persistent cache for S3 Cloud objects.

The RGW (Rados GateWay) is the module of Ceph that provides a complete S3- and Swift-compatible object storage within Ceph. Swift [33] is another Cloud solution meant for the provisioning of storage as an IaaS within the OpenStack Cloud infrastructure. Swift's details are not important for this project but it is important to specify the distinction between it and S3 because even if they are both Cloud object solutions they work with different sets of API.

The following picture shows the architectural model of the RGW which follows the same guidelines of other Ceph's high level modules. In fact, the RGW is a daemon process in charge of interacting with the underlying RADOS layer in order to compute and to serve users' requests.

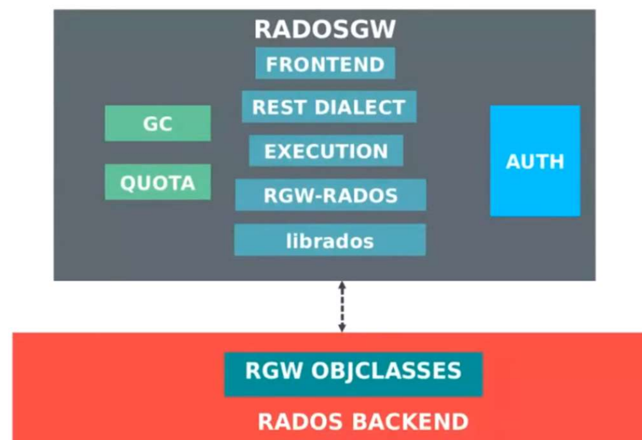



*Image 4.7 – The RGW architecture*

These specific protocols are REST/HTTP based therefore they need appropriate support to enable the communication with users. Along with a radosgw-daemon it is possible to deploy different HTTP frontends that are the support for receiving HTTP requests and sending the appropriate HTTP responses back to clients.

The main front-ends are [29] [36]:

- FastCGI: It needs Apache or other web servers in order to serve HTTP requests.
- Civetweb: It uses the Civetweb HTTP library [34] to implement a standalone synchronous frontend for the RGW.
- Beast: It is a standalone frontend that enables asynchronous communication by utilizing the Boost.Beast and Boost.Asio libraries [35] that respectively work for HTTP parsing and asynchronous network I/O operations.

When deploying a radosgw-daemon it is important to specify in the configuration file "/etc/ceph/ceph.conf" which frontend should be used. Also many other configuration parameters can be specified such as for example the connection port number, the path to the file containing the permission keys to interact with RADOS or the eventual debug log level that is very useful to track the internal operations execution.

Immediately after the front-end module there is the "rest dialect" abstraction component. Its principal goal in the presented model is to highlight the fact that many Cloud object paradigms are supported by the radosgw. At this level all their differences are overcome in order to pass to lower levels more generalized requests to be processed.

The execution module is very important for this work because as it will be better explained in the next chapter it is the one where the cache logic has been inserted in order to create the S3 persistent cache. Within this layer all the logic that implements the procedures to process users' requests is defined and thus to access the appropriate data on RADOS clusters.

Finally, before the librados layer, it is possible to see the so called "rgw-rados" module. This component is in charge of computing the differences between S3/Swift Cloud objects

and Ceph objects. In section number 4.2, the way by which files are mapped over RADOS objects has been discussed. Cloud objects are typically very big while Ceph objects are maintained small. Because of this reason, even if they follow the same model of object storages a mapping between these two different objects types is needed.

Along with these modules, which therefore represent the processing flow of an S3/Swift request by the radosgw, there are some other components:

- GC (Garbage Collector): Within Cloud Storage solutions it is possible to define expiration policies so that a certain object when expires can be deleted by the system. An expiration policy for instance could be expiring those objects that have not been accessed for a certain period. It is important to define these policies in order to remove useless data from the storage as clients are charged by Cloud vendors by the amount of GB that they store on remote COS. This module defines all the cleaning up operations of expired objects by deleting their related data from RADOS clusters.
- Quota: Another type of management that can be performed in order to limit costs and also to optimize storage usage is the definition of “quotas”. They allow buckets owner users to specify limits of the usage of storages as IaaS. For instance, it could be defined the maximum number of objects that can be stored into a specific bucket.
- Auth (Authorization): An important key concept of S3 storages is represented by authority permissions. In fact, it is not enough to just send REST commands to a remote COS. Authentication keys must be passed along with HTTP requests in order to identify users within the service. The RGW’s “auth” module is the one that implements the functionalities to manage users’ access permissions and policies. The presence of this module is one of the reasons that pushed towards the adoption of Ceph’s radosgw for the implementation of the S3 persistent cache because it already implements all the needed support for dealing with this important feature of S3 services.

The following picture summarizes these concepts and shows how the RGW is involved in the processing of S3 users' requests while dialoging with the underlying RADOS layer.

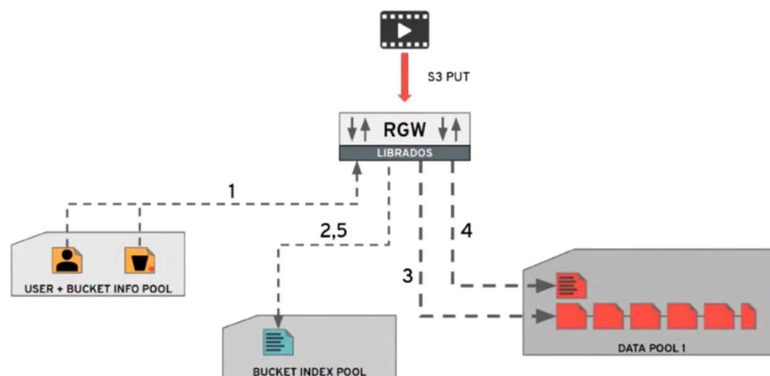



*Image 4.8 – Interaction with the RGW as a Cloud Object Service during a PUTOBJECT request*

Such as it happens during the processing of Ceph file system's operations there are many different pools that must be accessed in order to retrieve the needed data but also to serve clients requests while performing appropriate authorization checks.

When the RGW daemon receives an S3 request through the front-end that has been deployed, it first access the users' metadata information stored into the rgw.info pool. The retrieved objects contain all the information needed to identify clients within the Ceph system as S3 users. If the request has been made by users with appropriate permissions, then the rgw.bucket.index pool is accessed in order to retrieve all the names of the Ceph objects related to the S3 entity that has been specified by the received HTTP request. Finally, all the required data and metadata are accessed within the rgw.data pool.

While the first two mentioned pools are meant to perform Ceph management procedures, it is inside the objects stored into the rgw.data pool that all S3 policies are implemented. As previously mentioned, S3 objects are typically of the order of Gigabytes while Ceph objects usually do not go beyond the size of few Megabytes. An S3 object can be seen as the combination of a head and a tail. The head corresponds to a single RADOS object. It maintains the S3 object metadata such as for example acls, user attributes or the manifest stored into its xattributes. It may also optionally contain the start of the data. The tail instead

is a set of RADOS objects that contain the real data. When an S3 object is stored into RADOS, firstly the tail is stored and only afterwards the head so that eventual error occurred during the storing process would not become persistent within the system. Finally, it is interesting to see that if an S3 object is small enough it is stored and managed as a single head object only.

Finally, the Ceph RGW implements many services that are supported by remote COS such as STS, Encryption, Compression, lifecycle management and archiving. For instance, in section 3.3 the importance of S3 regions to optimize service's performance has been discussed. It is important to highlight how this kind of support can be easily implemented by the geo-replication and the RGW federation of RADOS clusters. Thanks to these types of management, it becomes possible to define zones and thus zone groups which therefore can be used as S3 regions.

It is important to keep in mind all these concepts in order to proceed with the discussion in the next chapter as they are fundamental for the S3 persistent cache implementation.

# 5 S3 CACHE LAYER FOR HYBRID- AND MULTI- CLOUD ENVIRONMENTS

All the details of the technologies that are involved in this project have been explained in the previous sections. In this chapter the S3 cache system that have been designed and developed to satisfy the needs and requirements of intensive data consuming workloads in Hybrid- and Multi- Cloud environments is presented.

To summarize, the idea of implementing a cache layer that fetches data to nodes characterized by computational workloads capable of consuming data at a very high rate, thanks to the introduction of modern computing accelerators such as for example GPUs or FPGAs, seems to be the perfect choice for new prominent Cloud paradigms. With the introduction of such a caching system it should be possible to reach the best tradeoff between the low costs offered by Cloud storage services and the high data rate demanded by these workloads such as for instance those related to Artificial intelligence and Scientific computing tasks.

In order to proceed, why and how the Ceph's RGW has been adopted in the implementation of the S3 cache layer will be clarified. In this way, the new model of Hybrid- and Multi-Cloud environments obtained with the introduction of this new computational layer, will be defined. Finally, an accurate description of the implementation details for the provisioning of caching support to S3 operations will be provided.

## 5.1 RGW FOR S3 CACHING – WHY? HOW?

The S3 protocol defines a set of REST API that can be used to communicate with a Cloud Object Storage in order to take advantage of storage devices offered over the Internet as IaaS. As explained in section 3.4 its operations can be grouped into two main categories which are the so called Simple Storage Service set and AWS Control set. This categorization is due to the fact that S3 storage services offer more functionalities than only

read and write operations. For instance, the identification of users within the service and the definition of management policies for buckets and stored objects are fundamental and cannot be detached from a product that aims to provide a full S3 interface. It is important to highlight this aspect because if it is true that the goal of the cache layer will be focused on the improvement of the performance of reading and writing operations, it is also true that the applications which will interact with it should be able to work without noticing its presence or absence. This means that the system will have to provide a full S3 interface to be transparent from the point of view of users' applications.

After an in-depth study, the unified storage system Ceph has shown good properties that make it particularly well suited for Cloud scenarios such as for example its portability over the heterogeneous devices that are typically deployed in modern server clusters. This makes Ceph very interesting as the environments where it will be introduced are those of Multi- and Hybrid- Cloud systems. Moreover, the abstractions made by the RGW, RBD and CephFS modules make it possible to optimize the management of distributed resources across many computational nodes. Thanks to these software packages the information stored within a RADOS system can be handled at a high level in the forms of Cloud Objects, Blocks or Files and also at a low level in the form of Ceph objects that are therefore stored into the Ceph pools related to the high level data paradigms. For instance, it is possible to interact with S3 objects within a Ceph storage system by using the S3 REST API at a high level or by using the functions defined by the librados library at a low level.

If focusing on the Cloud Object part of Ceph, that has been abundantly discussed in section 4.4, it is possible to notice how many services that support this particular storage paradigm have already been made available by the RGW which provides a full S3 and Swift compatible interface. In fact, mechanisms such as for example user authorization control, data encryption or object garbage collector and also support to ACL or IAM policies can already be employed in Cloud solutions created with Ceph. Because of this, inserting the cache logic inside the RGW appeared to be the most reasonable choice as it allows the

work to focus only on the implementation of those routines needed to support the caching of S3 objects during read and write operations.

Therefore, the decision has been made by design of inserting the cache logic inside the RGW's execution layer that is specified by the model shown in the image 4.7. There are many modules involved in the definition of the logic for processing client applications' requests. Because of this, in order to proceed, the main components that directly affect the execution of rgw-daemons within Ceph clusters must be presented and well understood.

These radosgw key packages can be summarized as [38]:

- *rgw_main*: It is the module that contains the "main" of the program that defines the behavior of a rgw-daemon. It includes all the functionalities involved during the startup and all the configuration procedures that must be performed depending on the type of the front-end that has been deployed.
- *rgw_civetweb_frontend*: It defines the behavior of the radosgw standalone synchronous front-end. When applications communicate with it they must wait for requests to be processed before responses can be sent back. In fact, this is the module that implements the synchronous communication paradigm for client-radosgw interactions.
- *rgw_asio_frontend*: It defines the behavior of the radosgw standalone asynchronous front-end. Contrary to the previous one, applications are free to continue with their normal execution as an answer is sent back independently from the outcome of processing a certain request and before its execution has effectively terminated.
- *rgw_process*: It is the module that implements the logic for processing users' requests. The final result that is obtained with the computation of the procedures characterizing the semantic of the operations specified by users is returned to the calling front-end that has received the request. In this way, the HTTP response will be finally sent back to the user with the appropriate response code and data. This module is very important because it defines the core structure for serving all S3 (and Swift) operations.

When the front-end receives a request it first parses the parameters that have been included within the received HTTP packet and then calls the *process_request()* function defined by the *rgw_process* module. At this point the differences between S3 and Swift paradigms are overcome by calling the appropriate data structures in order to represent the specific requests sent by client applications. With this design next procedures are performed without any Cloud-object paradigm-specific constraint by following a general execution scheme that has been made possible only thanks to the properties offered by the full object-oriented programming environment of C++.

The data structures that are of interest for this project work are those related to S3 operations. In particular, as strictly connected to the execution of operations concerning S3 requests it is important to mention the *rgw_op*, *rgw_rest* and *rgw_rest_s3* modules.

It is not important to enter any further in the details of the RGW source code that can be easily consulted on the Ceph's git-hub page at the path src/rgw. However, the computational flow defined by the *process_request()* function shows a common scheme that must be understood well before continuing.

From now on, all examples and related discussions will refer only to the S3 REST API as the cache layer will not include caching support for the Swift paradigm.

To summarize, when a front-end receives users' requests, after parsing the parameters that have been included within the specific HTTP packet, it calls the *process_request()* function, defined by the *rgw_process* module, which implements the following common computational flow:

1. A handler data structure is instantiated in accordance to the communication type of the received request by the deployed radosgw front-end (REST handler).
2. Depending on the type of the operation, the handler instantiates a data structure that represents the specific service requested by the user (S3 operation). This structure also includes all the parameters obtained by parsing the information contained in the head and the body of the received HTTP request packet.

3. The client user is identified within the Ceph system by checking the information maintained in the rgw.info pool in relation to the access keys provided with the REST request.
4. The request is processed with the semantic defined by the specified service operation. This is done by calling the *rgw_process_authenticated()* function that is also defined by the *rgw_process* module.

These steps are very important because they show how the radosgw manages the differences between S3 and Swift and also how it separates the management of users' authentication routines by those concerning the effective execution of operations' semantic.

The fourth step is particularly important because it is the one where the effectiveness of serving the requests once users have been authenticated within the S3-Ceph system is defined. Its computational behavior is represented by the *rgw_process_authenticated()* function which in turn can be summarized by the following steps:

1. operation-specific *permission check*: verifies that the identified user has the permission to execute the requested operation.
2. operation-specific *execution*: the operations that undertake the semantic of the requested REST operation are executed.
3. operation-specific *complete*: in the case of S3 REST operations the HTTP response is created and sent back to the front-end.

The following image provides a graphical view of just mentioned workflow.

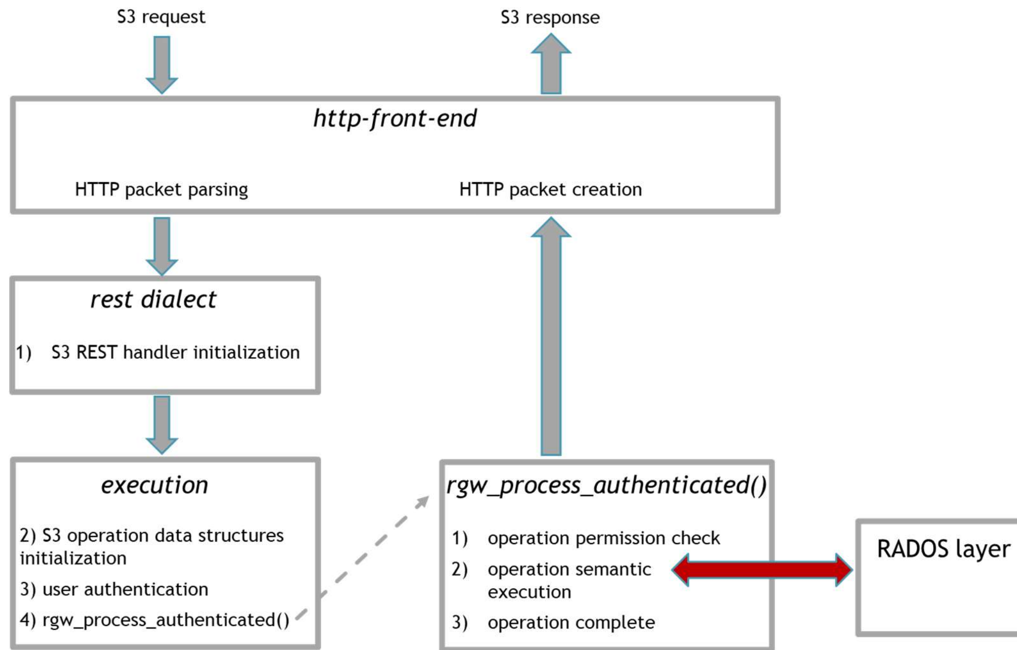



*Image 5.1 – Radosgw-daemon workflow*

The importance of user authentication mechanisms within S3 systems should be clear after presenting the computational flow of S3 operations processing. For this reason, in the next section a full explanation of the decisions made by design for the management of users identification will be provided, along with other important Ceph-S3 related concepts that must be taken into consideration.

## 5.2 S3 CACHE AUTHENTICATION MANAGEMENT AND PRE-IMPLEMENTATION CONSIDERATIONS

In the previous section the need of hiding the presence of the cache layer to user applications, that is the main factor that have pushed for the adoption of Ceph in the implementation of the S3 caching system, has been highlighted. Moreover, the importance of the user authentication process has been shown as it plays an important role in S3 storage systems. For these reasons, how the applications will interact with the cache layer and how to manage the problem of authorization and permission controls must be discussed.

As a choice made by design, the applications that will interact with the cache layer will work transparently from the prospective of authentication procedures. The only difference is represented by the need of specifying the authorization keys (at ~/.aws/credentials, as parameters of the sdk functions or as environment variables) that authenticate the user within the Ceph system and not within the remote COS. A file or data structure that specifies the mapping between cache and COS users should be passed to a cache rgw-daemon during its initialization (an XML file for example) so that when a certain user sends a request, the cache system knows with which credentials it has to interact with the remote storage service in order to provide caching support.

This implementation of the authentication control is only a possible solution. There are other possibilities which may provide even higher security and privacy levels. However, this is not central for this project as the work focused more on the enhancement of the performance related to the operations of reading and writing S3 objects.

The next picture shows how the computational model changes with the introduction of the persistent cache layer into a Cloud environment that utilizes S3 storage services.

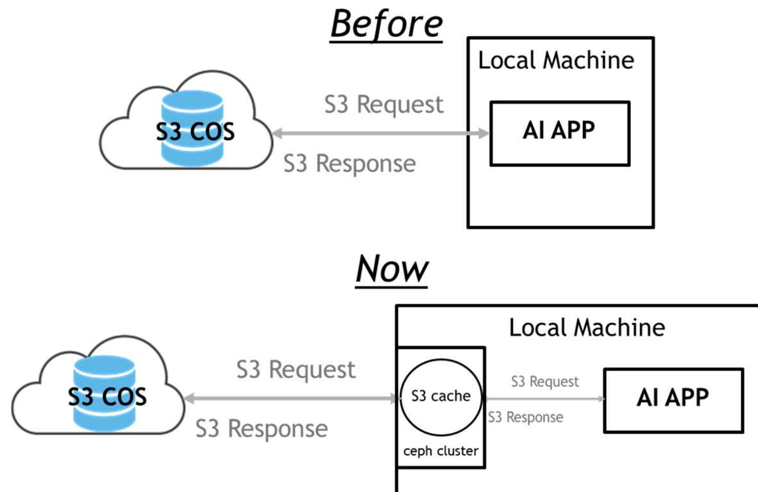



*Image 5.2 – Model changes with the introduction of the S3 cache layer*

In past scenarios, applications residing on local machines could only take advantage of S3 services via internet connections. Now, thanks to the introduction of the caching system, they can send local S3 requests with important results on reading and writing operations as explained in Chapter 6.

The presented model also shows how the cache layer introduces an intermediate level in the authentication process between client applications and remote COS which is the reason for the need of an appropriate management technique such as the proposed user mapping. It is important to understand that this type of management has been fundamental as it allowed the development of such a cache and thus the improvement of Cloud objects I/O performance. However, this is only a first proposal and can therefore surely be extended in future works.

In section 3.1 the general key concepts of cache memories have been presented along with flushing mechanisms that are meant to reduce memory occupation when data is not needed anymore or to manage situations of congestion. Cache devices are typically characterized by small memory size and need therefore an appropriate data eviction policy in order to limit memory utilization and thus not compromise storage systems' overall performance during I/O procedures.

The S3 service shows many mechanisms for the management of stored objects. For example, it gives the opportunity of defining "quotas" and "expiration policies". Quotas are meant to optimize storage services usage by limiting the number of objects in certain specified buckets. They can be seen as a preventative technique as they do not allow storing objects when the defined thresholds are reached. Expiration policies instead allow defining when certain objects should be deleted from the storage system. They can therefore be seen as an active technique.

All these mechanisms can surely be adopted in order to optimize the usage of the cache memory. However, only the second one is appropriate for the implementation of an eviction technique as the problem is deleting objects that have already been stored and not denying the memorization of new ones.

As already mentioned in the previous sections, it is possible to interact with RADOS S3 objects at low or high levels. Expiration policies defined to free the cache memory when it becomes full or when it reaches certain occupation thresholds could be seen as a high level implementation of the eviction's mechanisms. According to the S3 API semantic it is possible to define time intervals in the order of days or a specific date time [39] for the expiration of stored objects. On the other hand, if interacting with the Ceph storage system with the low level interface defined by the librados library and thus interacting directly with the RADOS storage, it is possible to eventually delete Ceph objects containing data of cached S3 objects in the rgw.data and rgw.index pools.

At the current state, the persistent cache system that has been developed does not have an implemented eviction policy which can therefore be implemented in future. This can be done by utilizing the librados functions and thus by working directly with the RADOS layer at low level. Another possibility is working at high level by defining expiration policies with the S3 bucket lifecycle management API. In any case this has not represented a problem during the execution of experiments as the final goal of this project is the demonstration of how it becomes possible to enhance I/O operations performance by

providing S3 caching support mechanisms to high rate data consuming computational workloads.

Finally, one last aspect regarding the specific S3 operations supported by the RGW should be considered in order to proceed with the implementation details. As already explained, when the radosgw front-end receives a REST HTPP request it calls the *process_request()* function defined by the *rgw_process* module which will compute the specified operation. In order to add caching support to the execution of S3 objects I/O operations it is important to understand how they are managed within the Ceph system. The RGW module defines a list of operations which remap the supported S3 ones. These operations are [40]:

| RGW_OP_UNKNOWN | RGW_OP_DELETE_BUCKET_POLICY |
| --- | --- |
| RGW_OP_GET_OBJ | RGW_OP_PUT_OBJ_TAGGING |
| RGW_OP_LIST_BUCKETS | RGW_OP_GET_OBJ_TAGGING |
| RGW_OP_STAT_ACCOUNT | RGW_OP_DELETE_OBJ_TAGGING |
| RGW_OP_LIST_BUCKET | RGW_OP_PUT_LC |
| RGW_OP_GET_BUCKET_LOGGING | RGW_OP_GET_LC |
| RGW_OP_GET_BUCKET_LOCATION | RGW_OP_DELETE_LC |
| RGW_OP_GET_BUCKET_VERSIONING | RGW_OP_PUT_USER_POLICY |
| RGW_OP_SET_BUCKET_VERSIONING | RGW_OP_GET_USER_POLICY |
| RGW_OP_GET_BUCKET_WEBSITE | RGW_OP_LIST_USER_POLICIES |
| RGW_OP_SET_BUCKET_WEBSITE | RGW_OP_DELETE_USER_POLICY |
| RGW_OP_STAT_BUCKET | RGW_OP_PUT_BUCKET_OBJ_LOCK |
| RGW_OP_CREATE_BUCKET | RGW_OP_GET_BUCKET_OBJ_LOCK |
| RGW_OP_DELETE_BUCKET | RGW_OP_PUT_OBJ_RETENTION |
| RGW_OP_PUT_OBJ | RGW_OP_GET_OBJ_RETENTION |
| RGW_OP_STAT_OBJ | RGW_OP_PUT_OBJ_LEGAL_HOLD |
| RGW_OP_POST_OBJ | RGW_OP_GET_OBJ_LEGAL_HOLD |
| RGW_OP_PUT_METADATA_ACCOUNT | **/* rgw specific */** |
| RGW_OP_PUT_METADATA_BUCKET | RGW_OP_ADMIN_SET_METADATA |
| RGW_OP_PUT_METADATA_OBJECT | RGW_OP_GET_OBJ_LAYOUT |
| RGW_OP_SET_TEMPURL | RGW_OP_BULK_UPLOAD |
| RGW_OP_DELETE_OBJ | RGW_OP_METADATA_SEARCH |
| RGW_OP_COPY_OBJ | RGW_OP_CONFIG_BUCKET_META_SEARCH |
| RGW_OP_GET_ACLS | RGW_OP_GET_BUCKET_META_SEARCH |
| RGW_OP_PUT_ACLS | RGW_OP_DEL_BUCKET_META_SEARCH |
| RGW_OP_GET_CORS | **/* sts specific*/** |
| RGW_OP_PUT_CORS | RGW_STS_ASSUME_ROLE |
| RGW_OP_DELETE_CORS | RGW_STS_GET_SESSION_TOKEN |

| | |
|---|---|
| RGW_OP_OPTIONS_CORS | RGW_STS_ASSUME_ROLE_WEB_IDENTITY |
| RGW_OP_GET_REQUEST_PAYMENT | **/* pubsub */** |
| RGW_OP_SET_REQUEST_PAYMENT | RGW_OP_PUBSUB_TOPIC_CREATE |
| RGW_OP_INIT_MULTIPART | RGW_OP_PUBSUB_TOPICS_LIST |
| RGW_OP_COMPLETE_MULTIPART | RGW_OP_PUBSUB_TOPIC_GET |
| RGW_OP_ABORT_MULTIPART | RGW_OP_PUBSUB_TOPIC_DELETE |
| RGW_OP_LIST_MULTIPART | RGW_OP_PUBSUB_SUB_CREATE |
| RGW_OP_LIST_BUCKET_MULTIPARTS | RGW_OP_PUBSUB_SUB_GET |
| RGW_OP_DELETE_MULTI_OBJ | RGW_OP_PUBSUB_SUB_DELETE |
| RGW_OP_BULK_DELETE | RGW_OP_PUBSUB_SUB_PULL |
| RGW_OP_SET_ATTRS | RGW_OP_PUBSUB_SUB_ACK |
| RGW_OP_GET_CROSS_DOMAIN_POLICY | RGW_OP_PUBSUB_NOTIF_CREATE |
| RGW_OP_GET_HEALTH_CHECK | RGW_OP_PUBSUB_NOTIF_DELETE |
| RGW_OP_GET_INFO | RGW_OP_PUBSUB_NOTIF_LIST |
| RGW_OP_CREATE_ROLE | RGW_OP_GET_BUCKET_TAGGING |
| RGW_OP_DELETE_ROLE | RGW_OP_PUT_BUCKET_TAGGING |
| RGW_OP_GET_ROLE | RGW_OP_DELETE_BUCKET_TAGGING |
| RGW_OP_MODIFY_ROLE | RGW_OP_GET_BUCKET_REPLICATION |
| RGW_OP_LIST_ROLES | RGW_OP_PUT_BUCKET_REPLICATION |
| RGW_OP_PUT_ROLE_POLICY | RGW_OP_DELETE_BUCKET_REPLICATION |
| RGW_OP_GET_ROLE_POLICY | **/* public access */** |
| RGW_OP_LIST_ROLE_POLICIES | RGW_OP_GET_BUCKET_POLICY_STATUS |
| RGW_OP_DELETE_ROLE_POLICY | RGW_OP_PUT_BUCKET_PUBLIC_ACCESS_BLOCK |
| RGW_OP_PUT_BUCKET_POLICY | RGW_OP_GET_BUCKET_PUBLIC_ACCESS_BLOCK |
| RGW_OP_GET_BUCKET_POLICY | RGW_OP_DELETE_BUCKET_PUBLIC_ACCESS_BLOCK |

*Table 5.1 – S3 RGW operations*

This list is particularly interesting because it shows what are the specific S3 operations that are supported by the RGW but also because it presents some additional operations that have been introduced which extend the S3 service such as those here presented in the categories “rgw specific”, “sts specific”, “pubsub” and “public access”. There are many RGW-specific S3 operations worth of mentioning such as those implementing the pub-sub paradigms. However, those of interest for this work are those remapping the S3 GetObject, ListObjects and PutObject which are respectively the RGW_OP_GET_OBJ, RGW_OP_LIST_BUCKET and RGW_OP_PUT_OBJ.

In the next sections the details about the caching support provided for these operations will be presented.

## 5.3 GETOBJECT

The S3 GetObject operation defines a specific request for downloading an S3 object contained into a bucket of a remote COS. The logic behind the implementation of caching support for this operation specifies that, with the introduction of the cache layer, if an object is available within the cache memory, the time required to transfer data to the client application is way lower as already on the same locality.

The possible situations that can occur during reading operations with the presence of the cache layer are:

- HIT: the object is already available within the cache
- MISS: the object has not been cached yet

HIT cases are expected to be way faster than the MISS ones which have instead to pay the additional cost for the caching service.

The applications should send a request to the persistent cache by specifying the cache credentials. These credentials will be automatically remapped to the COS ones when the S3 cache interacts with the specific remote Cloud Object Storage related to certain client users (the COS address is another parameter that should also be specified within the user mapping file because many users may work with different S3 storage services).

As the authorization management is automatically provided, the user sending the request will be identified by the RGW. At this point, when the *process_request()* function is called, the inserted caching logic first checks if the object is already available within the RADOS back-end system. This information is maintained inside a C++ object that represents the state of the cache which keeps track of the already available objects and buckets. The cache state is initialized during the startup of the caching rgw-daemon by the *rgw_main* module that has been appropriately modified.

If the request is a HIT, the processing of the RGW_OP_GET_OBJ can continue without the need of any additional operation. However, in case of a MISS caching support must be performed which means providing the required object to the RADOS back-end cluster.

The management of a cache MISS can be summarized as:

1. Downloading the requested object by the remote COS using the credentials and the address specified by the user map according to the identity of the user that has been authenticated by the cache.
2. Saving the downloaded object to the RADOS back-end system.
3. Continuing with the normal processing flow of the RGW as defined in section 5.1.

In order to implement the first point, it was necessary to create an S3 client with the APIs define by the C++ sdk available on the AWS git hub repositories [24].

For the implementation of the second point there were two possible solutions: using high or low level type of communications. The problem of interacting with the low level functions defined by the librados library is that when saving S3 objects they should also be translated into Ceph objects for performance and RGW compliant management matters. Because of this in order to keep the proposed solution as much simple as possible the decision was made by design of saving the S3 objects with the high level S3 API thus implementing another dedicated S3 client which will save the downloaded object by sending a PutObject request to the local cache layer.

This solution pays for its simplicity in performance. However, if it is true that communicating directly with the RADOS layer would have enhanced the performance reducing the time of saving Cloud objects to the cache, it is also true that the overhead introduced by the implemented S3 client would affect only the cache's radosgw front-end. Also, it is normal to have worse performance in the presence of MISS cases as the benefit provided by the cache is effectively implemented by the HIT so that what really matters is increasing the probability of HIT cases in order to increment the HIT rate.

Finally, once the object has been saved into the cache system, the computation can simply continue its natural flow because the object is now available and therefore when the RGW arrives at the point of reading the appropriate Ceph objects from the RADOS layer they will be found in their appropriate pools.

The next image summarizes the execution flow during the processing of RGW_OP_READ_OBJ requests with the introduction of the cache layer.

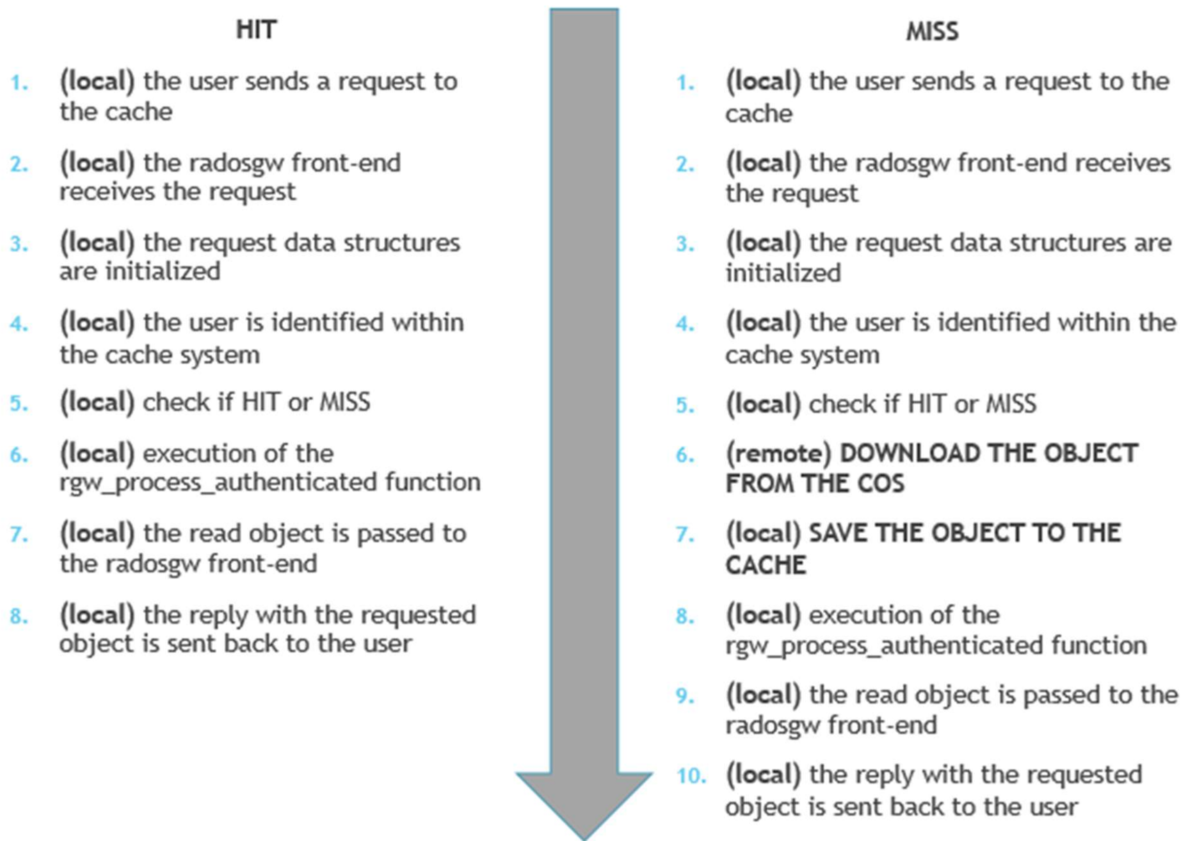



*Image 5.3 – Processing flow of RGW_OP_GET_OBJ operations during HIT and MISS scenarios*

## 5.4 PUTOBJECT

The final goal of cache memories is improving the overall I/O operations performance of storage systems. In this case, as specifically related to S3 storage services, it means reducing the time required to perform write operations and also possibly limiting their number in order to reduce clients' expenses as it has been discussed in section 3.4.

When developing a cache system, the main policies that can be undertaken to implement writing operations are those of write-back and write-through which are distinguished by the specific moment when the commit of write operations is sent back to the client. The write-back policy seems to be the most well suited to obtain the expected benefits in performance and cost reduction. Unfortunately, when working with this type of semantic, its lack of reliability must be remembered as possible errors can occur before committing the write operations to persistent storage devices, which would cause data inconsistency within the entire storage system. However, Ceph provides a reliable environment while implementing a CP storage which means that if errors occurred, the system wouldn't be affected by data inconsistency. This is the reason why the implemented technology is defined as a persistent cache.

In the PutObject scenario with the implemented write-back policy, clients only have to send the request to the cache which will then have the responsibility of uploading it to the remote COS at the most appropriate moment. In this way, users receive a commit and thus a HTTP reply for the write operation after writing the object on the local caching support. Thanks to this design the overhead introduced by the Internet during communications with remote resources is bypassed and the time required to execute write operations is drastically reduced.

In section 3.3 a deep insight of S3 objects related concepts has been provided. The object versioning is the main characteristic of this storage paradigm that must be considered when implementing write-back policies. As already explained, many versions of the same object can reside on a certain Cloud Object Storage. This means that in order to implement a

perfect alignment between the data stored on the cache memory and on the remote COS, every upload operation should correspond to a PutObject request made by client applications to the cache layer.

The proposed solution has been implemented with the logic of enhancing applications performance and of reducing the overall operations number with the assumption that clients performing many write operations of the same objects are not interested in maintaining all related versions. However, if this wasn't true, it is always possible to tune the cache in order to satisfy this requirement as well while still taking advantage of reduced execution times.

The following picture summarizes the general scheme of the computational flow of S3 objects write operations with the introduction of the cache layer.

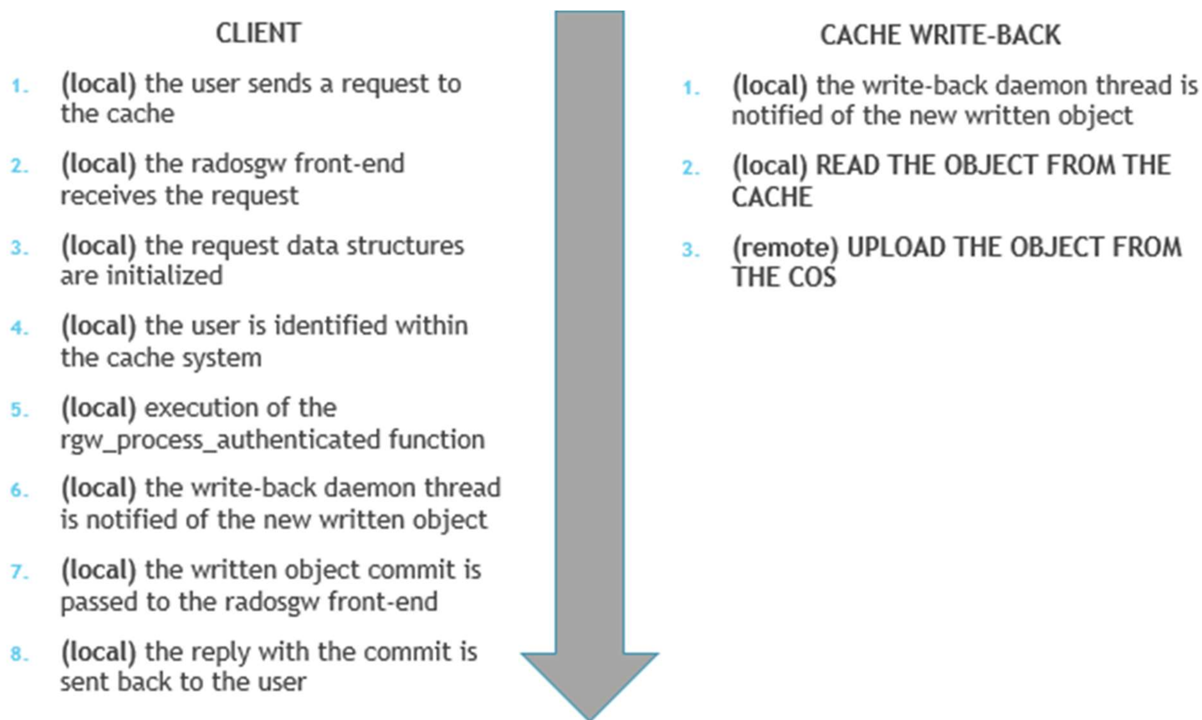



*Image 5.4 – Processing flow of RGW_OP_PUT_OBJ operations with the implemented write-back policy*

Thanks to this model, it is possible to appreciate that there is an important difference in the provisioning of caching support in comparison to what happens for the GetObject

operation. In fact, caching procedures are executed after and not before the *rgw_process_authenticated()* function is computed.

The write back procedure has been implemented as a daemon thread that constantly waits for new write requests. When a client application writes a new object into the cache, the daemon reads the object that has been saved from the Ceph system and then writes it to the remote COS according to the address and the user credentials that have been specified during the initialization of the cache rgw-daemon. Read and write operations have been implemented by adopting the same S3 sdk as for the GetObject caching support in order to implement appropriate S3 for clients that utilize the GetObject and PutObject operations via the HTTP REST API defined by the Simple Storage Service paradigm.

A reference to the new objects written to the cache which need to be uploaded to the remote COS are maintained in a queue. The write-back daemon continuously reads the first element from the queue and then aligns the S3 object with the storage service.

With this implementation there is only one daemon thread and it therefore may happen that when it reads the first element from the queue that element has already been uploaded. This is due to the response speed provided by the write-back support that can in any case be improved by deploying multiple write-back daemon threads. Therefore, it may happen that clients write the same object many times before it is uploaded to the remote COS as shown by the following picture.

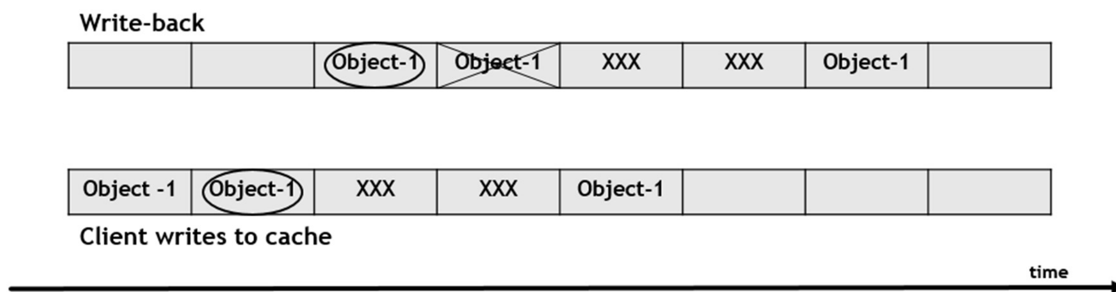



*Image 5.5 – S3 objects write-back*

It may happen that more versions of the same objects are written to the cache before the object is effectively uploaded to the appropriate COS by the cache write-back daemon

thread. This is very powerful for limiting the number of S3 operations which have important consequences in clients' expenses. However as previously mentioned, this causes a versioning misalignment in the data stored into the cache and the COS. Applications therefore must not be interested in the object versioning.

This scheme is very interesting because it shows the potential of the caching system in the enhancement of write operation performance and also in reducing clients expenses. It demonstrates the tradeoff between fast response and data alignment properties that must be tuned according to specific applications requirements:

- From one side, the more the cache write-back procedure waits, the more it is possible to reduce the number of I/O operations.
- From the other side, the more the cache write-back procedure waits, the more the data is misaligned with the remote COS.

However, in any case, independent from the specific policy adopted, this model is capable of providing better performance as it succeeds in reducing the time required for write operations.

Additional details will be presented in chapter 6 along with the experiments and the results.

## 5.5 LISTOBJECTS

The caching support provided for GetObject and PutObject operations has been implemented by adding functionalities to the execution flow of the corresponding RGW operations respectively before or after the *rgw_process_authenticated()* function call. In order to serve the execution of the ListObjects operation there are different things that must be considered in order to provide the appropriate caching support. In contrast to what happens with the previously mentioned S3 service functions the normal RGW execution flow must be modified, as downloading all the objects that have to be included into the objects list in order to take advantage of the already available procedures is not a reasonable solution. Also, the final result for this specific request involves objects' metadata that is maintained in different forms within cache and COS supports.

The HTTP response that has to be built needs to simultaneously take into consideration both the information stored locally and remotely. This makes it necessary to change the normal execution flow of the RGW as objects that have been written to the cache may not have been uploaded yet to the remote COS and many objects stored within the specified bucket may not be already available within the cache.

The Image 5.6 shows the flow for the execution of the RGW_OP_LIST_BUCKET operation which remaps the S3 ListObjects one.

To obtain the list of the objects that are stored by the Cloud Object Service, with also their related metadata such as modification time and object size, another S3 client has been implemented.

Firstly, the updated list of the objects stored into the specified bucket must be downloaded sending an appropriate request to the remote COS. Then, when the list has been returned it is important to include all the information related to those objects stored into the cache that have not been uploaded yet. This is important to overcome possible misalignment between cached objects and the information maintained by the remote storage service.

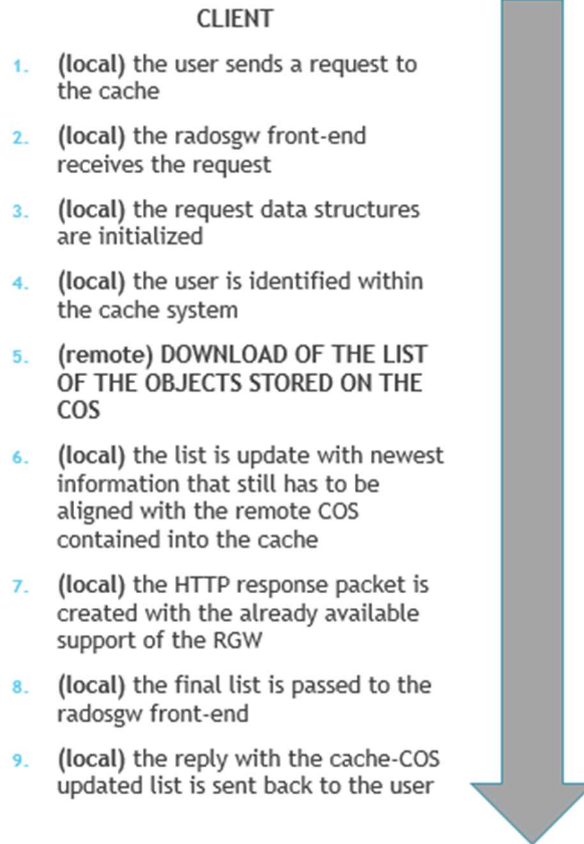



*Image 5.6 – Processing flow of RGW_OP_LIST_BUCKET operation with the introduction of the cache*

The ListObjects operation is very important for providing a full operative S3 cache system because it is typically utilized at the beginning of the most business processes that work with the S3 storage services. For instance, applications working with objects stored within a bucket do not have any more information than the bucket name and authorization credentials. This makes the execution of the ListObjects fundamental to acquire the information needed to be effectively capable of performing computational tasks as a direct access to stored objects is not possible without specifying their names. Finally, thanks to this operation it is possible to check the availability of required objects which may avoid the computations of tasks that cannot terminate and therefore reduce the costs by not unnecessarily occupying computational resources.

## 5.6 POST-IMPLEMENTATION CONSIDERATIONS

In the previous sections an in-depth description of the implementation details of the caching support provided to GetObject, ListObjects and PutObject functions have been given. In the next chapter the discussion will proceed by presenting the experiments that have been performed to test the performance of the cache while executing computational tasks characterized by many read and write procedures as the final goal is the improvement of such operations.

To complete and conclude this chapter it is important to consider how the choices made during the cache development have made it possible to achieve the results. For instance, as explained in section 3.4, when working with S3 storage systems there are some best practices and common guidelines that should be followed in order to obtain the best results as possible.

Significant delays in serving clients requests have been noticed after a first implementation of the cache system. If following the guidelines provided by AWS engineers [25] it is advised to set short timeout intervals for requests retransmissions. This is due to the fact that S3 services are implemented as Distributed Systems and it may happen therefore that the path chosen to serve a certain request may not be the best possible one during the first attempt which is typically less efficient than sending and staring a new service request.

This policy is already implemented by the S3 sdk that automatically resends the same HTTP request if a reply is not received before a certain time interval. This practice is very powerful when interacting with business S3 Cloud Services but it is detrimental when working with the developed local cache system.

As a choice by design, the caching support has been implemented by using the S3 sdk for reading and saving the Cloud objects that are stored within the cache. This may become a possible bottleneck of the deployed radosgw front-end if not properly managed. If it is true that in Distributed Systems, such as the RADOS back-end support, it is good practice to set short timeouts to reduce the probability of worst path situations, it is also true that this

possible scenario will never happen when interacting with the cache as its resources are implemented to work on the same locality along with client applications.

For this reason, huge timeout intervals have been set for the S3 clients providing caching support. Thanks to this measure the cache system has shown an important improvement of performance by 60% which means that the RGW requests management may become a possible bottleneck when serving many requests simultaneously. However, as it will be shown in the next chapter, this problem may be overcome by introducing more Ceph nodes within the RADOS layer.

An important property of S3 services is represented by the possibility of downloading only specific byte-ranges of a stored object. In previous sections a typical implementation of cache systems has been presented which are characterized by managing stored information within fixed size blocks. However, this is not possible when working with the S3 API because during uploading procedure only the entire object can be uploaded and not only part of it. Storing objects related information within fixed size blocks will surely improve the cache memory management and for this reason some more details regarding the S3 Ceph implementation are presented.

One of the S3 Ceph specific operations that are implemented by the RGW module is the ObjectAppend which allows the modification of an already existing object by adding the new information without the need of rewriting the entire object. At the current state the cache system does not provide support for the management of objects within fixed size blocks because there are many more assumptions to be considered (for example the management of byte-range objects related metadata) and as said before the strategy of keeping the solution as simple as possible has been followed. In any case, these concepts are worthy of being mentioned because they may be very useful for future extensions of this technology.

Along with the Append Object operation there are some more interesting radosgw services that have not been presented yet. It is not important to master these concepts but it is interesting to mention some radosgw study related concepts.

These services can be summarized as [36]:

- STS (Security Token Service): It allows the use of external services instead of the classic S3 model to perform user authentication procedures.
- Metadata Elastic Search: It allows the process of query related to stored objects metadata.
- Cloud Sync Module: It allows the backup of a Ceph S3 storage on a remote S3 service. The sync modules to perform these operations from a remote COS to Ceph are currently in development.
- Archive Sync Module: It allows the creation of an archive zone that can thus be used to perform archiving operations on S3 Ceph objects.
- Pub Sub Module: It allows the subscription of notifications on modification events within the S3 Ceph system.

These services may allow new possible extensions and improvements. The STS service for instance could be introduced to extend the user-mapping authentication management of the cache. Moreover, as the final goal of the cache system is the provisioning of S3 objects stored on a remote storage service, the Cloud Sync module that is currently under development may add new functionalities to the RGW which may make it possible to undertake new cache implementations.

These assumptions and studies are very important as they have been central in the implementation of such a cache service. Now that all these concepts have been finally clarified, it is possible to explain and appreciate the results obtained with the experiments that are going to be presented in the next chapter.

# 6 EXPERIMENTS AND RESULTS

The data transfer speed of the Internet's infrastructure has become the most prominent bottleneck for those computing workloads that rely on Cloud based storage services. The objects caching system described in this Thesis work aim at improving the performance, from storage perspective of such workloads. For instance, new Cloud environments adopting the S3 cache layer will benefit from faster I/O operations on objects stored on public Cloud object storage services.

The experiments performed are presented in the next sections. Firstly, a summary of the adopted benchmarks and information about the execution environment will be provided. Then, results comparisons and related considerations will be analyzed according to the achieved system's capabilities.

## 6.1 BENCHMARKS AND EXPERIMENTS' INFRASTRUCTURE

To be completely capable of understanding the results that are going to be presented in the next sections it is fundamental to have a clear vision of the adopted benchmark's workflows. Read/write of objects was monitored to measure the benefits of the proposed caching layer on benchmark's performance.

The benchmarks used for this evaluation are:

- A custom benchmark focused on monitoring the behavior of GetObject operations: multiple threads sending simultaneously GetObject requests to the cache. Each thread initially obtains a list of 100 objects that are stored into an S3 bucket and then, after randomly shuffling the order of the names contained in the list, starts issuing a GetObject request for each object. Each Object is 150 MB in size The program terminates only when every thread has completed its execution. The goal of this benchmark is twofold: i) model the access pattern of multiple concurrent

applications; ii) enables evaluating the performance of the cache with multiple cache hit values.

- A custom benchmark focused on monitoring the behavior of PutObject operations: multiple threads sending simultaneously PutObject requests to the cache. Each thread sends 75 PutObject requests to store 50 different objects into the cache. Each object is 100 MB in size. The program terminates after every thread has completed. The goal of this benchmark is twofold: i) model the access pattern of multiple concurrent applications; ii) evaluate reduction in number of PutObject operations to the remote S3 location.
- A real application likely to be executed on the Cloud: training the resnet deep neural model on the ImageNet dataset. The implementation of resnet is available as part of the open source TensorFlow official models [41]. This is a good representation of Cloud application using object storage where most of the accesses are object reads.

All the experiments were performed on one IBM Power8 server (20 cores, 1 TB RAM) part of the Hermes cluster at IBM Research – Ireland. This server was used to deploy multiple virtual machines used to model a distributed infrastructure, helping also in rapidly changing the infrastructure template for the experiments. In the specific, each node of the Ceph cluster serving as backend for the object caching was running on a dedicated KVM [42] virtual machine. The S3 endpoint was exposed via a custom radosgw-daemon. The execution model can be referred to the one that has been shown in image 5.2.

Two system configurations are used in this evaluation:

- 3 VMs
  - 100GB virtual disk for each VM
  - 8 CPU cores for each VM
  - 4 GPUs - NVIDIA Tesla P100 with 16GB memory (resnet training sessions)
- 5 VMs
  - 100GB virtual disk for each VM
  - 6 CPU cores for each VM

Each system configuration is meant for measuring the impact of the cache back-end to the overall performance. The actual caching logic remains unchanged in either of the configurations. All the benchmarks selected were evaluated with both system configurations.

The experiments' results will be presented in the following sections along with some additional details and related considerations.

## 6.2 CACHE GETOBJECT PERFORMANCE WITH THE S3 CUSTOM BENCHMARK

The improvements that are expected with the introduction of the developed cache layer are shorter time intervals to serve GetObject operations and a decrease in the number of the requests that are directly sent to the remote S3 service.

As already anticipated in section 5.3, MISS operations are characterized by worse performance than working without the cache layer. However, HIT operations are capable of providing very fast responses to client applications.

For these reasons, in order to see better applications' performance there are two possible scenarios:

- The cache is capable of prefetching foreseeing future applications' requested objects.
- Client applications' workflows are characterized by many read operations over the same objects.

This work focused on the second point during experiments execution and caching development but it is not hard to understand how the first one can be easily implemented if taking into account specific business tasks' characteristics. In addition, by adopting common cache policies such as those explained in section 3.1 it is possible to generally grant some improvements in this sense. The implemented caching system at the current state does not support data-prefetching.

However, if these types of workloads read the same objects more than once, there will surely be a reduction in ReadObject requests sent to the COS as they will correspond to only one cache MISS and many HITs. This is very important as it makes users interaction with S3 services less expensive while adopting the implemented caching storage system.

The following charts show the results that have been obtained by executing the read objects custom benchmark with the COS and the 3 VMs setup persistent S3 cache. The presented columns refer to different benchmark's configurations, respectively with 10, 20 and 50

simultaneous threads. Represented results show therefore cache's behavior depending on different applications workloads in serving every single read operation.

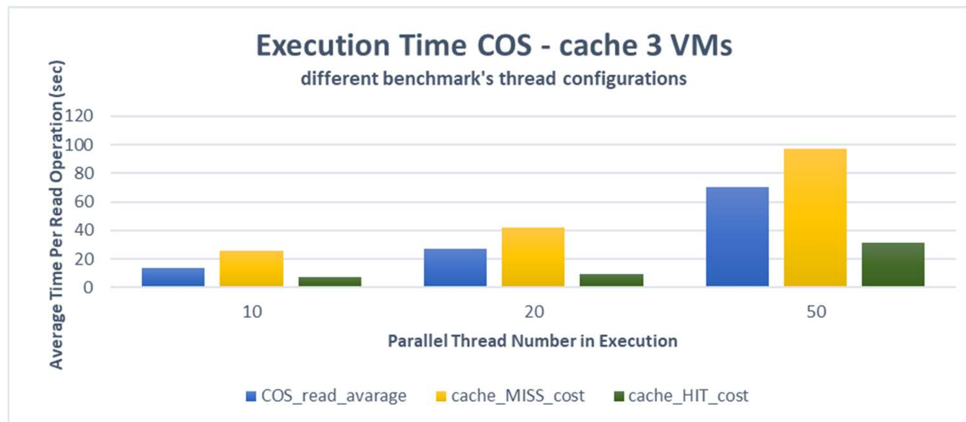
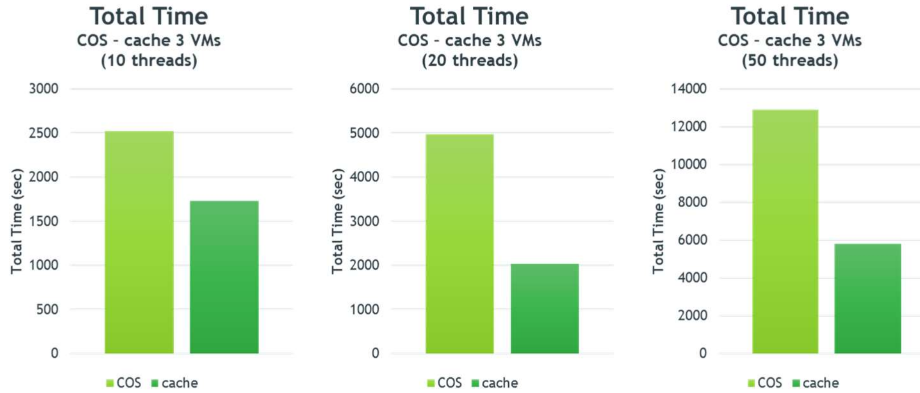



*Chart 6.1 – GetObject execution time COS – cache 3 VMs*

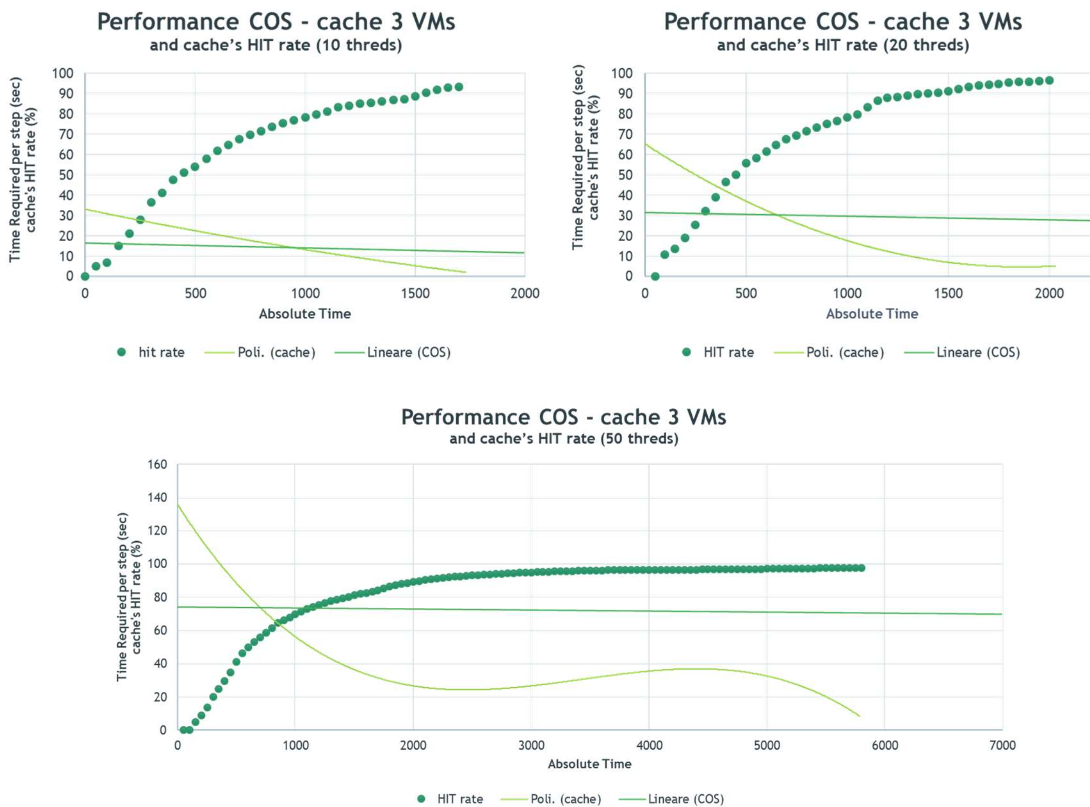
As expected, the registered average cost of HIT operations is lower than MISS ones. In fact, during MISS cases, before being able to return the requested object, the cache must perform all needed procedures to download the required data from the remote COS and then save it into the Ceph back-end system. HIT cases show a very low average time, thanks to the cache infrastructure proximity to workloads' computational nodes. This characteristic if properly exploited will result in performance improvements for applications with high cache's HIT rates as shown in charts 6.2.

Another important achievement is represented by the difference between the times registered when executing GetObject operations with the remote COS or the cache. COS operations respectively register a cost of 13, 27 and 70 seconds, cache's HITs of 7, 9 and 31 seconds and MISSs of 25, 41 and 97 seconds. COS provides faster responses than cache MISSs but way slower than cache HITs. Therefore, cache performs worse during objects' first time reads. However, because of the markedly difference between COS and cache HIT average times, the total time required should be lower if interacting with the cache layer. This becomes therefore beneficial to this type of workloads as shown in charts 6.2 where it is presented the cost of executing all read objects operations by all threads. It is noticeable how cache's overall costs are lower than COS ones.




*Charts 6.2 – Custom benchmark total time execution with different thread setups: comparison between COS and cache results*

Another interesting thing that can be analyzed with this benchmarks is the moment where the cache becomes being beneficial. Cache's behavior in serving single requests during the entire execution is shown in the following charts.




*Charts 6.3 – Custom benchmark behavior with different thread setups: comparison between COS and cache results*

As shown with all three different benchmark's setups the introduction of the developed cache reduces the overall time required. If looking at chart 6.3 it is possible to appreciate the time needed to process GetObject operations at every instant during tests' execution. At the beginning, operations execution times are higher for caches scenarios but after a first moment they gradually decrease while the HIT rate increases. The COS behavior instead can be considered constant in serving requests. It is very interesting to notice how cache's performance shows a constant improvement during tasks computation.

The HIT rate function has been calculated as the overall percentage of HITs operations from the start to every instant in intervals of 50 seconds. If looking at COS and cache graphs in chart 6.3 it can be noticed that the performance of workloads implying the cache layer becomes better when the HIT rate reaches an approximate 77%, 65% and 60% respectively for the 10, 20 and 50 setups. This suggests that according to the data consuming rate the cache layer is capable of providing better performance sooner. This is a good result because of the data consuming speed characterizing the applications target such as those of Artificial Intelligence.

Finally, experiments with different RADOS configurations have been performed to measure how different back-end deployments affect workloads computation.

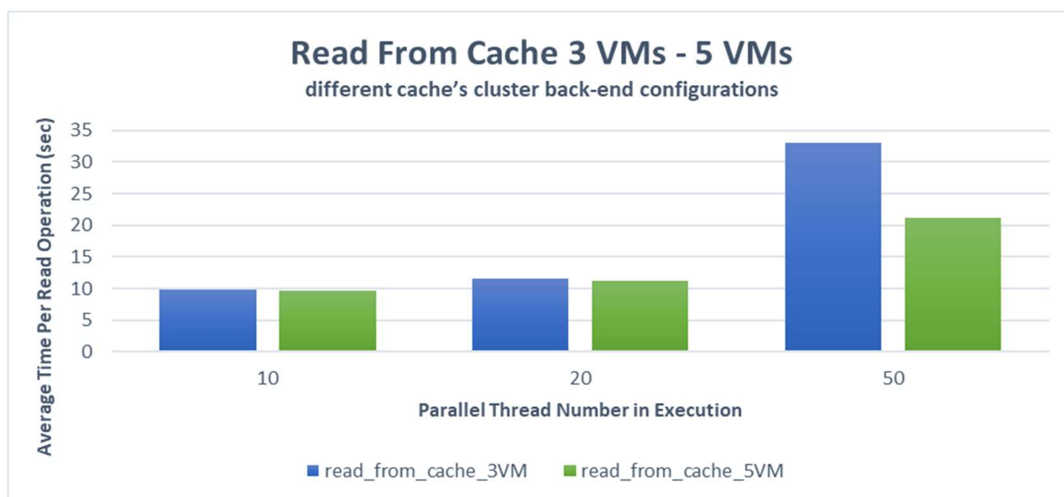



*Chart 6.4 – GetObject execution time cache 3 VMs – cache 5 VMs*

Cache's 3 VMs and 5 VMs configurations are characterized by different CPU capabilities with respectively 8 and 6 cores for each VM. Therefore, VMs of the former configuration are characterized by higher computational power. However, during benchmark's execution it has been noticed that VMs' CPU usage is very low. This makes the differences in computational power not affecting storage's performance which means that different results are due to the different nodes deployments of cache's back-end system.

As shown, deploying more nodes seems not affecting cache's setups of 10 and 20 threads while the 50 threads case shows a markedly difference in registered times. This result demonstrates an important characteristic of RADOS systems which maintain the stored data balanced within the deployed OSDs thanks to the translation from S3 objects to smaller Ceph objects. This particular data management is the reason for the performance improvement with the 5 VMs setup because data transfer operations are spread over more nodes and therefore each OSD workload is less heavy and S3 objects retrieval becomes faster.

This is an interesting example that highlights the power of efficient data management policies for optimizing devices usage and thus improving the capabilities offered by distributed storage systems such as Ceph.

## 6.3 CACHE PUTOBJECT PERFORMANCE WITH THE S3 CUSTOM BENCHMARK

Continuing on the same line of the discussion of the previous section, the improvement that are expected with the introduction of the developed cache layer in serving PutObject requests are the same as those of serving GetObject ones:

- shorter time intervals
- reduction in the number of those directly sent to the remote S3 service

As already mentioned in section 5.4 there are many considerations to be done when working with this specific operation. For instance, it is important to remember that the object versioning should be avoided when interested in limiting the number of requests sent to the COS. In addition, data alignment between the COS and the cache depends directly on write-back speed.

Write-back procedures are in charge of uploading to remote Cloud S3 storages the objects that have been written to the cache by client applications. A write-back implementation is considered slow when the write-back daemon does not immediately start uploading the object when written to the cache. However, when computing workloads characterized by many write operations over the same objects, a slow reacting write-back daemon will make it possible to considerably reduce the number of PutObject operations COS side.

If implementing more write-back threads, write-back policies would become more reacting and better data alignment would be provided. However, this trick may affect cache's overall performance as it would increase cache radosgw-daemon's workload. These considerations must be always taken into account when implementing write-back policies.

The experiments' results related to this type of workloads are presented as follows. Firstly, data in relation to PutObject performance will be presented and only after that, the discussion will proceed with all concerns about limiting the requests to be sent to the remote COS.

The following chart shows the results obtained with the execution of the custom benchmark of write operations with different thread configurations. These experiments have been meant to show how cache's behavior changes depending on different write workloads.

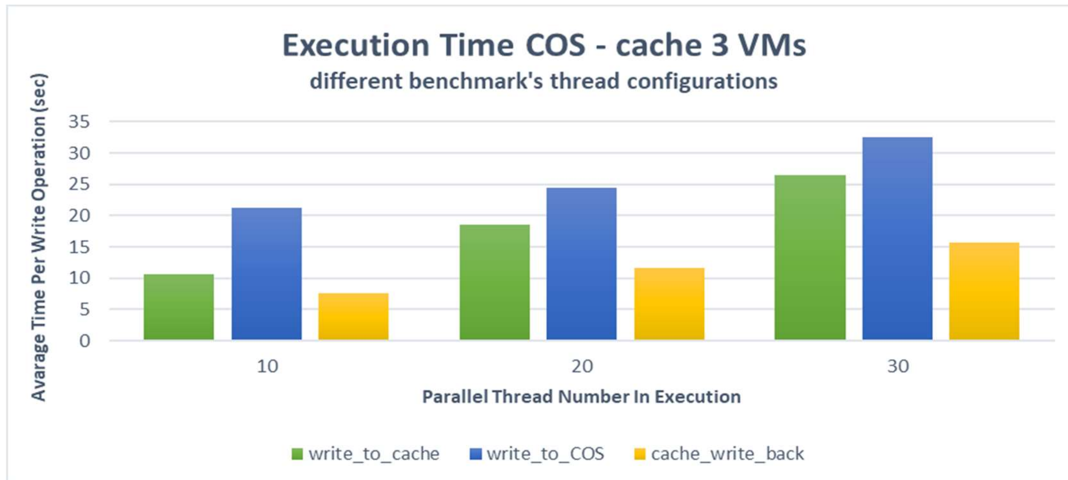



*Chart 6.5 – PutObject execution time COS – cache 3 VMs*

If looking at write_to_cache and write_to_COS columns it is possible to notice that with the introduction of the cache layer the time required to serve PutObject operations is lower. Response time reduction has been achieved in all three different benchmark's configurations and therefore it can be said that the cache is beneficial for write workloads. Thanks to its introduction, client applications are freed by the poor network data bandwidth limitation which is the most prominent bottleneck in this type of scenarios. As a consequence, the time required to transfer files in the form of Cloud objects over the S3 service is drastically reduced from the perspective of client applications.

If now looking at the chart in relation to the different benchmark's setup, it is interesting to notice how the gap between the values of cache and COS PutObject operations is similar for the 20 and 30 configurations. The 10 threads configuration instead is characterized by a considerable difference in such response times. The reason for this specific behavior is that while increasing storages' workloads the operations' times do not grow linearly with the number of simultaneous requests. Therefore, if over loading the cache layer it may

happen that its introduction stops being beneficial to the computation as the performance becomes equal than if communicating directly with the COS.

Following the same scheme of the discussion made in the previous section it is now interesting to see how cache's performance is affected by different RADOS back-end system's deployments.

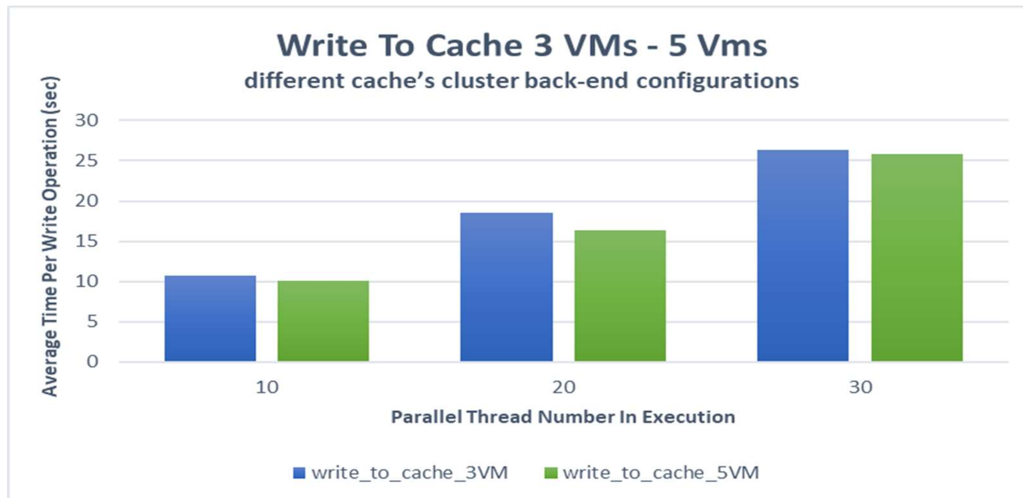



*Chart 6.6 – PutObject execution time cache 3 VMs – cache 5 VMs*

In accordance to what happens for GetObject workloads even during the execution of many PutObject operations the cache shows better response times when the number of back-end system's nodes is higher. In fact, the values presented in chart 6.6 show better cache's behavior when working with the 5 VMs RADOS configuration than with the 3 VMs one even if with less markedly gaps. This confirms the analysis made in the previous section and point out one more time how a good storage management policy is capable of providing good I/O performance when working with distributed storage systems.

Now that all performance related concepts have been clarified the discussion will continue by analyzing how it becomes possible to reduce the number of the requests that are sent remote S3 Cloud services during write workloads. In the next chart it is presented the incredible potential of the implemented S3 caching system in this direction. In fact, as it can be seen, the number of write-back operations that are effectively executed is definitely lower than those received by the cache.

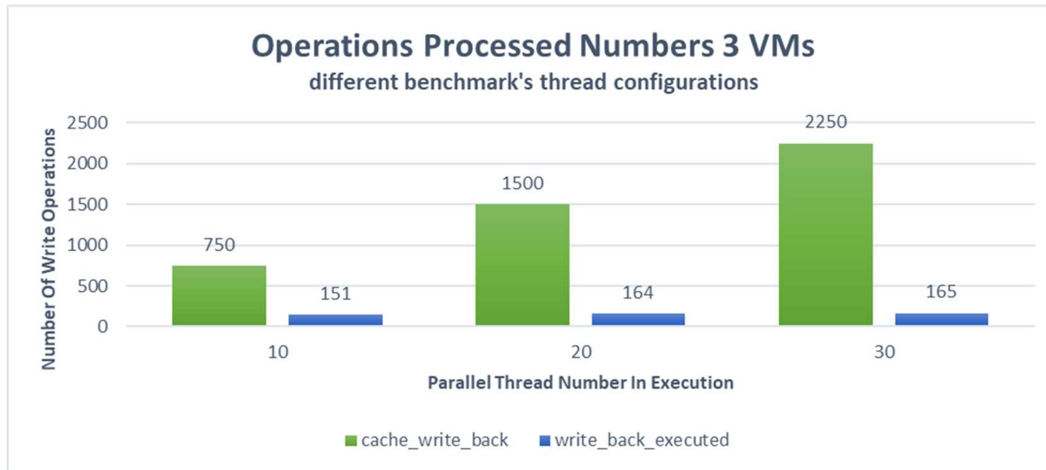



*Chart 6.7 – PutObject write back policy potential in reducing operations number*

As already explained, each thread of the custom benchmark executes 75 write operations of 50 different objects. As a consequence, the total number of executed operations changes depending on the different thread setup. However, the important result that must be pointed out is the number of the PutObject requests that are effectively sent to the remote COS. According to cache's implementation it is possible to reduce the operation number while aligning local objects to the remote support as shown in the image 5.5.

As already mentioned the write-back policy has been implemented as a single daemon thread which sequentially uploads cache's Cloud objects to the remote COS in order to align the stored data. If following the implementations details presented in section 5.4 it easy to understand how it becomes possible to reduce COS requests when renouncing to object versioning. However, the marked difference between the requests sent to the cache and the upload procedures effectively executed shown by the chart explains the important concepts that, the more the cache write-back waits, the more it is possible to reduce the requests number.

In fact, in this specific implementation the write back procedure is way slower than cache's speed in serving PutObject requests. The write-back queue is gradually emptied while write intensive workloads are in execution. However, as characterized by slow reacting times, when write workloads terminate the queue still has many elements and therefore many uploads still need to be executed. Because of this, the following PutObject requests that

will be sent to the remote COS will contain last object's versions of those stored into the cache and therefore many future uploads will not be needed and many requests COS side can be avoided.

This discussion aims to demonstrate the very big potential of S3 caches in limiting operations number which has important consequences for reducing S3 clients' expenses.

The specific workloads implied during the experiments are characterized by writing many times the same objects and it is hard to find similar business tasks. However, if implementing appropriate policies, these concepts for reducing service's costs remain valid.

## 6.4 CACHE GETOBJECT PERFORMANCE DURING TRAINING SESSIONS OF TENSORFLOW'S DNN RESNET OVER THE IMAGENET DATASET

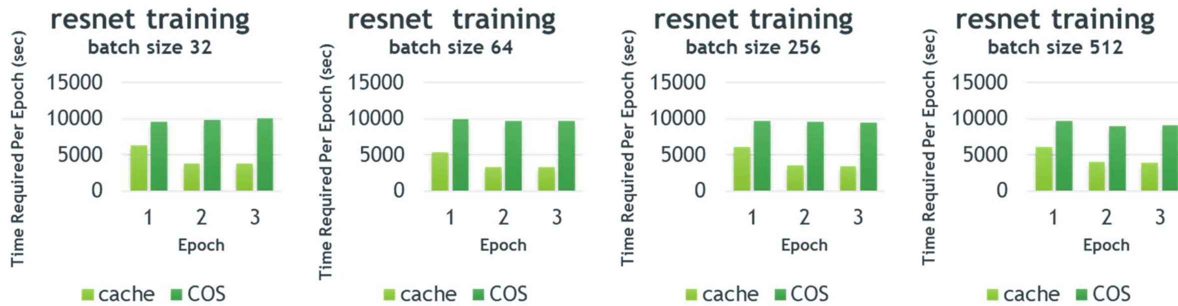
The results that are going to be presented in this section are related to workloads of deep learning which are characterized by intensive read operations. They are very important to test if the cache layer is capable of providing benefits also to real case scenarios.

As already mentioned in previous chapters, Cloud storage services as IaaS are increasingly being adopted in industrial scenarios because of their great capacity of offering low costs and hypothetically unlimited storage space. As a consequence, modern computational platforms such as for example TensorFlow for Artificial Intelligence applications or Apache Spark [40] for Big Data computing tasks started providing direct support to transparently access these types of Cloud services. For instance, just mentioned frameworks are capable of providing full support for I/O operations with S3 services.

The effect of different deployments over cache's performance have already been analyzed in previous sections for both read and write operations. Because of this, the experiments with TensorFlow's resnet have only focused on how different configurations affect training workloads' performance while interacting with the cache. Following results have been obtained by executing just mentioned operations with the 3 VMs setup of cache's back-end system. Training procedures have been executed utilizing 4 GPUs as previously presented in section 6.1.

The main difference of executed training sessions is represented by the training batch sizes as shown in the following charts. This will have important implications for the ReadObject requests sent to the cache.

The following charts show the time required to perform just mentioned training sessions during three training epochs.




*Charts 6.8 – Resnet training sessions time required depending on different batch sizes*

As it can be seen, training procedures require shorter time intervals when interacting with the cache rather than with the COS. Charts show time values near 9600 seconds for each epochs related to the COS. First epochs and following ones, related to the cache, register respectively values near 6000 and 3500 seconds. As expected, the first training epoch related to the cache requires much more time than the others while the trend of the COS remains constant. This is due to the fact that during the first epoch the objects containing the training dataset are not downloaded yet and therefore there are many MISS. From the second epoch instead, the behavior remains constant as all following GetObject requests will be cache's HITs. If computing more than three epochs the behavior would be the same and all those after the first one would show same performance as the second one.

Time needed to entirely perform just mentioned training workloads register approximately 13000 seconds average with the cache and 28000 average with the COS.

If looking carefully at the charts there is an important result worth of being discussed. In fact, seeing the first column of the cache lower than the COS one is not as expected as seeing bigger times to process the first epoch than the others. In the previous sections it has been discussed how MISS operations require longer time intervals to be processed than when interacting directly with a remote S3 service. Because of this, it would be expected to observe lower response times during the first epoch when interacting with the COS than when using the cache.

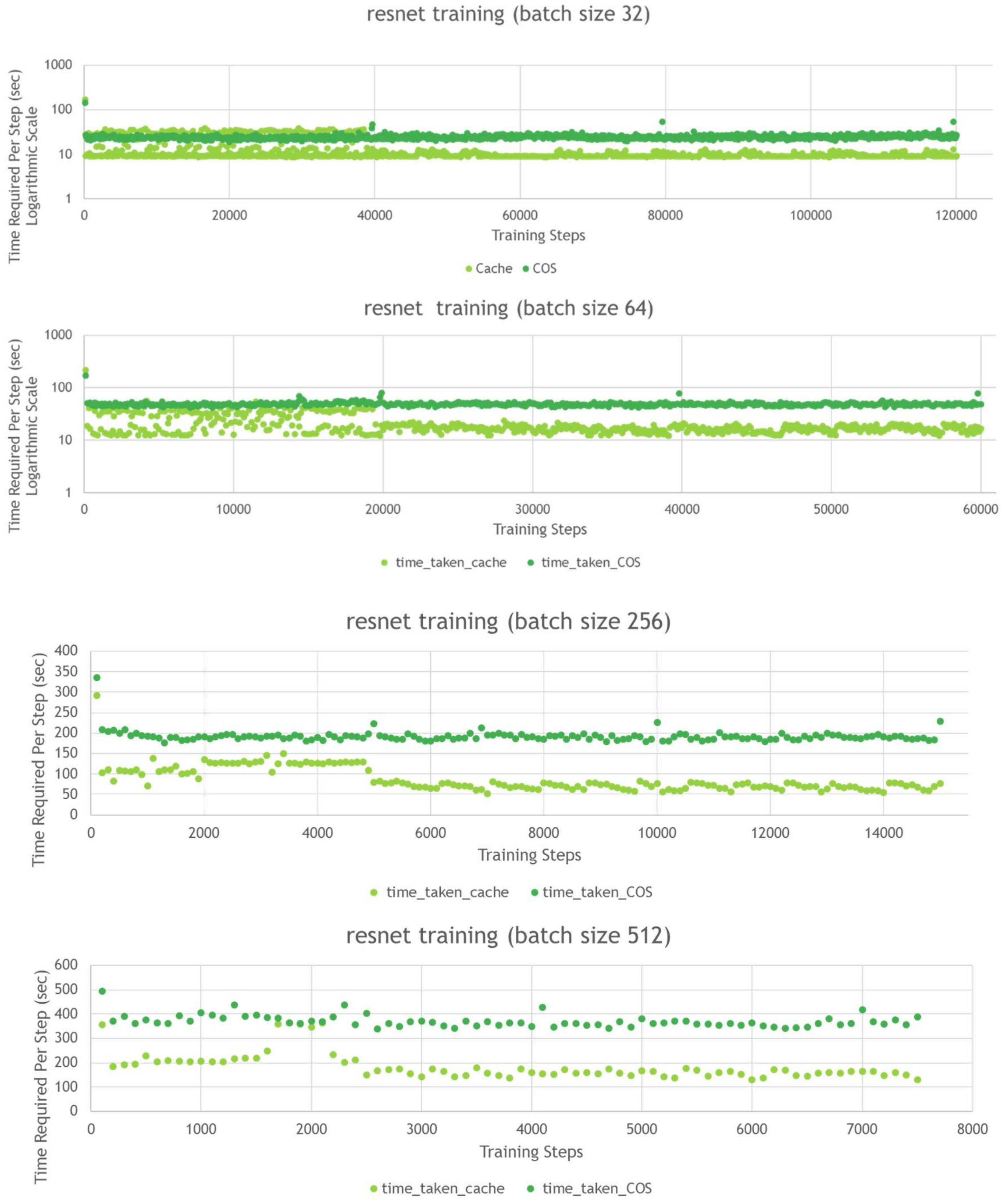
To understand this particular trend some additional details regarding resnet's workflow must be provided. In order to optimize S3 storage services usage, it is good practice to limit

the number of stored objects. For this reason, datasets of Artificial Intelligence tasks such as ImageNet are stored as huge blob objects each one containing many files. The TensorFlow's DNN resnet when reading dataset's files, do not request entire objects but only subsets of them by specifying byte-ranges within HTTP packets containing the GetObject requests. In this way download times become shorter and many operations can be executed simultaneously thus enhancing the parallelization as advised by best practice guidelines. In addition, many policies such as data prefetching have been performed by resnet application in order to optimize training's execution as much as possible.

With all these optimizations observing better performance with the cache during the first epoch is quite strange, also because of the worse performance of MISS operations than requests directly sent to the COS. However, if focusing on cache's implementation everything becomes clearer and reasonable.

The implemented cache layer is not capable of serving byte range requests, in the sense that when it receives such operations in MISS scenarios, the entire specified Cloud object will be downloaded and then only the specific byte-range will be returned to the applications client. Because of this, during the execution of the first training epoch with resnet it becomes possible to have HITs. In fact, it will be a MISS only the first byte-range request but after that, all future byte-requests referring to that object will be cache's HITs. Consequences of this cache's characteristic over the TensorFlow's Deep Neural Network are presented by the following charts.

Data presented in charts 6.9 show the time required to execute the training steps procedures. It is important to understand that their number is the same independently from the specific adopted device. As a consequence, it has been possible to compare COS and cache performance in the execution of every steps. Data related to total execution times required have already been shown in charts 6.8.




*Charts 6.9 – Resnet training sessions behavior with different batch sizes: comparison between COS and cache results*

During the data prefetching phase there are many objects that must be downloaded in order to process all needed information and this is the reason why the first 100 steps require more time than others in all four different batch configurations. Once all needed objects for the

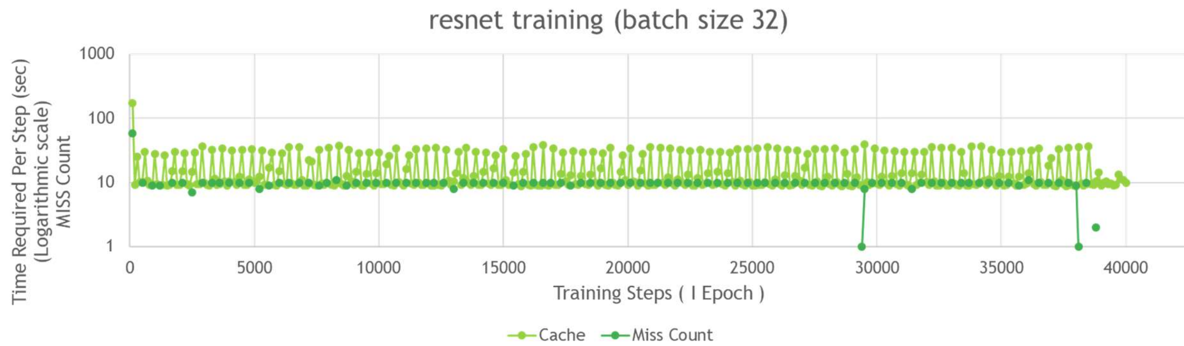
initial steps have been downloaded, the program keeps downloading those required by future steps without stopping. Because of this, if all byte ranges required refers to already downloaded objects there are no waiting times.

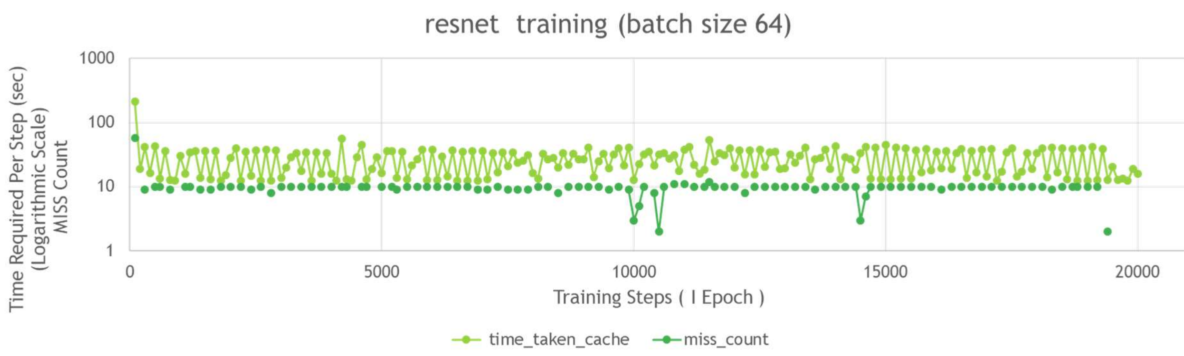
There is an interesting difference between first steps time intervals registered with small and big batch sizes. In fact, for the 32 and 64 configurations the first value of the cache is higher than the one of the COS while for the 256 and 512 ones the trend is the opposite. This is due to the fact that when working with small batch sizes, downloading the entire objects represents a drawback for initial steps as they are capable of computing faster each step. In contrary, when working with big batch sizes such as 256 and 512 it does not represent a limitation. Downloading many objects to compute each step in fact, makes files needed in future steps already available. Future steps do not have to wait before being in condition of working as bigger batch size requires more time to perform steps' computation and therefore more objects can be downloaded. This concept is well explained in charts 6.10.

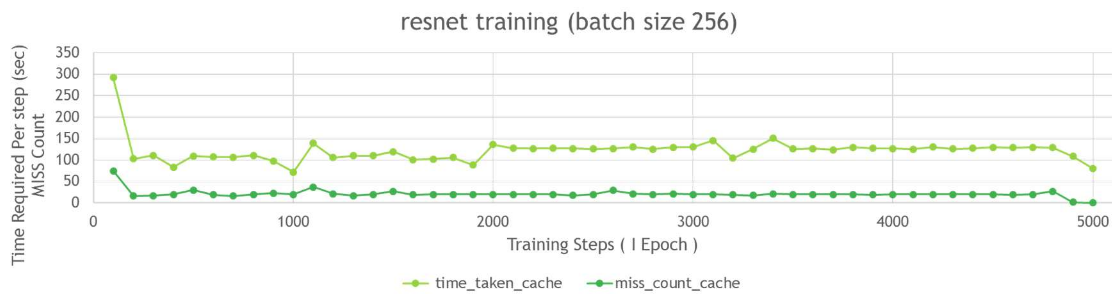
In addition, it is noticeable how all registered times of monitored training steps with the cache are lower than those related to the remote COS. There are only few cases where cache and COS show similar values which are those related to the first training epoch with a batch size of 32. This is due to the fact that when implying small batch sizes, time required to compute each step are lower. As a consequence, following steps are ready to compute but needed objects are not available yet. Therefore, waiting times during the first epoch are longer and may affect steps performance.

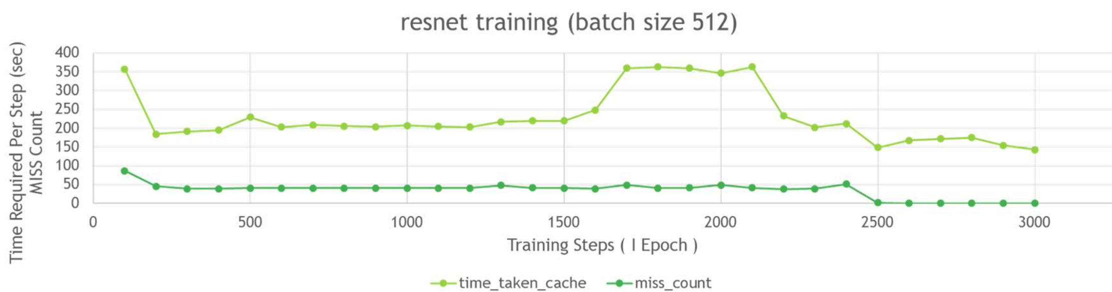
Finally, it is important to highlight how charts show a constant behavior for the COS during all different epochs while the cache after the worse performance of the first epoch maintain a constant and lower time to process all next steps. If focusing on cache's data, it is interesting to see how at a certain point time required values drastically drop and then remain constant until the end of training procedures.

The following charts provide specific information about the first training epoch in order to give more details of the relation between training step times and cache's MISS number.




*Charts 6.10 – Resnet first training epoch behavior with different batch sizes: relation to MISS number*

These graphs are important to highlight the relation between cache MISS and steps time depending on the different batch size. If looking at the charts of 32 and 64 it is possible to

notice that there are training steps where there are no MISSs at all. However bigger time intervals are registered in presence of MISSs during previous steps. As already said, this causes steps to wait more before being able to start executing for smaller batch sizes. However, small batch sizes make it possible to have training steps with HIT rates of 100% even during the first Epoch. This important result can also be appreciated with charts 6.11 which show HIT rate values during the training steps of the first epoch. When HIT rates do not reach the maximum value following steps are characterized by consequently bigger times as they must wait for the objects to be downloaded. HIT rates with no 100% are in fact characterized by some cache MISSs.

*Charts 6.11 – Resnet first training epoch HIT rate with different batch sizes*

While, in charts 6.10 there is a drastic decrease in time values in proximity to the end of the first epoch computation, charts 6.11 are characterized by noticeable increase. This is due to the fact that from the second epoch, the entire dataset has already been downloaded and therefore all future GetObject requests will correspond to cache's HITs.

# CONCLUSIONS

This Thesis work provided a comprehensive analysis of modern Distributed Systems and new prominent Cloud environments (chapter 1). An in-depth analysis of storage systems and paradigms have been made. In this way all concepts related to the scenarios of interests such as those of the Multi Cloud have been clarified (chapter 2).

Some prominent technologies at the state of the art have been presented. In this way it has been possible to point out the importance of distributed storage systems and how they can be adopted to support heterogeneous Cloud environments. Moreover, S3 storage services' key concepts and guidelines have been discussed as they have been the central topic of this work (chapter 3). The previous analysis continued by entering in the details of the Ceph distributed storage system. Its main components have been introduced to show what services it is capable of providing (chapter 4).

The developed persistent S3 cache system have been presented along with the design and implementation choices that have been made. As the Ceph storage system already offers many S3 related services it has been adopted in order to implement the S3 caching service. This has been very important especially for dealing with the S3 authentication service (chapter 5). Finally, its performance in serving high intensive workloads has been measured and the results have been discussed (chapter 6).

Thanks to the custom benchmarks used in the evaluation it can be seen that the impact of the cache is, as expected, higher for cache miss cases. An object read (GetObject) is in average 35% slower than a regular read from remote storages. While in case of cache hit an object read is 57% faster than a regular read from remote storages. Overall we have observed that, with the experimental setup used for this work, the caching system proposed starts being beneficial for applications with a hit rate higher than 60%.

Similarly, it has been observed that an object write (PutObject) is in average 30% faster if performed with the cache system, thanks to the implementation of a write-back policy. This causes a data misalignment with the remote storage service because write-back procedures

become slower as interacting with Cloud services. However, if properly exploited, this delay allows to drastically reduce the number of the S3 operations sent to remote storages of more than 90%. However, client applications must renounce to object versioning.

Thanks to the experiments performed using TensorFlow's resnet it has been possible to demonstrate cache's benefits in real case scenarios. Training steps of deep learning tasks have been measured. Workloads using the S3 cache layer are in average 40% faster in computing first training epochs than when interacting with remote Cloud services. In addition, starting from the second epoch, performance increases of 64%.

Ceph is a good choice for taking advantage of already implemented services such as user authentication of S3 requests. It is also a perfect solution to satisfy Multi Cloud environments' needs. In fact, it allows deployments of distributed storage systems over heterogeneous devices.

Finally, when executing benchmark applications, it has been noticed that with more intensive workloads cache's response times tend to increase. However, even in worst cases, registered values have shown better performance than those related to COS communications.

Overall, with this Thesis work it has been demonstrated that a caching system may result very effective in Hybrid- and Multi- Cloud environments. In addition, preliminary results have shown the incredible potential of implementing a good write-back policy.

To conclude, cache's capabilities will be extended in future works. Data-prefetching and eviction policies will be implemented by using already available S3 services offered by the Ceph's radosgw module. Solutions to store objects as fixed-size blocks within the cache's memory will be analyzed along with the opportunity of specifying byte-ranges of S3 objects during read operations. The possibility of interacting directly to the RADOS back-end system at low level will be explored and some prototypes will be implemented. Therefore, related effects on cache's performance will be analyzed. Additional caching services at the bucket level will be studied. Finally, other RGW modules will be utilized to

enhance cache’s management capabilities. For instance, the “Cloud Sync Module” should be studied in relation to cache-COS data alignment concerns. Also, the “Security Token Service” will be used in order to improve cache’s user authentication policies and thus enhancing cache’s security capabilities.

# REFERENCES


[1] "Distributed Systems Concepts and Design" – George Coulouris Jean Dollimor Tim Kindberg Gordon Blair

[2] "Cloud Computing: A Perspective Study" – Lizhe Wang Gregor von Laszewski Andrew J. Younge HE Xi

[3] "Cloud Computing: An Overview" – ACM Queue vol. 7 no. 5

[4] "What's Inside the Cloud? An Architectural Map of the Cloud Landscape" – Alexander Lenk Markus Klems Jens Nimis Stefan Tai Thomas Sandholm

[5] "The NIST Definition of Cloud Computing" – Peter Mell Timothy Grance

[6] "Mapping Cross-Cloud Systems: Challenges and Opportunities" – Yehia Elkhatib

[7] "POSIX Programmer's Guide Writing Portable UNIX Programs with the POSIX.1 Standard" – Donald A. Lewine

[8] "Information Storage and Management" – John Wiley and Sons Inc.

[9] "Hierarchical File Systems Are Dead" – Margo Seltzer Nicholas Murphy

[10] "The Hadoop Distributed File System" – Konstantin Shvachko Hairong Kuang Sanjay Radia Robert Chansler

[11] "The Google File System" – Sanjay Ghemawat Howard Gobioff Shun-Tak Leung

[12] "The Design and Implementation of the Database File System" – Nick Murphy Mark Tonkelowitz Mike Vernal

[13] "GPUfs: Integrating a File System with GPUs" – Mark Silberstein Bryan Ford Idit Keidar

[14] "Ceph: A Scalable High-Performance Distributed File System" – Sage A. Weil Scott A. Brandt Ethan L. Miller Darrell D. E. Long

[15] “Lustre: A Scalable High-Performance File System” – Cluster File Systems Inc.

[16] “To FUSE or Not to FUSE: Performance of User-Space File Systems” – Bharath Kumar Reddy Vangoor Vasily Tarasov Erez Zadok

[17] “S3fs” – Randy Rizun Dan Moore Adrian Petrescu Ben LeMasurier Takeshi Nakatani Andrew Gaul (https://github.com/s3fs-fuse/s3fs-fuse)

[18] “Cloud Distributed File Systems: a Benchmark of HDFS Ceph GlusterFS and XtremeFS” – Luca Acquaviva Paolo Bellavista Antonio Corradi Luca Foschini Leo Gioia Pasquale Carlo Maiorano Picone

[19] “Overview of Amazon Web Services” – Amazon Web Services Inc. (https://d1.awsstatic.com/whitepapers/aws-overview.pdf)

[20] “Amazon S3 Masterclass” – AWS Online Tech Talks (https://www.youtube.com/watch?v=VC0k-noNwOU)

[21] “Amazon Simple Storage Service Getting Started Guide” – Amazon Web Services Inc. (https://docs.aws.amazon.com/AmazonS3/latest/gsg/s3-gsg.pdf)

[22] “Amazon Simple Storage Service Developer Guide” – Amazon Web Services Inc. (https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-dg.pdf)

[23] “Amazon Simple Storage Service API Reference” – Amazon Web Services Inc. (https://docs.aws.amazon.com/AmazonS3/latest/API/s3-api.pdf)

[24] “AWS SDK for C++ Developer Guide” – Amazon Web Services Inc. (https://docs.aws.amazon.com/sdk-for-cpp/v1/developer-guide/aws-sdk-cpp-dg.pdf#welcome)

[25] "Best Practices Design Patterns: Optimizing Amazon S3 Performance" – Mai-Lan Tomsen Bukovec Andy Warfield Tim Harris – Amazon Web Services Inc. (https://d1.awsstatic.com/whitepapers/AmazonS3BestPractices.pdf)

[26] "REST API Design Rulebook" – O'reilly – Mark Massé

[27] "Amazon S3 pricing" (https://aws.amazon.com/s3/pricing/)

[28] "IBM Cloud Object Storage pricing" (https://www.ibm.com/Cloud/object-storage/pricing)

[29] "Ceph Documentation" – Ceph (https://docs.ceph.com/docs/master/)

[30] "Ceph Tech Talk – Intro to Ceph" - Ceph (https://www.youtube.com/watch?v=PmLPbrf-x9g)

[31] "Docker" – Docker (https://www.docker.com/)

[32] "Kubernetes" – Kubernetes (https://kubernetes.io/)

[32] "OpenShift" – OpenShift (https://www.openshift.com/)

[33] "Swift" – Open Stack (https://wiki.openstack.org/wiki/Swift)

[34] "Civetweb" – (https://github.com/civetweb/civetweb)

[35] "Boost" – (https://www.boost.org/)

[36] "Ceph Tech Talk – RGW" – Ceph (https://www.youtube.com/watch?v=PmLPbrf-x9g&t=3470s)

[37] “FUSE documentation” – FUSE
(https://libfuse.github.io/doxygen/structfuse__operations.html)

[38] “rgw” – Ceph
(https://github.com/ceph/ceph/tree/master/src/rgw)

[39] “put-bucket-lifecycle” – AWS CLI
(https://docs.aws.amazon.com/cli/latest/reference/s3api/put-bucket-lifecycle.html)

[40] “Apache Spark” – Apache
(https://spark.apache.org/)

[41] “resnet” – TensorFlow
(https://github.com/tensorflow/models/tree/master/official/vision/image_classification)

[42] “KVM” – linux-kvm
(https://www.linux-kvm.org/page/Main_Page)

# IMAGES

Image 1.1 Heterogeneous devices interacting with the Internet

Image 1.2 Representation of the Cloud Services and Cloud Infrastructures Stack

Image 1.3 Hybrid Cloud scenario where a Private Cloud interacts with a Public one to answer new companies' needs of privacy and elastic scaling

Image 2.1 Example of management of physical storages as virtual units [8]

Image 2.2 General file system hierarchy – each triangle represents a different file system which has been made available through a call to the operating system mount operation

Image 2.3 Examples of NAS and SAN in the internet scenario (https://searchstorage.techtarget.com/definition/storage-area-network-SAN)

Image 2.4 Unified information storage architecture [8]

Image 2.5 The storage paradigms of file block and object memories (https://ubuntu.com/blog/what-are-the-different-types-of-storage-block-object-and-file)

Image 3.1 An architectural view of Cloud Distributed File Systems [18]

Image 3.2 FUSE architecture [16]

Image 3.3 The FUSE s3fs study case

Image 4.1 The architecture of Ceph as a unified storage system [30]

Image 4.2 RADOS software components [30]

Image 4.3 CRASH calculated placement procedure [30]

Image 4.4 Representation of multimedia files data distributed over Ceph's objects pools placement groups and OSDs [30]

Image 4.5 Program-file interaction with CephFS and RADOS [30]

Image 4.6 View of the CephFS module within Ceph's software hierarchy [29]

Image 4.7 The RGW architecture [36]

Image 4.8 Interaction with the RGW as a Cloud Object Service during a PUTOBJECT request [30]

Image 5.1 Radosgw-daemon workflow

Image 5.2 Model changes with the introduction of the S3 cache layer

Image 5.3 Processing flow of RGW_OP_GET_OBJ operations during HIT and MISS scenarios

Image 5.4 Processing flow of RGW_OP_PUT_OBJ operations with the implemented write-back policy

Image 5.5 S3 objects write-back

Image 5.6 Processing flow of RGW_OP_LIST_BUCKET operation with the introduction of the cache

# TABLES AND CHARTS


Table 2.1 Storage paradigms' concepts summary

Table 3.1 S3 AWS and IBM COS pricing policies

Table 4.1 FUSE cache back-end – libcephfs POSIX mapping

Table 5.1 S3 RGW operations

Chart 6.1 GetObject execution time COS – cache 3 VMs

Charts 6.2 Custom benchmark total time execution with different thread setups: comparison between COS and cache results

Charts 6.3 Custom benchmark behavior with different thread setups: comparison between COS and cache results

Chart 6.4 GetObject execution time cache 3 VMs – cache 5 VMs

Chart 6.5 PutObject execution time COS – cache 3 VMs

Chart 6.6 PutObject execution time cache 3 VMs – cache 5 VMs

Chart 6.7 PutObject write back policy potential in reducing operations number

Charts 6.8 Resnet training sessions time required depending on different batch sizes

Charts 6.9 Resnet training sessions behavior with different batch sizes: comparison between COS and cache results

Charts 6.10 Resnet first training epoch behavior with different batch sizes: correlation to MISS number

Charts 6.11 Resnet first training epoch HIT rate with different batch sizes

# ACKNOWLEDGMENTS

With this work I conclude my Master Degree in Computer Engineering at the University of Bologna. This is a very important moment of my life. It is the final result of all the efforts and sacrifices I made during last six years.

I want to dedicate it to my family and my friends. Their support during good and bad moments have been fundamental. I don't know if I would have been here writing these lines without them.

During these years I had the honor of meeting incredible people. They all made my life worth of living and with these few more lines I want to express my deepest gratitude for sharing special moments with me.

First of all, I want to thank my parents Ivano and Monica, my grandparents Enzo, Marta, Mauro and Nilde, my cousin Filippo and my uncle Emilio. They have always been with me and I could never be more grateful for having them in my life.

I want to thank my friends of Reggio Emilia for all the time we had together and the time in front of us. You are part of my family as well.

I want to thank a special person I met in Madrid and I have in Dublin, I love you.

I want to thank my friends and flat mates in Bologna. I want to thank my Erasmus friends and flat mates in Madrid. I want to thank my IBM colleagues and friends in Dublin. I want to thank my University colleagues in Bologna. I want to thank the friends I have all around the world. I want to thank my Professors who gave me the opportunity of going abroad.

I want to thank life.