Dependability in cloud storage

Pierre Obame Meye

ANNÉE 2016

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MATISSE

présentée par

Pierre OBAME MEYE

préparée à l'unité de recherche UMR 6074 IRISA
Équipes d'accueil:
Orange Labs OLPS/ASE/DAPI/CSE - IRISA/CIDRE
Contrat CIFRE nº 2011-1550

La sûreté de fonctionnement dans le nuage de stockage

Dependability in cloud storage

Thèse soutenue à Rennes le 01 décembre 2016

devant le jury composé de :

EZHILCHELVAN Paul
Reader Université de Newcastle / rapporteur

Marc-Olivier KILLIJIAN
Directeur de recherche CNRS LAAS / rapporteur

Jean-Louis PAZAT
Professeur INSA / examinateur

Achour MOSTEFAOUI
Professeur Université de Nantes / examinateur

POTOP-BUTUCARU Maria
Professeur Université UPMC / examinatrice

Philippe RAIPIN
Ingénieur de Recherche Orange Labs / examinateur

Frédéric TRONEL
Maître de Conférence CentraleSupelec / examinateur

Emmanuelle ANCEAUME
Chargée de recherche CNRS / directrice de thèse

Résumé

Avec l'évolution des technologies et la prolifération d'objects connectés, nous assistons à une explosion de la quantité de données créées et consommées, et cette quantité de données ne cesse de croître. Un étude menée par Cisco [111] estime à plus de 50 milliards le nombre d'objets connectés en 2020. Une autre étude, menée l'International Data Corporation (IDC) [52], prédit que jusqu'à l'année 2020, le volume global des données dans le monde augmentera d'un facteur de 300, passant de 130 exabytes à 40000 exabytes avec une croissance doublant tous les deux ans. Cela pose des challenges aux fournisseurs de service de stockage qui doivent trouver des moyens de faire face à cette croissance de données de manière scalable, efficace, tout en optimisant les coûts.

Tout au long de cette thèse, nous nous sommes intéressés aux systèmes de stockage en environnements de type *cloud*. Un environnement de type cloud a la capacité de fournir à la demande des ressources (calcul, réseaux, stockage, etc) à des utilisateurs (Voir Chapitre 1). Dans notre cas, nous nous sommes focalisés sur les environnements cloud fournissant des ressources de stockage. Ce type d'environnements souvent appelé *nuage de stockage* est de plus en plus utilisé comme solution de gestion de données. L'IDC prédit notamment que d'ici l'année 2020, environ 40% des données dans le monde seront stockés ou traités dans un système de type cloud [52] .

Généralement, un nuage de stockage stocke les données dans des centres de données, communément appelés datacenters. Lorsqu'un système cloud est construit sur une architecture centralisée basée sur d'énormes datacenters, cela peut entraîner de fortes latences d'accès aux données car les utilisateurs ne sont pas nécessairement situés près du datacenter. Réduire les latences d'accès aux données est l'un des points que nous avons adressé dans cette thèse en considérant que les données devraient être situées près de leur utilisateurs.

A l'intérieur d'un datacenter, des moyens sont généralement mis en oeuvre pour assurer un certain niveau de fiabilité afin d'éviter ou limiter des incidents pouvant entraîner des pertes de données ou des indisponibilités de service. Toutefois un datacenter représente en soi un point unique de défaillance. Si le datacenter devient innacessible pour une quelconque raison (p.ex. coupure de courant, catastrophe naturelle, surchage, etc) toutes les données et les services hébergés par ce datacenter deviennent indisponibles.

Contrairement aux architectures centralisées, il est plus complexe de construire des architectures distribuées composées de plusieurs noeuds (c-à-d datacenter, serveurs, etc) géographiquement dispersés, ainsi que des mécanismes permettant au système global de continuer à fonctionner en tolérant les pannes ou les pertes de certains noeuds du système. Cependant il devient crucial de considérer ce genre de solution. Dans le contexte de nuage de stockage basé sur des architectures distribuées, nous nous sommes intéressés aux problématiques de sûreté de fonctionnement qui fait référence à un ensemble de propriétés permettant de définir ou de juger le bon fonctionnement d'un système (Voir Chapitre 3). Dans cet ensemble, nous nous sommes concentrés sur les propriétés telles que la disponibilité et la fiabilité des systèmes.

Un opérateur de télécommunication tel que Orange fournissant des services de stockage dans le nuage pourrait trouver dans sa propre infrastructure des atouts permettant de faire face aux différentes problématiques évoquées précédemment. En effet, en tant qu'opérateur de télécommunication, Orange a le contrôle et la maîtrise de son infrastructure réseau et pourrait tirer avantage de différents endroits dans son infrastructure pour installer des noeuds de stockage. Ces noeuds de stockage pourraient être des datacenters situés loin des utilisateurs, des Points de Présence situés près des utilisateurs en bordure du réseau coeur, mais aussi les box situées dans les maisons des utilisateurs du réseau d'Orange. La capacité de maîtrise et de contrôle du trafic réseau et sa

qualité de service, aussi bien que la possibilité d'exploiter différents endroits de stockage proches et loin des utilisateurs sont des atouts qu'il est difficile de trouver chez un fournisseur n'étant pas un opérateur de télécommunication.

Les contributions dans cette thèse ont été articulées autour des moyens de fournir un système de cloud distribué dans le contexte d'un opérateur de télécommunication tel que Orange. Ces moyens visent à améliorer la scalabilité, les performances d'accès et la fiabilité du système. Nous nous sommes particulièrement intéressés à l'étude des architectures distribuées et des propriétés de sûreté de fonctionnment telles que la disponibilité et la cohérence des données. Nous avons aussi considéré la sécurité des données dans le contexte de systèmes de stockage appliquant une déduplication des données afin de réduire les coût de stockage et de bande passante réseau.

Adresser ces différents points dans le contexte d'un opérateur de télécommunication est très peu couvert dans la littérature. Ce qui rend les travaux prometteurs mais aussi challengeant étant donné le nombre de problématiques et de domaines impliqués. Cependant, nous avons étudié ces domaines et proposé des solutions basées sur des idées innovantes qui ont été présentées et publiées dans des conférences internationales et dans certains cas, ces solutions ont fait l'objet de dépôts de brevets.

Contributions

- **Mistore:** Les Fournisseurs d'Accès à Internet (FAI) qui fournissent aussi des services de stockage dans le nuage, se basent généralement sur des architectures basées sur des énormes datacenters. Ce genre d'architectures parfois centralisées, possèdent certains inconvénients en termes de fiabilité ou de latence d'accès aux données du fait que ces datacenters représentent des points uniques de défaillance et qu'ils ne sont pas nécessairement situés près des utilisateurs. Mistore est un système de stockage distribué que nous avons conçu pour assurer la disponibilité des données, leur durabilité, ainsi que de faibles latences d'accès aux données en exploitant les atouts d'une infrastructure Digital Subscriber Line (xDSL) d'un FAI. Dans Mistore, nous addressons aussi les problèmes de cohérence de données en fournissant plusieurs critères de cohérence des données ainsi qu'un système de versionning. Cette contribution est décrite dans le Chapitre 4.

- **Placement et maintenance des données dans Mistore:** Cette contribution améliore le placement et la maintenance des données dans notre précédente contribution sur l'architecture de Mistore. Dans la précédente contribution, le placement des données dans chaque région est basée sur une méthode de type catalogue. Toutes les informations sur le placement des données dans une région à savoir quelle donnée est stockée sur quelle box, étaient maintenues par le point de présence de la région. Ce type d'approche est simple à concevoir mais elle a le défaut de perdre en scalabilité lorsque le volume de données et donc de métadonnées à gérer devient très grand. Dans cette contribution, nous avons conçu une méthode flexible et plus scalable dans laquelle le volume des métadonnées dépend du nombre de noeuds de stockage dans la région et non sur le volume de données potentiellement énorme des données. Cette contribution est décrite dans le Chapitre 5.

- **La déduplication en deux phases:** Le volume de données dans le monde augmente à un rythme impressionnant. Toutefois, une étude de l'IDC estime que 75% des données dans monde est une copie [51]. Il est important de garder plusieurs copies d'une données afin d'assurer sa disponibilité et sa durabilité mais dans certaines situations, le niveau de redondance peut être exagéré. En ne maintenant qu'une copie des données redondantes, la déduplication est considérée comme une des solutions les plus prometteuses pour réduire les coûts de stockage et améliorer l'expérience utilisateur en économisant la bande passante réseau et réduisant les temps de sauvegarde des données. Cependant, cette technologie souffre de certains problèmes de sécurité qui ont été reportés dans la littérature. Dans cette contribution, nous nous sommes focalisés sur les attaques orientées clients et nous avons proposé et conçu une méthode que nous avons appelé la déduplication en deux phases qui permet de

mettre en place un système de stockage effectuant la déduplication à la source tout en étant sécurisée contre les attaques provenant de clients malicieux. Notre solution reste efficace en termes d'économie de bande passante réseaux et d'espace de stockage et a fait l'objet d'un dépôt de brevet. Cette contribution est décrite dans le Chapitre 6.

- **Méthode d'externalisation des données d'entreprises:** L'adoption des technologies cloud pour externaliser les données d'entreprise est de plus en plus courant. Cependant de nombreuses entreprises y sont reluctantes dû au fait que les atouts du cloud viennent aussi avec un certain risque car un contrôle total sur les données est attribué au fournisseur du service de cloud storage. Dans cette contribution, nous avons conçu une solution permettant aux entreprises d'externaliser leurs données dans le cloud tout en gardant un certain contrôle. Notre solution se base sur une passerelle de stockage située sur le site de l'entreprise et a pour but d'implémenter la stratégie de contrôle sur les données stockées dans le nuage d'un fournisseur de cloud extérieur à l'entreprise. La passerelle de stockage améliore aussi les performances d'accès aux données, optimise l'utilisation de l'espace de stockage et de la bande passante réseau et gère la niveau de confidentialité des données externalisées. Pour cela un certain nombre de mécanismes sont mis en place sur le site à savoir un système de gestion des métadonnées, une fonctionnalité de cache des données, un système flexible de placement des données avec pour but de supporter des nuages de stockage de différents fournisseurs. Notre solution met aussi en place notre système de déduplication en deux phases afin d'améliorer l'usage de l'espace de stockage et de la bande passante réseau. Cette solution est décrite dans le Chapitre 7.

- **ODISEA:** Durant cette thèse, nous avons aussi pris part à un projet de recherche dénommé ODISEA pour Open Distributed Networked Storage. ODISEA était un projet collaboratif français impliquant plusieurs partenaires industriels et académiques à savoir Orange Labs, l'Université Pierre et Marie Curie, Technicolor, Ubistorage, eNovance, et l'Institut Télécom. Ce projet, dans lequel Orange était leader, a été selectionné par les Fonds Unique Interministeriel (FUI). Les FUI sont un programme français supportant la recherche appliquée en finançant un certain nombre de projets innovants impliquant des acteurs industriels et académiques. Ce projet avait pour but de fournir une plateforme de stockage distribuée exploitant au mieux les ressources au sein du réseau interconnectant les noeuds de stockage et en y intégrant de l'intelligence pour gérer le placement, l'accès et le stockage des données. Dans ce projet nous avons pris part aux processus de conception de la solution et au développement du système de gestion des métadonnées dans le prototype de la solution. Le prototype d'ODISEA a été présenté au Salon de la Recherche organisé à Paris en 2013 par Orange.

Résultats

Articles scientifiques

- Pierre Meye, Philippe Raïpin, Frédéric Tronel, and Emmanuelle Anceaume, "A secure two-phase data deduplication scheme", Proceedings of the 6th International Symposium on Cyberspace Safety and Security (CSS). Paris (FRANCE) 2014.

- Pierre Meye, Philippe Raïpin, Frédéric Tronel, and Emmanuelle Anceaume, "Mistore: A distributed storage system leveraging the DSL infrastructure of an ISP", Proceedings of the International Conference on High Performance Computing & Simulation (HPCS). Bologna (ITALIA) 2014.

- Pierre Meye, Philippe Raïpin, Frédéric Tronel, and Emmanuelle Anceaume, "Toward a distributed storage system leveraging the DSL infrastructure of an ISP", Proceedings of the 11th IEEE Consumer Communications and Networking Conference (CCNC). Las Vegas (USA) 2014.

Brevets

- Pierre Meye, Philippe Raïpin, Method for storing data in a computer system performing data deduplication. WO2014154973. 2014
- Philippe Raïpin, Pierre Meye, Method of processing data in a distributed storage system. WO2014096607. 2014

Livre blanc

- Sébastien Canard, Pierre Meye, Philippe Raïpin, Jacques Traoré, and Mohammed Tuhin. "Cloud storage and confidentiality", Research paper (Orange Labs). 2014

Présentations

- Pierre Meye, "A secure two-phase data deduplication scheme". WOS 4: Workshop on Storage and Processing of Big Data. Rennes (FRANCE) 2014.

Plan de la thèse

Ce manuscrit de thèse est organisé comme suit. Les Chapitres de 1 à 3 introduisent les différents concepts nécessaires à la compréhension des problématiques traitées et des contributions de cette thèse. Plus spécifiquement, dans le Chapitre 1, nous évoquons les notions de base du paradigme du cloud computing, contexte dans lequel cette thèse s'inscrit. Dans le Chapitre 2, nous présentons des technologies de stockage ainsi que leur spécificité. Nous présentons les bases de la sûreté de fonctionnement dans le Chapitre 3 avec un focus sur les propriétés liées à nos recherches et contributions. Dans le Chapitre 4, nous présentons Mistore le système de stockage distribué concu durant cette thèse. Dans le Chapitre 5, nous décrivons un système de placement et de maintenance de données concu pour gérer efficacement les données dans Mistore. Nous présentons dans le Chapitre 6 la déduplication en deux phases. Dans le Chapitre 7, nous décrivons une méthode permettant d'externaliser des données dans le cloud tout en préservant un contrôle sur les données. Enfin, nous clôturons, cette thèse par le Chapitre 8 dans lequel nous tirons une conclusion et évoquant des perspectives pour de futurs travaux de recherche.

Abstract

With the evolution technologies and proliferation of connected devices, we see an explosion in the amount of data created and consumed, and this amount of data is steadily growing. For instance, a study from Cisco [111] projects that their will be more than 50 billions connected devices by 2020 connecting every facet of human activity. Another study by the International Data Corporation (IDC) [52] predicts that until 2020, the global data volume will grow by a factor of 300, from 130 exabytes to 40,000 exabytes, representing a double growth every two years. All this will lead to tremendous quantities of data to manage, putting pressure on storage providers to find ways to handle such amount of data efficiently in term of scalability and in a cost-effectively manner.

In this PhD thesis, we have been interested in storages in the context of *cloud* environments. The concept of the cloud envisions the ability of a system to provide scalable on-demand resources (computing, network, storage,etc.) to users (See Chapter 1 for more information). In our case, we have been interested in cloud environments providing storage resources also called *cloud storage* which is a growing trends in data management solution. For instance, the IDC [52] predicts that by 2020, nearly 40% of the data in the world will be stored or processed in a cloud.

When cloud storage environments are built on centralized architectures based on large datacenters, which is an approach that is very common, it may lead to high data access latencies because data are likely located far from their users. Reducing data access latencies is a topic that we addressed in this PhD thesis driven by the idea that data should be located close to their users. Inside datacenters, solutions are usually set up to ensure a certain level of reliability in order to avoid data loss or services unavailability, but the datacenter still represents a single point of failure. That is, if the datacenter becomes unreachable for some reasons (*e.g.* power cut, strikes, etc.) all the data and storage services provided by that datacenter become unavailable.

Building distributed architectures, compounded of geographically dispersed nodes (*i.e.* datacenter, servers, etc.) as well as mechanisms allowing the global architecture to provide storage services even in the case of the unavailability of a subset of the nodes, is more complex than building centralized ones as it introduces several challenges to face. However it becomes necessary to consider these types of architectures. In the context of building cloud storage systems based on distributed architectures, we have been interested in challenges regarding dependability. Dependability is a generic concept used to define or attest the correctness of how a system performs. In dependability, we deal with properties such as system availability, reliability, etc. (See Chapter 3).

A Telecommunication provider such as Orange providing cloud storage services can find in its own infrastructure some assets to face scalability and data access latencies issues. In fact a telecommunication provider owns and masters the network it provides and can leverage several locations in it infrastructure to set storage nodes. Those storage nodes may be datacenters that may be far from users, Points of Presence located close to users at the edge of the network, and also in set-op-boxes or home gateways located in the home of their customers. The ability to control the network traffic and Quality of Service (QoS) as well the opportunity to leverage several locations for storage purpose near and far from users is a real assets that is not likely to be found on non-telecommunication providers.

The contributions of this thesis have been articulated around the means to provide cost-effective, distributed cloud storage architectures that are efficient in terms of scalability and data access latency in the context of a telecommunication provider or Internet Service Provider (ISP), namely Orange. Specifically, we have been focused on distributed storage architectures and dependability

properties such as data redundancy, consistency, and placement. We also considered the data security and confidentiality in the context of storage systems applying data deduplication which is becoming one of the most popular data technologies to reduce the storage cost.

Addressing these topics in the context of a telecommunication provider is something that have not been intensively studied in the literature making it promising and also challenging as several domains and problematics are involved. Nonetheless we studied these topics and proposed viable solutions using innovative ideas that have been presented, published in international scientific conferences, or patented.

Contributions

- **Mistore:** Internet Service Providers (ISPs) furnishing cloud storage services usually rely on big datacenters. These centralized architectures induce many drawbacks in terms of scalability, reliability, and high access latency as datacenters are single points of failure and are not necessarily located close to the users. Mistore is a distributed storage system that we designed to ensure data availability, durability, low access latency by leveraging the Digital Subscriber Line (xDSL) infrastructure of an ISP. Mistore uses the available storage resources of a large number of home gateways and Points of Presence for content storage and caching facilities. Mistore also targets data consistency by providing multiple types of consistency criteria on content and a versioning system. Mistore is described in details in Chapter 4.

- **Data placement and maintenance in Mistore:** This contribution improves the data placement and maintenance in our previous contribution on the architecture of Mistore. In the previous contribution, the data placement method used is a table-based one. That is, all the metadata to locate data within the system where maintained. Table-based data placement methods are simple to deploy however they lack in scalability as the amount of data and metadata to manage increases. In this contribution, we design a more scalable and flexible data placement scheme in which the amount of metadata to maintain depends only on the number of nodes in the system and not on the potentially enormous amount of data to manage. Our solution is described in Chapter 5.

- **The two phase data deduplication:** Data grow at an impressive rate. However a study by the IDC shows that 75% of the digital world is a copy [51]. Although keeping multiple copies of data is necessary to guarantee their availability and long term durability, in many situations the amount of data redundancy is immoderate. By keeping a single copy of repeated data, data deduplication is considered as one of the most promising solutions to reduce the storage costs, and improve users experience by saving network bandwidth and reducing backup time. However, this technology still suffers from many security issues that have been discussed in the literature. In this contribution, we have been focused on client side attacks and designed a method that we called two-phase data deduplication allowing to build a storage system performing a client-side data deduplication that is secure against attacks from malicious clients. Our solution remains effective in terms of storage space and network bandwidth consumption and have been patented. We describe the solution in Chapter 6

- **An enterprise data outsourcing method:** While the adoption of cloud storage services to outsource every type of data is steadily growing, enterprises are still reluctant to outsource completely all their data to the cloud because its benefits come with some inherent risks implied by the fact that full control on data is given to the cloud storage providers. In this contribution, we designed a solution allowing enterprises to outsource data to the cloud while keeping control on them. Our solution leverages a cloud gateway located at the enterprise premises that aims to provide control on data stored in the cloud, improve data access performance, optimize storage and network bandwidth usage, and provide data confidentiality. To achieve that, this cloud gateway implements a metadata management on-premises as

well as features such as caching, flexible data placement over multiple cloud storage services, data encryption. Our method also used the secure two-phase data deduplication to improve storage and network bandwidth consumption. We describe this solution Chapter 7.

- **ODISEA:** During this thesis, we have been involved in the Open Distributed Networked Storage Architecture (ODISEA) project. It was a french collaborative project involving several industrial and academic partners, namely Orange Labs, University Pierre et Marie Curie, Technicolor, Ubistorage, eNovance, Institut Telecom). This project has been selected by the FUI (Fonds Unique Interministeriel). The FUI is a French program supporting applied research by providing funds to selected innovative and collaborative (*i.e.* involving industrial and academic partners) project. Orange Labs was the leader of the project. The project aimed to provide an open and distributed storage cloud by designing an architecture exploiting and optimizing network resource in the core network and at the edge of the network to manage and deploy storage resources and services. We have been involved in several tasks such as brainstorming or the metadata system development for the system prototype that have been presented to the Salon de la Recherche organized by Orange in 2013.

Results

Scientific articles

- Pierre Meye, Philippe Raïpin, Frédéric Tronel, and Emmanuelle Anceaume, "A secure two-phase data deduplication scheme", Proceedings of the 6th International Symposium on Cyberspace Safety and Security (CSS). Paris (FRANCE) 2014.

- Pierre Meye, Philippe Raïpin, Frédéric Tronel, and Emmanuelle Anceaume, "Mistore: A distributed storage system leveraging the DSL infrastructure of an ISP", Proceedings of the International Conference on High Performance Computing & Simulation (HPCS). Bologna (ITALIA) 2014.

- Pierre Meye, Philippe Raïpin, Frédéric Tronel, and Emmanuelle Anceaume, "Toward a distributed storage system leveraging the DSL infrastructure of an ISP", Proceedings of the 11th IEEE Consumer Communications and Networking Conference (CCNC). Las Vegas (USA) 2014.

Patents

- Pierre Meye, Philippe Raïpin, Method for storing data in a computer system performing data deduplication. WO2014154973. 2014

- Philippe Raïpin, Pierre Meye, Method of processing data in a distributed storage system. WO2014096607. 2014

White paper

- Sébastien Canard, Pierre Meye, Philippe Raïpin, Jacques Traoré, and Mohammed Tuhin. "Cloud storage and confidentiality", Research paper (Orange Labs). 2014

Talks

- Pierre Meye, "A secure two-phase data deduplication scheme". WOS 4: Workshop on Storage and Processing of Big Data. Rennes (FRANCE) 2014.

Organization

The thesis is organized as follows. Chapters 1 to 3 introduce and discuss fundamental concepts and techniques that are relevant for understanding the problems and the contributions made in this thesis. More specifically, in Chapter 1 we review the basics of the cloud computing paradigm, the context in which this thesis is contained. In Chapter 2 we discuss about data storage and its importance. We present the basics of dependability in Chapter 3 with a focus on the properties that are relevant to our contributions that we present from Chapter 4 to 7. More specifically in Chapter 4 we present Mistore the distributed storage system designed during this thesis. In Chapter 5 we describe a data placement and maintenance method suitable to manage data stored in Mistore. We present in Chapter 6 the two-phase data deduplication. In Chapter 7 we describe a method dedicated to enterprises allowing to outsourcing data in cloud storage while preserving control on the outsourced data. Finally, we conclude this thesis in Chapter 8.

Acknowledgments

I would like to thank everyone who contributed directly and indirectly to the achievements of this work.

First and foremost, I would like to thank my advisors, Philippe RAIPIN, Frédéric TRONEL, and my thesis director Emmanuelle ANCEAUME. They gave me the opportunity to follow this PhD thesis and allowed me to work on topics that interested me particularly, while also providing me valuable feedback and guidance throughout this work.

I would like to express my sincere gratitude to all the members of the jury; my reviewers, Paul EZHILCHELVAN, Reader at Newcastle university and Marc-Olivier KILLIJIAN, Research Director at CNRS/LAAS; my examinators POTOP-BUTUCARU Maria, Professor at Pierre et Marie Curie University, Achour MOSTEFAOUI, Professor at Nantes University, and Jean-Louis PAZAT, Professor at INSA and president of the jury. I would like to thank Eric MOURGAYA, System Engineer at Arkea who accepted to be a guest member of the jury. I appreciate their insightful and constructive comments.

I would also like to thank Jerome BIDET and all the members of the CSE research team at Orange Labs for their support during these years.

Last but not least, I would like to thank and dedicate my PhD thesis to my parents; to Jerzy, Aristide, Marie, Ophelie, and Yvanna. They always believed in me and none of this would have been possible without their unconditional support and encouragements.

"let the wise listen and add to their learning,
and let the discerning get guidance"

— Proverb 1:5

Contents

List of Figures

List of Tables

1

Cloud computing

Since the contribution of this thesis is related to a cloud environment where the resources provided to clients is the storage, we think that is relevant to review the fundamental concepts of the cloud computing paradigm before diving into cloud storage. This is the objective of this chapter.

Contents

1.1 Introduction

Cloud computing is a paradigm promising to speed up and ease services deployment to clients and reduce the cost of such service deployment. Typically, cloud vendors provide virtually unlimited resources (network, storage, compute,etc) or applications available on-demand that clients will pay-per-use. There exist several attempts of definition of the cloud computing paradigm, the most notable definitions being the NIST definition of cloud computing [85] and the Berkeley cloud computing view [10] which also addresses the potential and obstacles to the cloud computing growth. This section is inspired by the NIST definition of cloud computing which is certainly the most adopted definition. The NIST defines the cloud as following:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

Figure 1.1 shows the differences between managment responsability of clients and cloud service providers in the traditional IT model and in the three service models of the cloud computing.

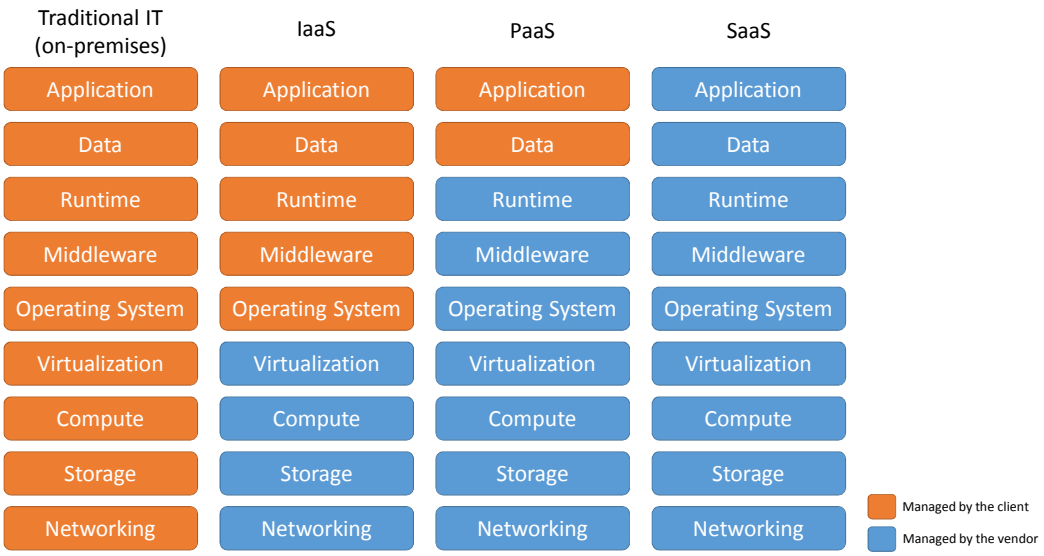


Figure 1.1: Cloud computing stacks regarding the different cloud services and the traditional IT service model.

In the remainder of this chapter, we review in Section 1.2 the main characteristics of cloud environments. In Section 1.3 we describe the different cloud service models. Section 1.4 describes different cloud deployment scenarios. Section 1.5 concludes this chapter and opens the next one.

1.2 Cloud characteristics

Here are the essential characteristics that a computing cloud environment should provide.

On-demand self-services: The clients can get network, storage, or computing resources automatically without interaction with the Cloud Service Provider.

Ubiquitous access: Resources can be accessed from anywhere using any device.

Multi-tenancy and Resource pooling: The ability to use shared resources and provide them to clients (*i.e.* tenants) with on-demand self-services capability in such a way that each client is isolated from the others and unaware of the fact that they use shared resources. Multi-tenancy and resource pooling commonly rely on visualization technologies.

Elasticity: The ability to adapt (*i.e.* increase, decrease, delete) the provided resources automatically depending on some conditions at runtime (*e.g.* a peak of charge). Elasticity allows the pay-per-use model in which clients pay only for the resources used. Note that the scaling process may be configured by the client or the Cloud Service Provider.

Measure usage: The ability to monitor and keep track of the provided resources. This allows transparency through control and reporting between the Cloud Service Provider and the clients.

1.3 Cloud Service models

A cloud service model represents an ensemble of resources that is provided as a package to clients. Here after the most common service models:

Infrastructure as a Service (IaaS): The possibility provided to clients to provision fundamental raw IT resources such as computing capacity, storage, networks (See Figure 1.1). The clients can deploy and run arbitrary applications using the provided resources. This model is mostly dedicated to clients in the need of a high level control on the cloud environment. In fact resources provided by IaaS are generally not pre-configured. Amazon is one of the most popular IaaS providers with the Elastic Compute Cloud (EC2) service[1].

Platform as a Service (PaaS): A cloud environment provided to clients containing deployed resources that are pre-configured. Clients can deploy custom applications using languages, library, services, and tools supported by the provider. Clients do not have control on the underlying cloud infrastructure but can have control over the deployed applications and possibly configuration settings for the resources in the provided environment (See Figure 1.1). The Microsoft Azure platform[2] and the Google App Engine[3] are prominent example of PaaS services.

Software as a Service (SaaS) The capability to clients to use a provider's application running on a cloud infrastructure (*e.g.*, a webmail service). The applications may be accessible from various client devices through either a thin client interface, such as a web browser, or a program interface. Clients of SaaS generally have a very limited control over the SaaS implementation as well as over the underlying infrastructure (See Figure 1.1). Salesforce is a well known SaaS provider[4].

Note that a PaaS can be used by a client to deploy a cloud service as a SaaS that will be provided to other clients. Figure 1.2 illustrates relations between actors (*i.e.* vendors, application developpers, and end-users) and services (*i.e.* IaaS, PaaS, and SaaS).

1.4 Cloud Deployment models

A cloud deployment model specifies how a cloud environment is set up. The focus is made on the relation between the actors providing the infrastructure (*i.e.* the raw IT resources) that we call the Infrastructure Providers and the actors providing services such as PaaS and SaaS. There

[1] http://aws.amazon.com/ec2
[2] http:///www.microsoft.com/windowsazure
[3] http://code.google.com/appengine/
[4] http://www.salesforce.com

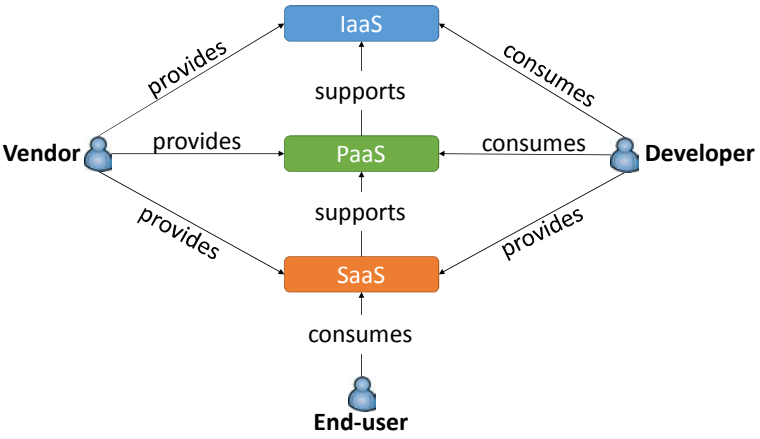


Figure 1.2: Relations between the different cloud service models, cloud vendors, developers, and the end-user.

exist several cloud deployment scenarios depending on which actor provides what resources. Here after we review some of the most common cloud deployment scenarios. There are illustrated in Figure 1.3.

Private cloud: The cloud environment, infrastructure and service, is owned and/or used by a single entity or organization (*e.g.*, an enterprise). Figure 1.3a illustrates a private cloud. It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may be deployed on- or off-premises.

Public cloud: The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. This is the most common cloud scenario. Figure 1.3c illustrates a public cloud. Note that the Infrastructure Provider and the Service Provider may be different.

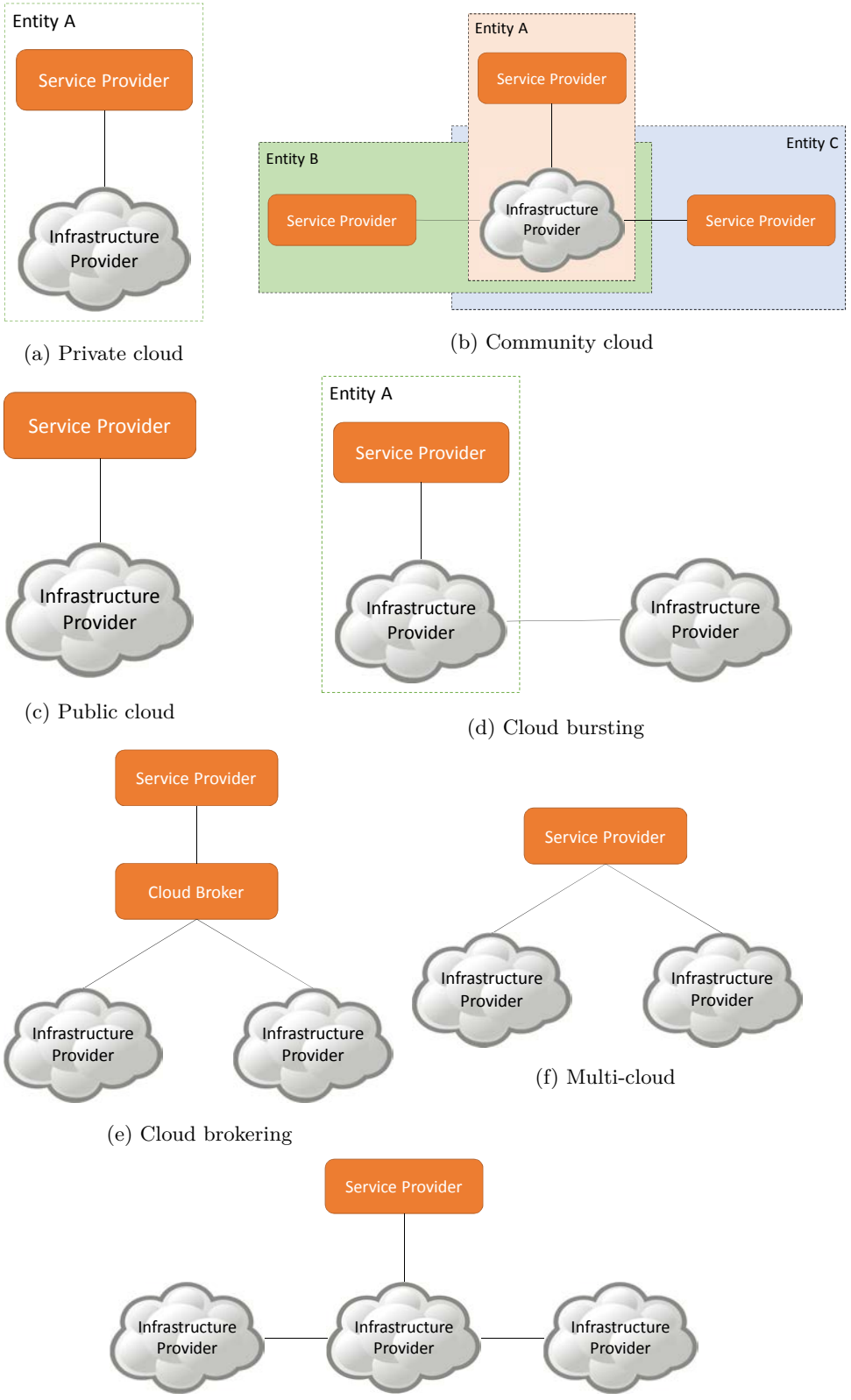
Community cloud: The cloud environment, infrastructure and service, is provisioned for an exclusive use by a specific community of clients of organizations that have shared concerns (*e.g.*, mission, security requirements, policy, and compliance considerations). The cloud environment may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them.

Hybrid cloud: This type of cloud is created by composing two or more cloud infrastructures owned by different providers. There is some advantages to leverage several cloud infrastructures instead of a single one [79, 130]. For instance it allows to avoid vendor lock-in when the client is able to move freely data or tasks from on provider to another. It also improves reliability and availability when redundancy is leveraged across different providers allowing to tolerate unavailability or outages that may occurs at any individual cloud infrastructure. Cloud composition also helps to be compliant to some legislation. In fact it is unlikely that the infrastructure of a single provider can span all the countries in the world and while some applications can have legislative requirements, composing several cloud infrastructures can help to deploy applications that can works across larger geographical regions and be compliant to several legislations. Moreover, because all cloud infrastructures are not equal, a cloud infrastructure may be the best choice for a specific task than another. There exists many type of hybrid cloud deployments. Here after examples of several types of hybrid cloud deployment scenarios:

- **Cloud bursting:** Figure 1.3d illustrates a private cloud bursting. In this scenario a private cloud can extend its capacity using dynamically the resources provided by another Infrastructure Provider. Its can be used for example to get more resources when the limits of the private cloud are reached.
- **Multi-cloud:** Figure 1.3f illustrates a multi-cloud scenario. Here, the Service Provider is able to select which cloud infrastructure to use in the service. A multi-cloud approach can help to avoid cloud vendor lock-in, improve availability by using redundancy across two or more cloud infrastructures, choose the suitable cloud infrastructure for a specific task.
- **Cloud federation:** Figure 1.3g illustrates a cloud federation scenario between three Infrastructure Providers. In this case, Infrastructure Providers are collaborating together. It allows an Infrastructure Provider to offload some tasks to others, or to support services that can span a larger geographical region. The federation between the Infrastructure Providers is transparent to the Service Provider. In fact the Service provider subscribes to an Infrastructure Provider and it is not necessary aware of existing cloud federation agreements.
- **Cloud brokering:** Figure 1.3e illustrates a cloud brokering scenario. A cloud broker is an intermediate actor between Service Providers and Infrastructure Providers. The cloud broker is responsible of the selection of the suitable Infrastructure Provider on behalf of the Service Provider depending on the requirements of the Service Provider. The cloud broker may have partnerships with several Infrastructure Providers. A cloud broker actually hides the complexity of selecting, subscribing, and accessing to resources provided by Infrastructure Providers.

1.5 Summary

In this chapter we have presented the basics of the cloud computing paradigm. While vendors are providing more and more cloud computing solutions, their adoption is still below the expectation. This is due to some inherent issues that suffer cloud computing environment. Among these issues, we can notice dependability, lost of control because the client does not necessarily master all the stack of the resources, and the flexibility to avoid vendor locking situation [71, 107]. In this thesis our contributions regarding these issues take place in a cloud computing environnement where the resource provided to clients is storage.



(a) Private cloud

(b) Community cloud

(c) Public cloud

(d) Cloud bursting

(e) Cloud brokering

(f) Multi-cloud

(g) Cloud federation

Figure 1.3: Cloud deployment models

2

Data Storage

The purpose of this chapter is to give some understanding about the data and the technologies used to manage them. We review the different natures of data, some storage service features, and data storage access technologies such as block-based, file-based, object-based storage.

Contents

2.1 The different types of data

There is different types of data and a storage system may be optimized for a certain type of data depending on the needs of the applications. Data can be structured, unstructured, or semi-structured. There is also metadata, also known as data about data.

- **Structured data**: They represent data that can be classified, modeled, and attributed to a fixed field. They are usually associated to data tables, spreadsheets, or to databases. In this kind of data, it is possible to define the field of the data that will be stored and their type (*e.g.* numeric, text, name, date, address, etc). They are simple to collect, stored, and retrieved or queried. The Structure Query Language (SQL) [34] is a well known method to manage structured data.

- **Unstructured data:** They are the data that cannot be easily classified, modeled, and attributed to a fixed field. These kind of data include documents (*e.g*, pdf files, PowerPoint presentations, etc), photos, videos, web-pages, etc.

- **Semi-structured data:** They have structured and unstructured data properties. Email can be considered as semi-structured as there are composed of different part that can be structured (the header) and unstructured (the body, and the attachment part). XML (Extensible Markup Language) and JSON (JavaScript Objects Notation) documents are also a form of semi-structured documents.

- **Metadata:** The National Information Standards Organization (NISO) defines metadata as follows: *Metadata is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource. Metadata is often called data about data or information about information* [97]. Example of metadata of a data item may be the creation date or the size of this data item.

2.2 Data storage service

When in comes to storage services, from the users point of view, details of conception and implementation of the storage service are not really the most important. What really matter is usually the features and the level of dependability provided by the cloud storage service. In the following we review some of these features.

Backup and versioning

Nowadays cloud storage service are more and more used to backup data on a cloud. Rationales are the unlimited storage space, the low cost, and the availability that the cloud promises. In general, users get a local copy of these data. Remote copies are used for backup when the local copy is lost due to a bad manipulation or the crash of the local storage devices. Copies stored in the cloud also provides to users the ability to access to their data from everywhere at any time even if they do not have access to their local storage device. The access can be done via a web browser for example. A feature of backup that is very important in cloud storage services is the versioning system. In fact, when files are updated, if the cloud storage service implements a versioning system, users are able to get any prior versions of their data. This can be useful in the case of a bad manipulation, a user can recover a prior version. Some cloud storage services [1][2][3] also allow to recover a file that has been deleted within a certain retention period.

[1] https://www.dropbox.com
[2] https://onedrive.live.com
[3] https://www.google.com/drive

Data synchronization

With the evolution of technologies, it becomes common for users to own several mobiles devices (laptop, tablet, smartphone, etc). Data synchronization is a feature that is more and more provided by cloud storage services. It allows to get a local copy of data on several devices. Having a local copy improves access performance as users do not have to get their data from the cloud. In other to keep the different copies consistent and up-to-date, when data are modified on a device, the updates are propagated toward the other devices of the user. A synchronization mechanism must provide ways to resolve conflicts when multiple data modifications appear on several devices concurrently [76]. Note that if synchronization is provided with a backup feature, a copy can also be available in the cloud so that users can get access to their data even if they do not have access to their devices, via a web browser for example.

Content sharing/delivery

Due to the proliferation of mobiles devices and the evolution of the technologies, users create and share more and more data [22]. This is the advent of the user-generated content. Users are not anymore only consumers but they are also producers [35]. Thus, content sharing/delivery mechanism are important to users in order to share data with friends, family, colleagues, or with everyone on the web. Data shared may have some restrictions on the set of operations that will be allowed to the recipients, *e.g.* the permission to read, update, or delete. Most of the user-generated content been unstructured data, it represents the type of data exhibiting the largest growth with the evolution of technology.

2.3 Storage access technologies

In this section, we review the most popular storage access technologies namely block based, file-based, and object based storage. We will see that the difference between them is mainly how the storage is presented to the system, the protocol used to process read and write requests, and the way storage components is linked to the host.

2.3.1 Block-based storage

2.3.1.1 Direct attached storage technology

A basic method to perform data storage is to use a storage component (*e.g.*, a hard drive, USB keys, disk array etc.) that is directly connected to the host (*e.g.*, a server, laptop, etc). This is what we call a Direct Attached Storage (DAS) (See Figure 2.1) which is a simple architecture and provides the best performance as applications on the host can store and retrieve data directly from the storage component. There is no network devices such as hub, switch, or router between the storage component and the host. Communication between the host and the storage component is made via the I/O bus of the host through block storage interfaces (*e.g,* IDE, SATA, SAS, SCSI, etc) that split and transfer data in raw data blocks of the same size. The DAS is presented to the host as a locally attached disk, making DAS suitable to store various type of data (*e.g.* documents, databases, Operating Systems, virtual machines, etc.)

File system: Note that to actually use the storage component a *file system* or *filesystem* is usually put on top of the storage component. The filesystem is the link between applications and the storage component. Note that some applications such as databases may implement their own filesystem. We can divide a filesystem into two parts (See Figure 2.2)[87]:

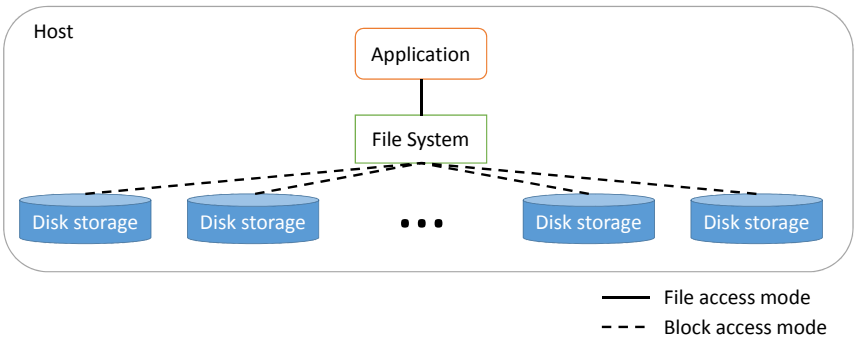


Figure 2.1: Direct Attached Storage (DAS).

- **The user component:** It handles the way data are presented to applications and the ability to access to these data. For example through a hierarchical logical representation of directories or folders containing the files.
- **The storage component:** It handles the way data are actually stored and retrieved on the storage component and its metadata.

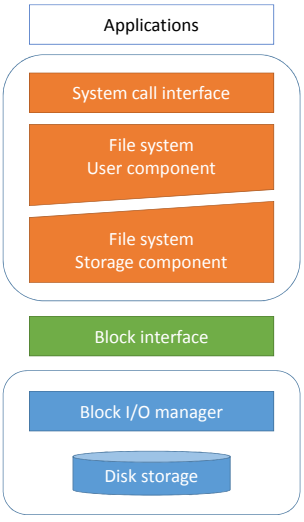


Figure 2.2: Block-based storage. The file system is divided in two parts the user component and the storage component.

One of the limitations of DAS is that with its basic configuration, data on the storage component can not be shared among several hosts. Even it is possible for the host, to which the storage component is attached, to share the storage component to other hosts, it requires special support and it is still limited by the hardware connection and thus lack in scalability.

2.3.1.2 Storage Area network (SAN)

Storage Area Network (SAN) is a storage networking solution that solves the scalability and storage components sharing issues of DAS solutions. A SAN is a network that is formed by a switched fabric of storage components and which is dedicated to the storage operations. While a host sees a DAS as a locally attached disk, a SAN is presented to hosts as a network dedicated to storage. Transfers are made using protocols such as Fiber Channel, iSCSI. It is then possible to scale by

adding storage components to the switched fabric that form the SAN. Applications on hosts can then access to a large capacity of storage directly with an improvement in performance since data splitted in multiple blocks can be stored and retrieved in parallel on a large number of storage components interconnected by a possibly high speed network. Similarly to a virtual environment, a SAN solution allows to create and isolate logical storage zones on top of raw storage making a SAN suitable to be used by various kind of applications over the network (*e.g.* documents, databases, Operating Systems, virtual machines, etc.). However the security mechanism in a SAN creates isolation to storage components level not to data or data block levels leading to some security issues. Furthermore, since SAN presents raw storage to hosts, it is the applications or the file systems that are responsible of the data and metadata management (See Figure 2.3). This point makes data sharing among several applications and hosts complex since the applications or the file systems on these hosts need to have the same knowledge about how the data are stored (*i.e.* the metadata) and coordinate themselves to store data in order to keep the SAN system consistent.

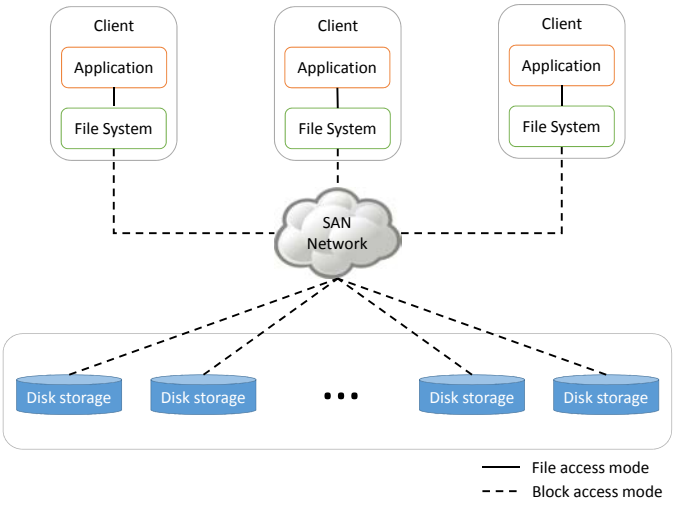


Figure 2.3: Storage Area Network (SAN).

2.3.2 File-based storage

2.3.2.1 Network attached storage (NAS)

A Network Appliance Storage (NAS) is the most popular form of implementation of a file-based storage. A NAS is actually a dedicated server located in the same network of the clients or a different but accessible network (See Figure 2.4). A file-based system presents data to clients and applications using a file system data representation *i.e.* a hierarchy of files and folders. The storage component used by a file server may be a DAS or a SAN. The Network File System (NFS) and Common Internet File System (CIFS) are the most popular protocols working on ethernet and allowing applications to communicate with a NAS through the network.

A NAS provides a solution to the data access security and sharing issues across several applications and hosts via a centralized architecture. In fact the server manages all the metadata of the data that is stored in the storage component attached to it. It enhances security by implementing a control on the access though Access Control Level (ACL) per folder or file. However these benefits come at the expense of performance and scalability due to the centralized architecture of a NAS which introduces a single point of failure as all the requests have to be processed by the server. Note that even if the storage component used by the server is a high-speed SAN network, the performance of the SAN will be hidden by the contention at the server. Moreover, a file-based system only shares data using a file system data representation *i.e.* a hierarchy of files and folders. This makes file-based systems suitable for applications dealing only with files and folders.

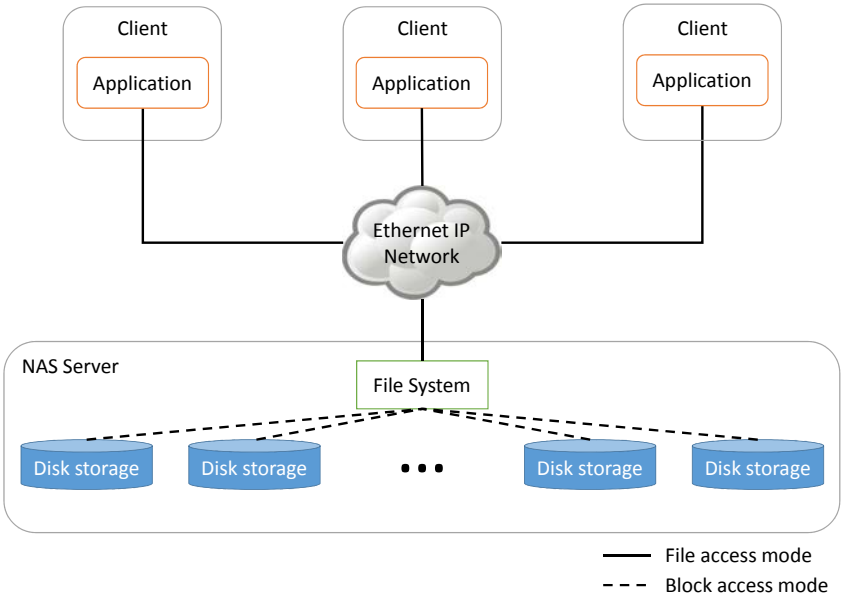


Figure 2.4: Network Attached Storage (NAS).

2.3.2.2 SAN file system

A SAN file system is a combinaison of a NAS and a SAN storage in order to benefits from the performance of the SAN and the simplicity of data sharing of a NAS. In a NAS file system, a server is responsible of the entire file system (*i.e.* the metadata in the system). Clients address their metadata request to this server however the server does not make the I/O requests to the storage component like in the case of a simple NAS. The server actually direct clients the SAN storage. This makes the clients accessing directly to the storage components reducing the contention at the server and improving performance (See Figure 2.5). However this architecture introduces security concerns in fact, the storage component in the SAN are not aware of the relations between clients and the data they store so upon I/O request by clients, it can not perform any control. In this architecture, clients are assumed to be trusted.

2.3.3 Object-based storage

In block-based and the file-based storage previously presented, storage system designers are limited by the possibility of the block storage interface and metadata management leading to trade-offs in system performance, scalability, security, and data sharing possibility. Object storage is a new technology aiming to provide at the same time flexibility of medata management as well as security, performance, scalability, and data sharing.

Object storage relies on intelligent storage components that we call Object-based Storage Devices (OSDs) which possess several resources such as storage capacity, memory, high-speed network connection, or computing resources. All these resources make OSDs able to be aware of the data been stored and the relation between them and the applications. These resources give to storage components the ability to process data being stored in them.

The access to OSDs is made through an object storage interface. Figure 2.6 illustrates the difference between a traditional block-based interface (used in DAS, SAN, and NAS), and an object interface. In object storage, the storage interface is changed from block I/O interface to object interface, and the storage component of the file system is offloaded to the OSD which is responsible of the storage space allocation and the metadata management.

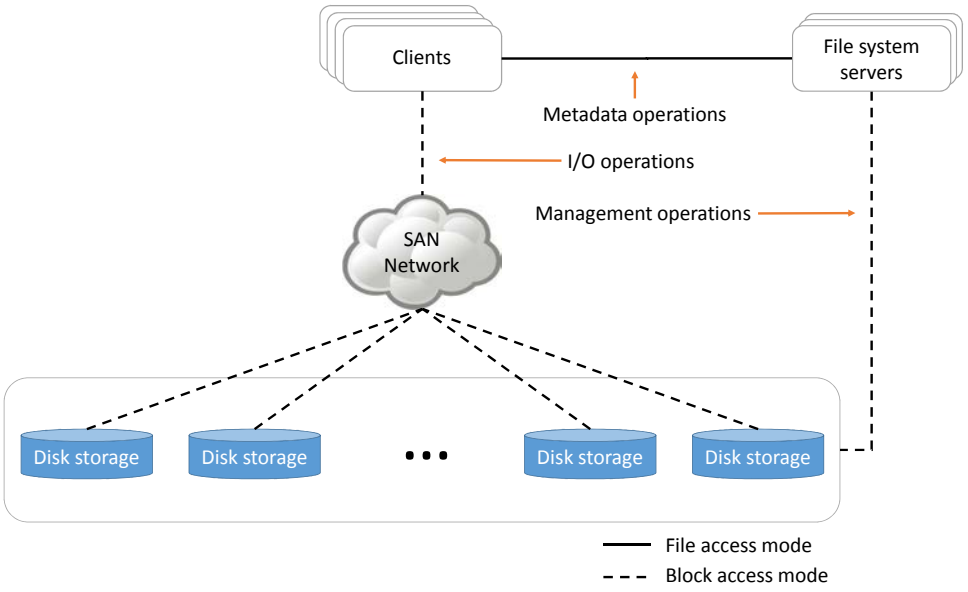


Figure 2.5: SAN file system.

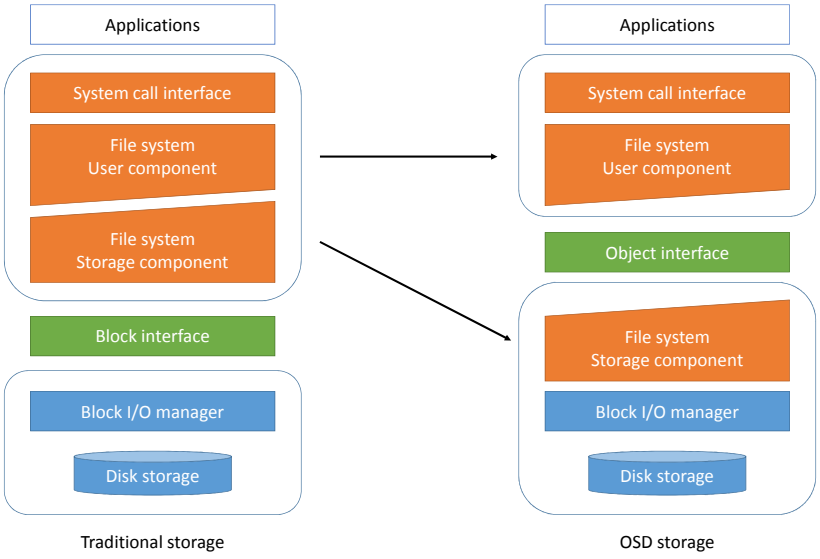


Figure 2.6: Differences between traditional block storage and an implementation of object storage. In object storage the storage component of the file system is offloaded to the OSD.[87]

In object storage, data are stored as *objects* and managed through a flat hierarchy unlike traditional file systems which implement a hierarchical structure. An object is made of a data item and its metadata. Unlike in block-based storage where data are managed as data blocks of the same size, in object storage, there is no limitation on the size of the data or block of data. Object storage associates an unique identifier to objects. Objects are accessed using the unique identifier. There is no limit in the amount of metadata you can add to an object so they can be more detailed than in a traditional file system. Note that an application may implement a metadata system allowing to locate the data by their identifiers. These metatada can be stored separately using a

database, an other storage system, or in an object storage system. The object interface supports simple operations as get, put, delete, update, making it flexible and convenient be to be used over network protocol such as HTTP. Popular object interfaces are Amazon S3 and CDMI.

Figure 2.7 illustrates an object storage system. A metadata server manages the information necessary to locate the data (*i.e.* the association between data identifiers and the different OSDs for example). The server grant capabilities to client that can access to data directly on the OSDs. Because the OSDs are aware of the data they store, they can verify the capability of the clients in order to grant the access to a data item or even to a block of data. Object storage offloads the storage space allocation and metadata to storage components and allows direct access to storage component which improves performance, scalability and data sharing across application. It also improves security since security policy can be enforced by the OSDs thanks to the intelligence they get from the different resources they possess.

Due to all its advantages and flexibility to be used with the HTTP protocol, object storage is becoming very popular with the advent of cloud computing and is often used in cloud storage services to store unstructured data (See Section 2.1). It is also used to store data that are less and less accessed (*i.e.* backup or archival).

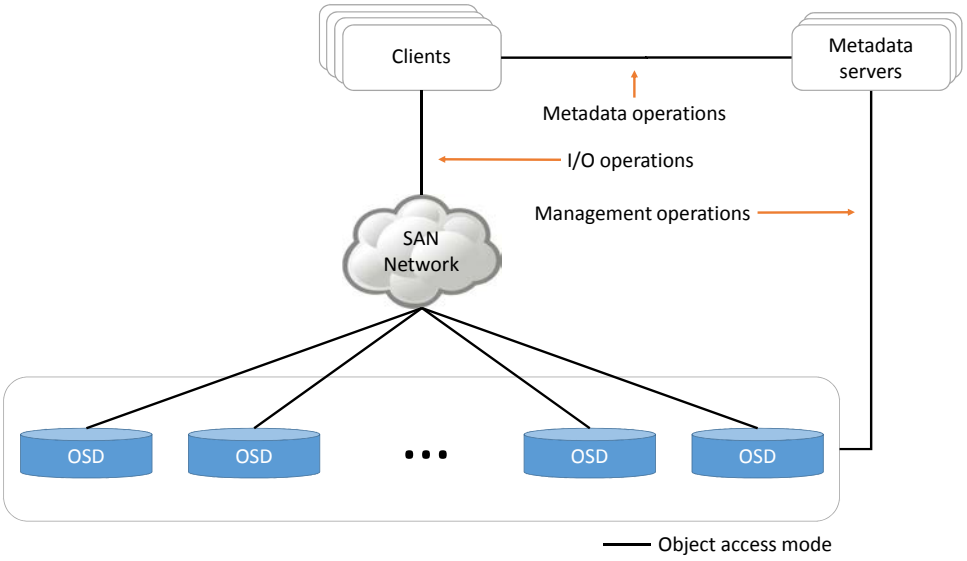


Figure 2.7: Object storage.

2.3.4 Unified storage

A unified storage is a system allowing several data access modes to a single storage platform, for example a file-based, a block-based, and object-based access. A unified storage provides ease of deployment and management as only a single platform is required instead of using separate platforms for different data access modes. An unified storage is illustrated in Figure 2.8 where the backend storage used is actually an object-storage and metadata servers on top of the storage backend implement various kind of data access interfaces. Thus, several existing applications can access to the same storage platform using different interfaces supported by the platform, for example, using CIFS and NFS for file-based access, iSCSI and Fiber Channel for block-based access, and S3 or CDMI for object-based access. Unified storage storage is becoming more and more adopted. Ceph[4] (block, file, object) and GlusterFS[5](object, file) are examples of such systems.

[4]www.ceph.com/
[5]http://www.gluster.org/

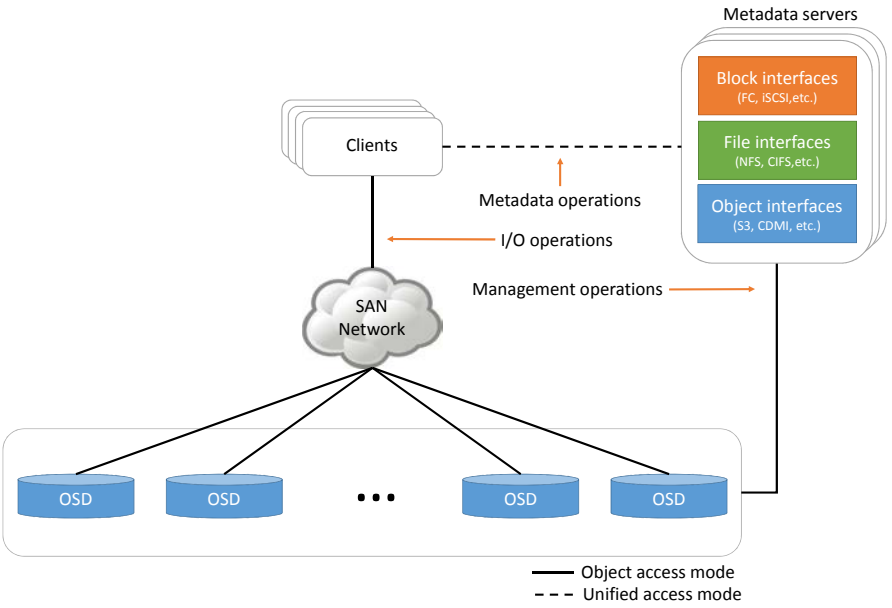


Figure 2.8: Example of a unified storage architecture.

2.4 Summary

In this chapter we covered the basics of storage technologies and we reviewed some of the users interests in terms of features in storage systems. In order to meet the users interests and sustain the ever growing amount of data, storage providers have to face several challenges. Some of them wil be addressed in this thesis.

Our contributions target dependability properties and storage cost optimization. Addressing dependability properties allows to provides efficient, scalable, secure, and reliable storage services. Implementing storage cost optimization is also appealing for storage providers since it allows them to sustain the growing amount of data to store while keeping the storage cost low.

We review the dependability concepts in Chapter 3. After that we present our contributions about dependability in Chapter 4 and 5. Chapter 6 and Chapter 7 deal with confidentiality and a storage cost optimization technology called deduplication.

3

Dependability

In this chapter, we review the basics concepts of dependability. Then we focus on the properties of dependability that are the most relevant, regarding our thesis, to storage systems; namely, availability, durability, consistency, and security. We present their rational and explain how they are usually provided in storage systems.

Contents

3.1 Basic concepts of dependability

Dependability is a generic concept used to define or attest the correctness of how a system performs. It is defined in [11] as the ability to deliver services that can justifiably be trusted. Usually it is defined by several attributes. Although there exists several definitions of what dependability is, it is generally accepted that dependability encompasses a certain number of attributes that can be called core dependability attributes. These are there descriptions of the core attributes as defined in [11]:

- **Availability:** The readiness for correct services. Regarding security the availability is perceived as the fact that only authorized actions are possible.
- **Reliability:** The continuity of correct services.
- **Safety:** The absence of catastrophic consequences for the user(s) and the environment.
- **Integrity:** The absence of improper system alterations. Regarding security it is perceived as the absence of unauthorized actions.
- **Maintenability:** The ability to undergo modifications and repairs.
- **Confidentiality:** The absence of unauthorized disclosure of information.

In these core attributes we can notice that some attributes are related to security which compounds the attributes of availability, integrity, and confidentiality. Note that depending on the application, the dependability attributes may vary. Not all the applications require the same dependability attributes. On the other hand, the core attributes can also be divided in several secondary attributes to refine or bring more precision on the attribute.

In this chapter we will focus on the attributes that are more common in the vocabulary regarding cloud storage namely data availability, durability, consistency, confidentiality, and integrity.

3.2 Data availability and durability

Data availability is a term used to describe the accessibility of data. Data are available if they can be accessed on demand (*i.e.* when a user requests it). Data durability means that data stored in the system is not lost even in the presence of disk or node failures in the storage system [26]. The difference between them is that data can be durable but not available. For example, if some data are stored on a storage node that is offline, these data are not available but are not lost. These data become available as soon as the storage node gets back in an online mode. Notice that it is impossible to ensure data availability without ensuring durability. In fact if eventually data get lost the availability will be impossible to achieve.

To the best of our knowledge, it is not possible to build nodes or devices that never fails. Thus, hardware and more generally nodes failure is even considered as the norm rather that the exception, especially when the system involves thousands of components [115, 55, 53]. A study by Visions Solutions [120] states that:

> *According to Dunn & Bradstreet, 59% of Fortune 500 companies experience a minimum of 1.6 hours of downtime per week. To put this in perspective, assume that an average Fortune 500 company has 10,000 employees who are paid an average of $56 per hour, including benefits ($40 per hour salary + $16 per hour in benefits). Just the labor component of downtime costs for such a company would be $896,000 weekly, which translates into more than $46 million per year.*

It thus important to design systems that can tolerate components failures while ensuring data availability and durability since failures, system downtime, and data lost can have negative impacts for the clients but also for the Cloud Storage Provider.

Data availability and durability is usually achieved with redundancy implemented by software. In the following we present how *data redundancy* (See Section 3.2.1 is leveraged to ensure data availability and durability. We also review concepts of *data placement* (See Section 3.2.3) and *data maintenance* (See Section 3.2.2) that can have an impact on the data availability and durability.

3.2.1 Data redundancy

Data redundancy is the common method used to ensure data availability and durability. Having data redundantly stored in the system at different locations (*i.e.* hard drives, storage nodes, storage servers, datacenters, etc.) helps to cope with failures. If a copy of a data item is lost because its storage node failed for example, we can recover the data item from another copy stored at a different location.

The replication is a technique widely adopted to create redundancy in which several copies of a same data item are stored ideally in different locations such that the loss of some copies of that data does not affect its availability and durability. A common redundancy strategy is to create three copies [117, 53] also called *tri-plication*. Note that the number of copies or the degree of redundancy has to be maintained. In fact as losses occurred, new copies have to be created (See Section 3.2.2). Data replication is also used to improve availability regarding access latency. Most of cache systems are based on replication of data closed to users in order to reduce the access latency.

The replication is very easy to implement however it may require a large amount of storage space to provide data availability and durability. Other redundancy techniques include erasure coding techniques [135, 32, 17, 110] that aim to provide the same level of data availability but with a lower storage space consumption.

3.2.2 Data maintenance

Data redundancy is not enough to ensure long term availability and durability of data in the system, a data maintenance is required. In fact, we know that storage nodes unavailability (due to overload, failures, etc) are inevitable so the system may lose some data if no actions are taken when failures occur. It ensures the data availability and durability despite the occurrence of transient or permanent nodes unavailability.

One of the main requirements of a data maintenance system is a monitoring service that can detect all the changes (*e.g.* addition or removal of nodes) in the system [24]. It is usually implemented by sending periodically small messages sometimes called *keepalive* or *heartbeat* to a node that is checking or maintaining the aliveness of the sending node.

Storage nodes unavailability reduces the redundancy level of the data stored on them so eventually some data may be lost. Unavailability may be due to several events (*e.g.*, nodes failures, overload, network failures, etc.) but we can distinguish two types of unavailability:

- **Temporary unavailability:** In this case, the storage node is not available at a specific moment. The node may been powered off for example. In this case, the data stored on such an unavailable nodes are unavailable too but not lost. As soon that the node gets online in the system, its data become available.

- **Definitive unavailability:** the storage node goes offline and will never come back online in the storage system. This may be due to a failure for example. In this case, all the data stored on the unavailable node are lost.

A data maintenance system must maintains a certain level or factor of data redundancy despite storage nodes failures. Note that the level of redundancy may be defined, specified in the form of a threshold or a minimum redundancy level [32]. There is two common methods to address nodes departures, reactive and proactive maintenance methods.

3.2.2.1 Reactive maintenance

A reactive maintenance system applies action when a node unavailability is detected. We can qualify a reactive maintenance system to work in an eager or lazy mode:

- **Eager mode:** In this mode a level of redundancy is defined. When a storage node is unavailable, the system re-creates immediately redundancy for all the data impacted by this unavailability. One of the concerns of this solution is that there is no difference between temporary and definitive unavailability. The system creates redundancy even when the unavailability is just temporary. The consequences are an unnecessary consumption (computing, network bandwidth, storage space, etc). However, when some nodes get back online, the redundancy level of their data may exceed their pre-defined level. Managing the redundancy level when the temporary unavailability are frequent makes the system maintenance difficult and overwhelming. In fact, a reactive system can create a spike in the network bandwidth consumption to respond to an unavailability.
- **Lazy mode:** This mode mitigates the unnecessary resources consumption in data reconstruction present in eager reactive maintenance systems. The idea is to not create replicas as soon as an unavailability is detected. It helps to reduce the redundancy created in case of temporary unavailability. Usually these methods define a threshold in the redundancy level of data and assume that data availability and durability can be ensured when the redundancy level is equal or superior to the threshold. When an unavailability is detected and the redundancy level of some data is still superior to the defined threshold, no redundancy is added. When the redundancy level is lower than the threshold for some data, the system recreate redundancy for those data. These strategy is appealing when the redundancy is created using erasure codes that can reconstruct several fragments at the same time.

We can notice however, that these two modes do not manage efficiently the network bandwidth consumption. In fact in the two modes, when the system decides to rebuild data lost, it is done in a busty mode without concern of the network state. This, can create spikes in the network bandwidth.

3.2.2.2 Proactive maintenance

Proactive maintenance systems aim to make a better use of the network bandwidth consumption than reactive maintenance systems. Contrarily to reactive maintenance systems, the proactive maintenance systems does not respond to nodes unavailability. Instead they create redundancy continually at a defined rate. We can distinguish two kinds of proactive maintenance systems, static and adaptive:

- **Static:** Here redundancy are created in background periodically at a low rate. With this strategy, the bandwidth consumption is predictable and avoid network bandwidth spikes to respond to nodes unavailability [119]. The concern about this strategy is that the system must create redundancy faster than the occurrence of node unavailability. Moreover, it assumes a static system as the rate at which redundancy is created is fixed.
- **Adaptive:** This mode improves the static proactive maintenance strategy. Here the rate at which redundancy is created depends on nodes availability pattern [46]. An adaptive strategy assumes than the nodes availability may vary over time and adapt itself to it contrarily to a static proactive scheme. This strategy requires an effective node availability predictor to work efficiently.

3.2.2.3 Data maintenance and load balancing

As data are stored in the system and storage nodes are added and removed, the location of data may be the right location at time t but at $t + k$ it may not be the right location anymore. Maintaining the data at the right location depends on several factors such as the data access pattern, nodes departures or arrivals, etc. Ideally, all the storage nodes should equally be leveraged in the storage system activity. We can distinguish two operations that are important when dealing with nodes arrivals et departures, load balancing and data migration.

- **Load balancing:** For a better usage of the storage resources in the system, the existing storage resources should share the load of the system taking into account their capacity that may be heterogeneous. For instance, a node having more resources should receive more requests. If the data placement is based on a catalog maintaining the locations of all the data in the system, it is simple to decide where to store data such that the storage nodes resources are evenly distributed among all the nodes taking into account the potential heterogeneity of the nodes. This technique may suffer from scalability problems since the amount of metadata to maintain may be large. On the other hand, data placement based on hashing techniques are more scalable since there require a few amount of information to store regarding the location of data. The information to store are mainly about the nodes in the system and not the data.
- **Data migration:** It can be considered as a side effect of the load balancing operations. In order to evenly share the storage workload amount all the nodes in the system, when some nodes are introduced in the system, some data stored on older nodes may have to be migrated to newer nodes. Similarly when a node is removed from the system, the data stored on this node should be replicated on the remaining nodes. The complexity of the data migration is to find the right schedule to move the data such that regular read and write operations are not impacted by the migration operations.

3.2.3 Data placement

The data placement strategy in a storage system is a crucial point. The location where data are stored in the system can have an impact on several factors such as the performance, the availability, the scalability of the system, or the cost of the storage architecture.

- **Impact on performance:** A data placement strategy that stores data close to their users reduce data access latency. Data access performance may also be increased with techniques such as data stripping where data are splitted in several blocks which are stored or retrieved in parallel on several nodes.
- **Impact on data availability:** A data item and its replica should be located in different places otherwise it does not improve data availability or resiliency since the lost of that node result in the lost of several replicas.
- **Scalability of the system:** A data placement strategy requires metadata. These metadata may concern information about the data, their replica and their locations. Data placement methods that can find quickly where to store or get data while keeping the amount of metadata to maintain low provide a good scalability. Otherwise, it creates a bottleneck in the path to access data and impact negatively the scalability of the system.
- **Storage cost**: All the data do not have the same value. Storage cost optimization techniques such as storage-tiering actually relies on a well thinking data placement on nodes to optimize performance, availability, and recovery while minimizing the cost of the overall storage architecture [6]. For instance storing data that are frequently accessed on expensive nodes with high performance and low storage capacity, and store data that are rarely or less frequently accessed data that not more or less frequently accessed on cheaper node that have

lower performance and high capacity. A knowledge of the data, their usage, and their life-cycle is necessary to implement such solutions. See Section 6.6.1 for more details on storage cost-optimization technique.

In the following we describe some of the most common placement techniques namely, the directory-based, and the pseudo-randomized techniques.

3.2.3.1 Directory-based approaches

These approaches rely on a table or directory to store the metadata regarding the location of every data in the system. That is, the table keeps a list of data identifier and the list of the nodes where they are stored. The data placement method may also keep track of the free space in the system in order to balance the load when allocating new data to storage nodes. This type of technique requires to traverse the metadata directory or table in order to perform I/O operation [30, 19]. Thus performance may decline as the amount of metadata increases reducing system scalability, availability, and performance [125, 142].

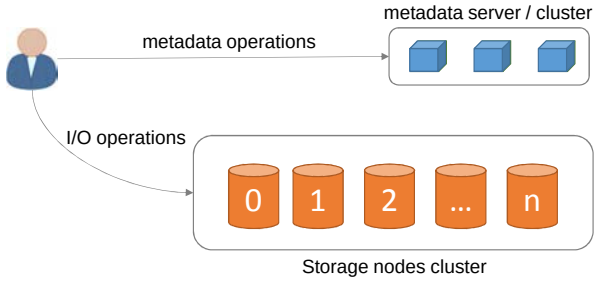


Figure 3.1: Example of a directory based data placement.

3.2.3.2 Pseudo-randomize approaches

Pseudo-randomize approaches aim to offer a better alternative to a directory or table based data placement in terms of performance and scalability. They do not rely on a table containing the location of all the data stored in the system. They usually require that each node and data item in the system be assigned a unique identifier in the same namespace. A node is responsible of one or several portions or ranges in the system namespace. In order to find where to read or write a data item, a function (usually a hash function) taking into parameters the data item identifier and the list of nodes in the system is used. This function, maps the object identifier to a point in the namespace, the storage node responsible to portion of the namespace containing this point will be associated to the data item [72, 25, 137, 90, 67].

Figure 3.2 illustrates a pseudo-randomize placement method based on the modulo function. For instance for an object with an identifier 7 and n the number of nodes been equal to 5, the object will be stored and read on the node with the identifier 2 because $7\%5 = 2$. Here, the amount of metadata to maintain in the system depends on the number of nodes in the system, not on the amount of data items or data blocks to store.

While the method illustrated in Figure 3.2 helps to catch the concept of pseudo-randomize data placement methods, it contains several flaws. In the following we discuss these flaws and how advanced pseudo-randomize techniques deal with it.

Efficient migration: In Figure 3.2, when adding or removing storage nodes, a large amount of data have to be migrated to others nodes to keep consistent the lookup mechanism. Depending on the amount of existing data and the workload, migration can have a negative impact on performance

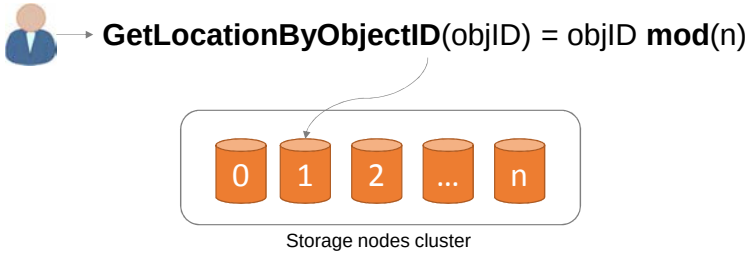


Figure 3.2: Example of a pseudo-randomize data placement.

and availability of the system [61, 116]. Advanced pseudo-randomize techniques try to minimize the amount of data that have to be re-located in case of node arrival or removal [25, 90, 67, 136]. Ideally the amount of data to migrate upon nodes arrival or removal should be equal to m/n with m the overall amount of data and the n the total number of nodes in the system considering all nodes having the same capacity.

Load balancing and heterogeneity support: In Figure 3.2, the function assumes that all the nodes are equal which is not necessary the case. Some nodes may have more resources (storage, compute, memory, etc), other nodes may be new in the system and may have more free space. A placement method taking into account the heterogeneity of the nodes will try to keep balanced the load among all the storage nodes in the system proportionally to their capabilities. Note that to keep a system well balanced, data migration is needed in the case of nodes arrivals and departures. To achieve it, in some methods, depending on their capabilities, storage nodes are responsible of one or several portions or ranges of the namespace proportionally to its capacity [121, 90]. Depending on the placement method, the number of ranges or portions of the namespace a node is responsible of may vary upon nodes arrivals and departures.

Redundancy support: In Figure 3.2, a data item and its replica will be stored on the same node canceling all the benefits of data redundancy. More complex and efficient pseudo-randomize data placement methods ensure that a data item and its replica will be stored on different nodes. In certain systems, mostly Peer-to-Peer systems [84, 8], to achieve it, a node maintains a list of other nodes that are its *neighbors* or *successors* and the system allows a node to store on its neighbors the replica of the data it is responsible of. An other strategy is to store replica using different keys/identifiers (*e.g.* using the data item and a replica index) so that they can be stored by different nodes. Other systems use the notion of placement groups sometimes called replication groups or redundancy groups [136, 142, 145]. In this strategy data items are associated to a placement group instead of being associated to a node. The data item will be replicated on all the nodes of the placement group.

Reverse lookup: In Figure 3.2 and in most of pseudo-randomize data placement methods, the lookup function, usually a hash function, is non-reversible *i.e.* knowing the output (the storage node identifier) of the function, you can not recover the input (*i.e.* the data item identifier). When a node is removed (*e.g.* because of a crash), the redundancy level of data stored on such nodes is impacted so it may be relevant to create additional replicas. Finding what data are impacted by the removal of a node is a process sometimes called *reverse lookup* [142, 145]. If the location of all data stored in the system is not maintained in a centralized place like in directory-based method, it is not simple to get the list of data that were stored in the removed node. The placement group notion (See the previous paragraph on *redundancy support*) helps to achieve the reverse lookup while maintaining low the amount of location metadata to maintain (usually the list of nodes and their placement groups) [136, 142, 145]. In fact because a data item is replicated on all the nodes

of the placement group, when a node is removed, the list of data that were stored on that node can be retrieved using the list of data stored on the remaining nodes of the placement groups the removed node was part of.

Table 3.1 shows a comparison of some placement technique regarding the previous characteristics.

	Efficient migration	**Load balancing and heterogeneity support**	**Redundancy support**	**Reverse lookup**
Directory-based methods [30, 19]	Possible through nodes capacity and data monitoring	Possible through nodes capacity and data monitoring	Yes	Through metadata traversal
Consistent hashing [72]	Migration is done only among the neighbors of the new/removed node. The load is not share by all the nodes of the system	Load balancing only for homogeneous nodes. Systems using consistent hashing like Chord [121] provide heterogeneity using virtual node techniques	Can be provided by replicating data among neighbors or by storing replica using different keys (e.g. using the data item and a replica index) so that they can be stored by different nodes	Coordination between nodes storing replica.
CRUSH [137]	yes	yes	Managed via a placement group technique in Ceph[136], the system using CRUSH	Managed via a placement group technique in Ceph, the system using CRUSH
Random slice [90]	yes	yes	We are not aware of any system using the random slice placement technique. However reverse lookup can be added using placement group technique	We are not aware of any system using the random slice placement technique. However reverse lookup can be added using placement group technique
Asura [67]	yes	yes	We are not aware of any system using the Asura placement technique. However redundancy can be added using placement group technique	We are not aware of any system using the Asura placement technique. However reverse lookup can be added using placement group technique

Table 3.1: Comparison of some placement and maintenance algorithms.

3.3 Data consistency

In a distributed system where data are mutable, because of the data redundancy mechanism, it is possible to observe at a specific time different versions of the same data. For example, when only a subset of the replica of an existing data item have been updated. In this situation, if several clients perform concurrently a read operation on the same data item, they may receive different versions of that data. In such a distributed system, designers have to define what we call the data consistency provided in the system.

We can define the data consistency as the guaranty of the freshness of the data that we get upon a read operations of a data. If the system ensures that a read operation always returns the latest version of data we say that this system provides a strong consistency. Otherwise, the system provides a weak consistency. There exists various consistency criteria from strong consistency to weak consistency.

In cloud storage systems the consistency criteria that is the most adopted is the eventually consistency, a form of weak consistency, which has also several variations. This consistency criteria ensures that if no updates (*i.e.* write operations) are made on a data item, eventually all the read operations will return the last updated version of that data item. The reason why eventually consistency is widely adopted is due to the fact that strong consistency can be costly to achieve in terms of performance reducing the scalability of the systems if there is a lot of write and read operations that are concurrent on the same data. This leads to different tradeoffs that designers have to face regarding data consistency management.

3.3.1 Tradeoffs in data consistency management

Due to the cost of providing data consistency in a distributed system, system designers usually face some tradeofs regarding data consistency management. These tradeofs are mainly related to the guaranty of providing strong consistency, availability, performance, and network partition tolerance. These tradeofs are commonly expressed by the CAP theorem. In this section we describe the CAP theorem and also the PACELC approach which is a other way to consider the tradeofs in data management consistency.

3.3.1.1 The CAP Theorem

CAP stands for Consistency, Availability, and Partitioning:

- **Consistency:** The ability of the system to always provide strong consistency.
- **Availability:** The ability of the system to never exhibit downtime even in the presence of some nodes unavailability (*i.e.* read and write operation always succeed).
- **Partitioning:** The ability of the system to tolerate network partition and arbitrary messages loss. For example if the network is splitted in two disjoint subsets, in each subset the system is still able to respond to clients requests.

The CAP theorem has been first stated by Eric Brewer at the PODC 2000 keynote. It has been formally proved in 2002 by Nancy Lynch [54]. It states that when building a distributed system, it is impossible to provide at the same time strong consistency guaranty, availability, and network partition tolerance. Only two among the three properties can be provided by a distributed system (See Figure 3.3). The mostly adopted idea is that in any distributed system, partitions may occur so it is unavoidable. So a distributed system must tolerate partitioning, it leads system designers to choose between consistency and availability. However many cloud systems put a high priority in availability making several cloud system designers sacrifice data consistency.

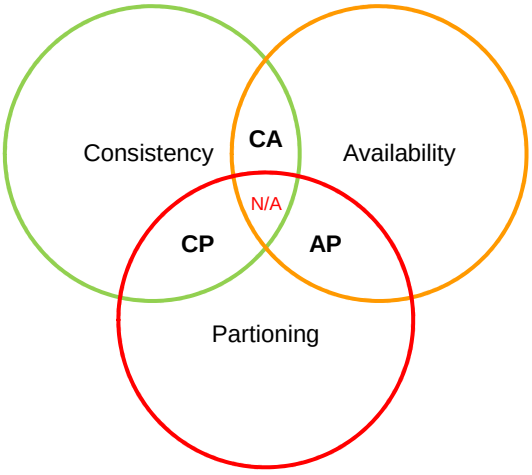


Figure 3.3: The CAP Theorem.

3.3.1.2 The approach PACELC

The approach PACELC [2] comes from the observation that considering the CAP theorem, many system designers automatically sacrifice strong consistency when building their systems. This approach suggests to think about the CAP theorem in the way that availability and consistency can not be achieved in the presence of network partition. The same suggestion has also been made by Eric Brewer in 2012 [20]. This approach considers that when there is no network partitions, the tradeoff becomes between consistency and latency (See Figure 3.4). In fact ensuring strong consistency introduces latencies due to read and writes requests synchronization. The choice may be to relax consistency to reduce the latencies. This approach states that the tradeoff between latency and consistency is to consider when there is no partition. If there is network partitions, the CAP theorem is then applied.

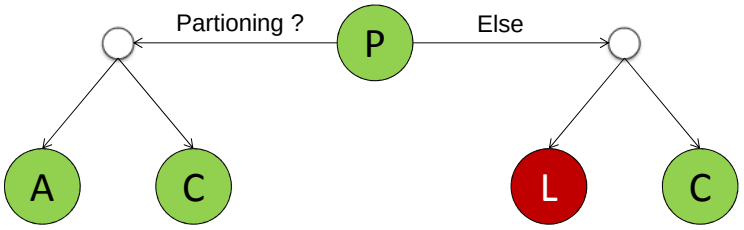


Figure 3.4: The approach PACELC: In the presence of network partitions, the CAP theorem is applied. Otherwise, the trade-off should be between latency and consistency.

3.3.2 Emerging trends in data consistency

Due to the high cost of providing strong consistency criteria, several storage systems choose to provide only the eventual consistency model [134]. However, even if it is sufficient for a large panel of applications some others require strong consistency criteria [36, 74]. In fact, a strong consistency is a desirable property for application programmers as it is easy to reason about.

As the consideration of the tradeoffs in data consistency evolves [20, 2, 127], system designers tend to consider approaches that are different than just provide the eventual consistency model. Among these approaches, we can mention the following one:

- **Static consistency selection:** In this case, the system is able to provide several consistency models [29, 102]. This strategy has been introduced in several well known systems such as Cassandra[74] or DynamoDB[36]. However the targeted consistency model is fixed.
- **Adaptive and dynamic consistency selection:** In this approach, the idea is to change dynamically the consistency model in order to provide strong consistency when necessary and relax consistency criteria to meet performance SLAs depending on parameters such as the access behavior, etc.
- **Consistency using conflict-free replicated data type:** This approach relies on the conception of data structures that can support concurrent requests on the same data and still maintain consistency without concurrency control. A pioneer of this approach is the Commutative Replicated Data Type (CRDTs) [104] which a data type in which the operations on the same data commute when they are concurrent.

3.4 Data confidentiality

While the adoption of cloud storage services to outsource data is steadily growing, its benefits come with some inherent risks implied by the fact that full control on data is given to the CSPs [71, 107, 109]. Clients (*e.g.*, individuals, scientists, enterprises, governmental agencies, etc) may want to restrict the control or access of the CSP to their sensitive data in order to preserve some confidentiality.

3.4.1 Basic concepts

Data confidentiality in cloud storage systems is usually achieved using cryptographic methods that transform a plaintext file into a ciphertext file, using a specific key. The result of this transformation is incomprehensible in order to preserve the confidentiality of the plaintext file. This is called an encryption. Another specific key is required to transform the ciphertext back into the comprehensible original plaintext file. Recovering the plaintext file is called decryption. The security level, and thus the confidentiality, is based on the difficulty to decrypt a file without the proper key. There are two basic cryptographic methods to achieve encryption, the symmetric (also known as secret key encryption) and the asymmetric (also known as public key encryption) [70, 118].

Symmetric encryption

The same key is used to both encrypt and decrypt data. This scheme is generally faster than the asymmetric encryption schemes and consumes less processing resources. However sharing the encryption keys is not trivial. For example, when a user Alice wants to share an encrypted content with another user Bob, Alice must share the key with Bob so that he will be able to decrypt the content. Sharing an encryption key with another user must be done through a secure communication channels, which can be difficult to achieve. This scheme has strengths and weaknesses. In fact different keys may be used for different files, so when a key is compromised, only the files encrypted with this key are compromised. Different keys can also be used for a same file when sharing it with different users. Moreover encryption keys may have to be renewed periodically. So as the number of keys grows, managing and securing all the keys become cumbersome.

In 1997 the National Institute of Standards and Technology (NIST) launched a competition about symmetric encryption algorithms. In 2000, among 15 propositions, the Advanced Encryption Standard (AES) algorithm [33] is declared the winner. This algorithm has been standardized in 2000. To encrypt data, the AES algorithm operates on the data blocks, one at a time. Data blocks of 128 bits long are scrambled in multiple mathematical processes using a key that may be 128, 192, or 256 bits long. In order to break an AES key of length k, a brute force attack will take $2k$ trials. At this time, there is no known efficient attack on the AES algorithm.

Asymmetric encryption

This scheme relies on two different keys to encrypt and decrypt contents. A public key is used to encrypt contents and a private key to decrypt them. The owner of the pair of keys must keep the private key secret, while the public key can be publicly available. This scheme solves the encryption keys sharing issue of symmetric encryption. In fact, only the private key (*i.e.* the decryption key) has to be kept secret. For example, when Alice wants to share a file with Bob, Alice just needs to use the public key of Bob which is publicly available to encrypt the file and send the encrypted file to Bob. Thus, the only one able to decrypt the file is Bob using his private key. Managing keys is also less cumbersome with this encryption scheme, as it requires only a few keys to manage. However, the security advantages of the asymmetric encryption schemes come with price. Due to the complex arithmetic computations used in the asymmetric encryption, they are slower than symmetric encryption.

The RSA asymmetric encryption algorithm [69] is a well-known standard of asymmetric encryption schemes. The NIST recommends using a RSA algorithm with public keys that are 2048 bits long. Keys that are 1024 bits long were recommended until 2010. Some researchers who succeeded in breaking a RSA-768 bits long in 2010 say that breaking a RSA-1024 bits long would be thousand times harder and that it could take a decade before one can break a RSA-1024.

While the first option is to increment the keys length to get security of the RSA asymmetric encryption for more additional years, other efforts are made to use the Elliptic Curve Cryptography (ECC) that can bring an equivalent (or better) security level while having shorter keys. The ECC has been standardized in 2006 by the NIST.

Combine asymmetric and symmetric encryption

Many systems combine these two basic encryption schemes to take advantage of their respective benefits. It allows a good security performance tradeoff. The common use case is to encrypt/decrypt data with a symmetric encryption schemes and use an asymmetric encryption scheme to protect and share the encryption keys.

Key management

While the encryption schemes protect data with encryption keys, a key management system protects the encryption keys themselves [4]. In fact, their safety is crucial as it is usually easier for attackers to steal encryption keys than to break cryptographic algorithms. Losing encryption keys may also be problematic as, without the proper key, one will not be able to decrypt his or her own data anymore. A key management system aims to securely store, backup, organize, control the access to the keys, and recover encryption keys.

When it comes to store encrypted data in the cloud, there are usually three models to manage the encryption keys depending on the trust we can grant to a third party.

1. First encryption keys are kept in-house. For instance a company may use its own datacenters to manage their encryption keys.

2. Second an external trusted key server may be used to manage the key on behalf of the cloud storage service provider.

3. Third, the encryption may also be stored in a cloud but this brings the same confidentiality issues encountered on data confidentiality.

3.5 Summary

In this section, we review the basic of dependability and the property that are the most relevant for the understanding of this thesis. Dependability is really important as it can impact economically the users and also for the providers. In the contributions of this thesis, efforts has been made to provide a scalable distributed architecture that can reduce the data access latency, ensure data availability through redundancy, and a data consistency management that allow service system designers the ability to choose at runtime the level of consistency of a read operation (See Chapter 4). We propose a scalable data placement scheme aiming to reduce the amount of data location metadata to maintain and giving the flexibility to schedule data migration of configure data redundancy (See Chapter 5). We also address data security and confidentiality in Chapter 6.

4

Mistore: a distributed storage system leveraging the DSL infrastructure of an ISP

Internet Service Providers (ISPs) furnishing cloud storage services usually rely on big datacenters. These centralized architectures induce many drawbacks in terms of scalability, reliability, and high access latency as datacenters are single points of failure and are not necessarily located close to the users. In this chapter we present Mistore, a distributed storage system that we designed to ensure data availability, durability, low access latency by leveraging the Digital Subscriber Line (xDSL) infrastructure of an ISP. Mistore uses the available storage resources of a large number of home gateways and points of presence for content storage and caching facilities. Mistore also targets data consistency by providing multiple types of consistency criteria on content and a versioning system. Mistore validation has been achieved through extensive simulations.

Contents

4.1 Introduction

Most of the existing architectures for data storage of Internet Service Providers (ISPs) are based on very large centralized datacenters that store and manage all the information related to their clients and their data. With the ever growing number of clients and the amount of data, centralized architectures reach their limit in terms of scalability, reliability, and access latencies. In fact, if the datacenter becomes unavailable due to an outage or an overload, all the data stored in this datacenter become unavailable which may cause clients frustration and have an negative impact on the economy of the Cloud storage provider (CSP). Data access latencies may also be high because datacenters are not necessary located close to the users.

These drawbacks are commonly addressed through the system decentralization over multiple geo-distributed nodes. Data are then placed on nodes close to users to reduce the data access latencies and improve the system scalability and reliability by eliminating the single point of failure issue. Usually storage systems addressing these issues rely on Peer to Peer (P2P) technologies using unreliable peers or reliable nodes like datacenters [83, 36]. The main limitation of using peers is their low availability that makes systems suffer from the churn problem (*i.e.*, frequent arrivals and departures of peers) resulting in an expensive system maintenance [17]. On the other hand, in systems relying on reliable datacenters, data are still not necessarily located close to users. Hybrid approaches relying on both peers and datacenters have been introduced in which the peers reduce datacenters workload and data access latencies, and improve system scalability while datacenters compensate peers instability [129, 138].

In this work, we investigate the design of a hybrid distributed storage system by leveraging highly available nodes in the network infrastructure of an ISP. Specifically, our approach combines the use of datacenters with home gateways and Points of Presence (POPs) equipments. In the following, we first briefly describe the Digital Subscriber Line (DSL) infrastructure of an ISP and then we summarize our contributions.

4.1.1 Overview of a DSL infrastructure

Figure 6.13 illustrates a simple but common network infrastructure providing access to Internet through a DSL technology. Each subscriber is equipped with a home gateway, that provides access to multiple services like telephone, television, and Internet. Each line of subscribers is first aggregated by a network equipment called Digital Subscriber Line Access Multiplexer (DSLAM) that can aggregate thousands of DSL lines. These subscriber lines are aggregated in a high-capacity Asynchronous Transfer Mode (ATM) or Gigabit Ethernet link from a DSLAM to the IP core network of the ISP. The flow coming from a DSLAM enters the ISP core network via a Point of Presence (POP) that can handle links from a large number of DSLAMs. The ISP core network is directly connected to the Internet backbone, and uses large datacenters to store information related to their subscribers and their data when a storage service is provided.

4.1.2 Contributions

Based on the classic xDSL infrastructure described in Section 4.1.1, we aim at:

- **leveraging Home Gateways (HGs)** to take advantage of their native computing, memory and storage resources[1], and to take advantage of the fact that users let most of the time their HGs powered on as shown in [133, 37]. Thus, exploiting resources provided by HGs should yield a large number of high available and intelligent storage nodes located very close to the users.

[1]For instance the livebox play (http://liveboxplay.orange.fr/) of the french ISP Orange contains a 1.2GHz Intel Atom CE4257 processor, 2GB of RAM, and a hard drive of 320GB.

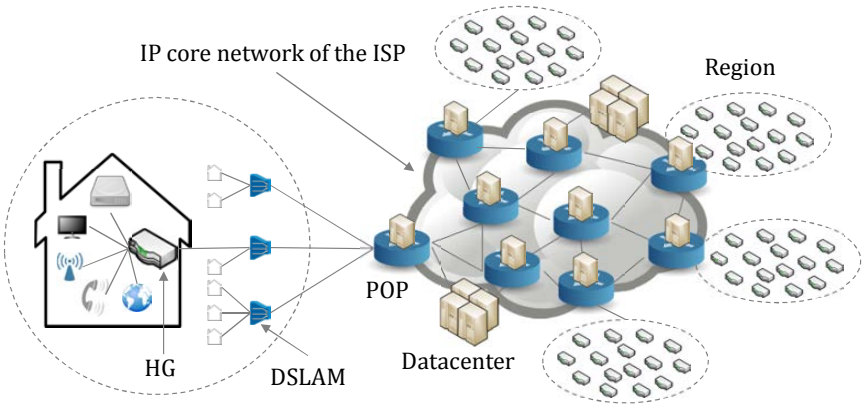


Figure 4.1: Simple overview of a DSL infrastructure of an ISP.

- **leveraging Points of Presence (POPs)** to benefit from their natural geographic repartition and their position at the edge of the ISP network, and the fact that all the traffic within and across ISPs goes through the POPs. Thus, it is interesting to make them provide new services by leveraging these benefits. For instance, they are leveraged, in some cases assisted with peers, for content caching and distribution in Content Delivery Networks (CDNs) [65, 99, 100]. We aim to leverage the POPs also to bring the intelligence of the storage system close to users.

We present Mistore, a distributed storage system dedicated to users who access Internet via a DSL technology that ensures data availability, durability, and low access latency by fully exploiting HGs storage capacities, POPs physical localizations, and datacenters. The focus of this chapter is on how data consistency and replication are managed in Mistore.

- **Data consistency:** System designers usually face the CAP theorem (trade-offs about Consistency, Availability, and network Partition tolerance) when designing a distributed storage system [134, 2]. We choose to serialize the write requests in order to avoid the complexity of conflicts resolution to application developers. On the other hand multiple consistency criteria to parametrize the read requests are provided, namely a readers/writer mutual exclusion, an atomic consistency, and an eventual monotonic-read consistency criteria. We also provide a versioning system on data. This mechanism creates a new version of a data item when an update (*i.e.* a write request) is performed on it, and naturally provides a linear history about data and their updates.

- **Data availability and durability:** We ensure data availability and durability through replication. Users send their write requests to the closest POP that will perform the data replication on behalf of them. It aims to mitigate the bottleneck that users low upload bandwidths may cause by moving the replication overhead to the IP core network of the ISP where the bandwidth is much larger. To reduce latencies data are replicated synchronously on POPs and asynchronously on HGs and stripping techniques can be used to store and retrieve the data fragments composing data items in parallel.

The remainder of this chapter is organized as follows. Section 4.2 presents the system model and the assumptions considered. The architecture is presented in section 4.3. The different consistency criteria are defined in Section 4.4 and the way we implement it as well as read and write operations is described in Section 4.5. Section 4.6 presents the evaluation of our design choices. Section 4.7 presents related works and we conclude in Section 4.8.

4.2 System model

Mistore targets personal data (*e.g.*, documents, pictures, videos, music, etc). In the following, these data will be called *objects*. Communications are assumed to be reliable, and messages between two nodes are assumed to be received in the order they have been sent. POPs are interconnected via highly available and redundant channels provided by the ISP. Network partitions at this level occur with a negligible probability as the IP core network can recover and restore its state to avoid service disruption due to links/nodes failures [50]. So we assume a system with no network partitions between the POPs. We assume different failures models for POPs, HGs, and the datacenter. The model of failure for POPs and datacenter is crash recovery, *i.e.*, we suppose that they both recover from a consistent state by using stable storage. Conversely, the model of failures exhibited by HGs is fail stop. We assume that crash are eventually detected by an eventual perfect detector failure [24]. We leave for future work more aggressive failures, Byzantine failures and objects corruption.

4.3 System architecture

We consider a DSL infrastructure operated by a single ISP (See Figure 6.13), whose main components are as follows.

- **The home gateways (HGs)** are connected in a client/server mode to the POP aggregating their DSL lines and form its *region*. They communicate with the storage system via this POP. A fraction of the storage capacity of HGs is dedicated to the storage system. One part of this fraction stores some of the data that have been warehoused within the storage system. Note that these data do not necessarily belong to the owner of the HG. The other part of the fraction caches the data recently or frequently accessed by the HG owner.

- **The Points of Presence (POPs)** provide several functionalities. They cache some data that transit through them, implement the data consistency and replication strategy of the storage system, and monitor the HGs they aggregate. Monitoring data (*e.g.*, storage capacity, data stored, availability state, etc.) are periodically transferred to datacenters.

- **The datacenter (DCs)** main role is to collect and store metadata and usage patterns from the HGs and POPs. It also offers a POP backup functionality for objects when their primary POP is unavailable (See Section 4.4).

4.4 Object consistency

Mistore replicates user objects to guarantee both their durability and availability. Since, as objects can be updated, consistency issues must be taken into account. Before diving into the different consistency criteria provided by Mistore, we briefly recall the main concepts and definitions that we will use. Traditionally consistency criteria are classified according to the fact that, given a distributed execution satisfying the considered criteria, it is always possible or not to build an history of this execution that could have been executed on a single processor and produce the same visible result. When this property is satisfied, the considered consistency criteria is said to be *strong*, otherwise it is *weak*. A strong consistency criteria requires stronger synchronization between processes that participate to the computation. This impacts the performance of the protocol but makes the life of application developers easier. One can expect better performance of weak consistency criteria at the cost of more complex situations to deal with when manipulating data due to to conflicts that may happen. Mistore implements both strong and weak consistency criteria. Hereafter, we rapidly expose the theory behind strong consistency criteria [106], and give a formal treatment of the weak consistency criteria implemented by Mistore.

4.4.1 Processes and operations

We consider a *concurrent system* composed of a set Π of n processes denoted $p_1, p_2, \ldots, p_n$ that cooperate through a finite set of *shared objects* X (*e.g.*, files in the case of Mistore). Each shared object $x \in X$ supports two types of operations: a `read` operation, and a `write` operation. The execution of writing value v into object x by process p_i is denoted by $w_i(x)v$. Conversely, the execution of reading value v from object x by process p_i is denoted $r_i(x)v$. We may omit the identity of the process that performs the operation when it is not relevant. To simplify we assume that each value written in an object is unique.[2]

4.4.2 History, legality and linear extension

Let $h_{i|_W}$ denotes the set of `write` operations performed by process p_i. Conversely let $h_{i|_R}$ denotes the set of `read` operations performed by process p_i. Let $h_i = h_{i|_R} \cup h_{i|_W}$ denotes the set of operations (both `read` and `write`) performed by process p_i. We assume that on a given process p_i only a single operation can occur at a given time. Hence operations in h_i are naturally totally ordered by an order relation that we denote $\rightarrow_i$. We call *local history* of p_i the set h_i ordered by $\rightarrow_i$. It is denoted by $\widehat{h_i} = (h_i, \rightarrow_i)$.

Definition 1 (Global history) *A global history $\widehat{H} = (H, \rightarrow_H)$ of a concurrent system (Π, X) is a set H partially ordered by $\rightarrow_H$, with $H = \bigcup_i h_i$ and $\rightarrow_H$ a partial order relation containing $\rightarrow_{rf}$ (that is $\rightarrow_{rf} \subseteq \rightarrow_H$), where $\rightarrow_{rf}$ is a partial order relation called read from order and defined as follows. For any two operations $op_1 \in H$ and $op_2 \in H$, we say that $op_1 \rightarrow_{rf} op_2$ if and only if*

1. *either $\exists i$ such that $op_1 \in h_i$ and $op_2 \in h_i$ and $op_1 \rightarrow_i op_2$,*
2. *or $\exists x, v$ such that $op_1 = w(x)v$ and $op_2 = r(x)v$,*
3. *or $\exists op_3 \in H$ such that $op_1 \rightarrow_{rf} op_3$ and $op_3 \rightarrow_{rf} op_2$.*

Definition 2 (Linear extension of a global history) *Given a global history $\widehat{H} = (H, \rightarrow_H)$, a linear extension of $\widehat{H}$ and denoted $\vec{H} = (H, \rightarrow)$ is a total order $\rightarrow$ on H that is compatible with $\rightarrow_H$, that is for any two operations $op_1 \in H$ and $op_2 \in H$ if $op_1 \rightarrow_H op_2$ then $op_1 \rightarrow op_2$.*

A linear extension of a global history $\widehat{H}$ can be seen as a topological sort of the directed acyclic graph[3] induced by the partial order relation $\rightarrow_H$.

We assume that an initial value has been written (by a fictitious `write` operation) in each variable. With this assumption, a linear history is *legal* if the following holds.

Definition 3 (Legality of a read) *Let $\vec{H} = (H, \rightarrow)$ be a linear extension. A `read` operation $op_r = r(x)v \in H$ is legal if and only if:*

1. *there exists a corresponding `write` operation $op_w = w(x)v \in H$ such that $op_w \rightarrow op_r$,*
2. *and for all operations op such that $op_w \rightarrow op \rightarrow op_r$ $op \neq w(x)\star$.*

Definition 4 (Legality of a linear extension) *A linear extension $\vec{H} = (H, \rightarrow)$ is legal if and only if all of its `read` operations are legal.*

A linear history is legal when both its `read` and `write` operations respect the expected semantics of variables on a single threaded processor: a `read` operation on a variable returns the last value written in this variable.

[2] This hypothesis can be easily implemented by assuming that each value written in an object is a triplet of the form of $(p_i, v, count_i)$ where $count_i$ is a counter maintained by process p_i counting its `write` operations.

[3] This graph $G = (V, E)$ is defined by its vertexes $V = H$ and there is a edge between two vertices $op_1 \in V$ and $op_2 \in V$ if and only if $op_1 \rightarrow_H op_2$.

4.4.3 Strong consistency criteria

Definition 5 (Sequential consistency) *A global history $H = (H, \rightarrow_H)$ is sequentially consistent if it admits a linear extension which is legal.*

This consistency criterion is the simplest and weakest example of strong consistency criteria. It means that the concurrent execution could have happened on a single processor. This criteria has been first proposed by Lamport in [75].

We now present a stronger consistency criteria which brings into play physical real time. Each operation *op* happening on a process starts at a given real time denoted by *op.start* and ends at a given real time denoted by *op.end* (we have $op.start < op.end$). With these notations in hand, we can specify the partial order relation denoted by $\rightarrow_{rt}$ as follows.

Definition 6 (Real time precedence) *Let H be a set of operations, and let $op_1, op_2 \in H$ be any two operations. We say that op_1 precedes op_2 with respect to real time, which is denoted by $op_1 \rightarrow_{rt} op_2$, if and only if $op_1.end \leq op_2.start$.*

Definition 7 (Real time concurrency) *We say that two operations op_1 and op_2 are real-time concurrent, which will be denoted by $op_1 \|_{rt} op_2$, if and only if neither $op_1 \rightarrow_{rt} op_2$, nor $op_2 \rightarrow_{rt} op_1$.*

Definition 8 (Atomicity) *A global history $\widehat{H} = (H, \rightarrow_H)$ is atomically consistent if it admits a linear extension $\vec{H} = (H, \rightarrow)$ such that:*

1. *$\vec{H}$ is sequentially consistent,*
2. *and $\rightarrow_{rt} \subseteq \rightarrow$ (i.e $\rightarrow$ is compatible with $\rightarrow_{rt}$).*

Definition 9 (Read/write order) *Let H be a set of operations, and let $op_1, op_2 \in H$ be any two operations. We say that op_1 precedes op_2 with respect to the **read/write order**, which will be denoted by $op_1 \rightarrow_{rw} op_2$, if and only if:*

- *either $op_1 \rightarrow_{rt} op_2$,*
- *or $op_1 \|_{rt} op_2$, $op_1 = r(x)\star$ and $op_2 = w(x)\star$, then*
  1. *$op_1 \rightarrow_{rw} op_2 \Leftrightarrow op_1.start \leq op_2.start$,*
  2. *$op_2 \rightarrow_{rw} op_1 \Leftrightarrow op_2.start < op_1.start$.*

It is clear that $\rightarrow_{rt} \subseteq \rightarrow_{rw}$.

Definition 10 (Read/write mutual exclusion) *A global history $\widehat{H} = (H, \rightarrow_H)$ is compatible with the **read-write mutual exclusion** consistency criteria if it admits a linear extension $\vec{H} = (H, \rightarrow)$ such that $\rightarrow_{rw} \subseteq \rightarrow$.*

4.4.4 Weak consistency criterion

From these criteria, we propose to define a weak consistency criteria, that we will call **monotonic read** consistency. To the best of our knowledge it is the first time that a weak criteria is defined in the same formalism framework as strong ones.

Definition 11 (Local history) *Let $\widehat{H} = (H, \rightarrow_H)$ be a global history. A local history $(H_i, \rightarrow_{H_i})$ with respect to process p_i is defined as follows :*

- $H_i = h_i \cup \left(\bigcup_{j \neq i} h_{j|_W} \right)$,

- $\rightarrow_{H_i}$ *is compatible with* $\rightarrow_H$ *: for any operations* op_1 *and* op_2 *in* H_i*, we have* $op_1 \rightarrow_i op_2 \Leftrightarrow op_1 \rightarrow op_2$*.*

Definition 12 (Read monotonic local history) *Let* $\widehat{H} = (H, \rightarrow_H)$ *be a global history.* $\widehat{H}$ *is consistent with the* ***read monotonic*** *consistency criteria if and only if for all processes* $p_i \in \Pi$*, the local history* $(H_i, \rightarrow_{H_i})$ *is* ***sequentially*** *consistent.*

4.5 Implementation

Mistore implements the different levels of consistency described in Section 4.4 through a lock service. Two types of locks exist. A *read lock* that ensures the mutual exclusion consistency criteria, and a *write lock* that ensures the Single Writer Multiple Reader (SWMR) model.

The lock service follows a primary-backup scheme [7]. A primary and backup lock server per object is implemented and are executed by the POPs.

Algorithm 1 and Algorithm 2 show the pseudo-code executed by HGs and POPs to acquire a lock from respectively their associated POP and the primary POP of the concerned object. The lock on an object is created by function GRANTLOCK shown in Algorithm 3. Function ISAVAILABLE checks if a requested lock on a specific object can be granted to a HG. To deal with nodes failures and deadlocks, a *lease* is associated with each granted lock [58]. At *lease time*, that is, at the expiration time of the lock, a client has to renew the lease if the current read/write operation is not completed. This is achieved by contacting the primary in charge of the object. Note that for communication costs reasons, the lease time is only activated on the primary, not on the backup. Thus, the primary does not have to acknowledge the backup each time a lock lease is renewed. The backup activates the lease time of a lock only when it takes over this role upon detection of the primary failure.

Algorithm 1 Home gateway: Function ACQUIRELOCK.
acquireLock(o_{id}, lockType)

Input: The identifier of the object o and the desired lock type
Output: The granted lock.

```
1: myPOP ← GETMYPOP()
2: lock ← myPOP.ACQUIRELOCK(selfID, o_id, lockType)
3: return lock
```

Algorithm 2 Point of Presence: Function ACQUIRELOCK.
acquireLock(hg_{id}, o_{id}, lockType)

Input: The HG identifier, the identifier of the object o, and the desired lock type
Output: The granted lock.

```
1: if self.ISPRIMARYOF(o_id) then
2:     lock ← GRANTLOCK(hg_id, o_id, lockType)
3: else
4:     primary ← GETPRIMARYOF(o_id)
5:     lock ← primary.ACQUIRELOCK(hg_id, o_id, lockType)
6: end if
7: return lock
```

4.5.1 Write operation

A write operation on an object creates a new version of this object in Mistore. When a client wants to write (update) an object o, it sends a request to the POP p associated to his HG as presented in Algorithm 4 and Algorithm 5. If o is written for the first time, a primary and a backup POP are

Algorithm 3 Point of Presence : Function GRANTLOCK.
grantLock(hg_{id}, o_{id}, lockType)

Input: The HG identifier, the identifier of the object o, and the desired lock type
Output: The granted lock.

```
1: lock ← null
2: if typeOf(lockType) = readLock then
3:    if ¬ ISAVAILABLE(hg_id, o_id, writeLock) then
4:       lock ← INCREMENTLOCK(hg_id, o_id, readLock)
5:    end if
6: else
7:    if ISAVAILABLE(hg_id, o_id, lockType) then
8:       lock ← LOCK(hg_id, o_id, lockType)
9:    end if
10: end if
11: return lock
```

affected to it (See Algorithm 6). These POPs are responsible for the replication of that object and for its consistency management. The primary is the POP associated with the client HG issuing the creation request. The backup is randomly determined by the primary among the other POPs of the system (invocation of the GETBACKUP() function in Algorithm 6). The create operation returns o_{id} the identifier of an object o, which is the concatenation of the identifiers of o primary, o backup, o HG, and a creation counter that represents the number of objects created by the HG that owns o. In the following the primary and the backup of object o are respectively denoted by o_p and o_b. When the primary of an object is unavailable, its backup becomes the primary and the datacenter becomes the new backup. When the primary recovers, it takes over its role. Our solution only makes use of one backup because of the high availability of these nodes [56]. A write operation on object on o that has already been created in the system must contain the write lock for o to respect the SWMR model. Note that as a client can crash or goes off before the end of a write operation, the write lock on o is maintained by POP p as long as the operation is not completed to ensure the SWMR model. POP p locally caches a copy of o and asks both the primary o_p and the backup o_b to replicate o. When o_p and o_b receive this request, they both locally cache o, acknowledge POP p (as described in Algorithm 7), and trigger the replication operation on their HGs (see Lines 3–7 in Algorithm 7, and Algorithm 8). Upon receipt of both acknowledgements, POP p releases the write lock and notifies the client that the write operation is completed. Object o is then available for subsequent read and write operations, which makes the replication process on HGs transparent to clients.

Algorithm 4 Home gateway: Function WRITE.
write(hg_{id}, o_{id}, writeLock, counter, o)

Input: The HG identifier, the identifier of the object o, a write lock, a creation counter, and the object o.
Output: An acknowledgment containing the object identifier and its new version number.

```
1: myPOP ← self.GETPOP()
2: writeAck ← myPOP.WRITE(hg_id, o_id, writeLock, counter)
3: UPDATECACHE(writeAck.o_id, writeAck.o_version, o)
4: return writeAck
```

4.5.2 Read operation

Mistore gives the opportunity to specify the consistency criteria that the read object must guarantee, or a specific version of desired object. Both choices are specified in the *version flag* parameter of the read operation. In addition, in the quest of reducing the latency of read operations, Mistore favors readings from local caches prior to propagating the request to both the primary and the backup of the object. Read operations handle two parameters, the identifier o_{id} of the requested

Algorithm 5 Point of Presence: Function `WRITE`.
write(hg_{id}, o_{id}, writeLock, counter, o)

Input: The HG identifier, the identifier of the object `o`, a write lock, a creation counter, and the object `o`.
Output: An acknowledgment containing the object identifier and its new version number.

1: **if** ¬writeLock **then**
2: $o_{id} \leftarrow$ CREATE(hg_{id}, o_{id}, counter)
3: $o_{newVersion} \leftarrow 1$
4: **else**
5: KEEPALIVE(writeLock)
6: $o_{newVersion} \leftarrow o_{version} + 1$
7: **end if**
8: UPDATECACHE(o_{id}, $o_{newVersion}$, o)
9: primary $\leftarrow$ PRIMARYOF(o_{id})
10: primaryStoreAck $\leftarrow$ primary.STORE(hg_{id}, o_{id}, writeLock, $o_{newVersion}$, o)
11: backup $\leftarrow$ BACKUPOF(o_{id})
12: backupStoreAck $\leftarrow$ backup.STORE(hg_{id}, o_{id}, writeLock, $o_{newVersion}$, o)
13: self.LOG_OUT(hg_{id}, o_{id}, primaryStoreAck, backupStoreAck)
14: **return** [o_{id}, $o_{newVersion}$]

Algorithm 6 Point of Presence: Function `CREATE`.
write(hg_{id}, $old_{o_{id}}$, counter)

Input: The HG identifier, the identifier of the object `o`, and a creation counter.
Output: The object identifier.

1: backup $\leftarrow$ GETBACKUP($old_{o_{id}}$)
2: $o_{id} \leftarrow$ CREATEID(self, backup, hg_{id}, counter)
3: **return** o_{id}

Algorithm 7 Point of Presence: Function `STORE`.
store(hg_{id}, o_{id}, writeLock, $o_{newVersion}$, o)

Input: The HG identifier, the identifier of the object `o`, a write lock, the version number, and the object `o`.
Output: An acknowledgment containing the POP identifier, the object identifier and its new version number.

1: UPDATECACHE(o_{id}, $o_{newVersion}$, o)
2: HGList $\leftarrow$ GETHGLIST()
3: **for each** hg in HGList **do**
4: storeAck $\leftarrow$ hg.STORE(o_{id}, $o_{newVersion}$, o)
5: self.LOG_IN(hg_{id}, o_{id}, storeAck)
6: **end for**
7: **return** [self, o_{id}, $o_{newVersion}$]

Algorithm 8 Home Gateway: Function `STORE`.
store(o_{id}, o, $o_{version}$)

Input: The object `o`, its identifier, and the version number.
Output: The identifier of the HG, the object identifier, and the version number.

1: self.WRITEONDISK(o_{id}, o, $o_{version}$)
2: UPDATECACHE(o_{id}, $o_{version}$, o)
3: **return** [self, o_{id}, $o_{version}$]

object, and the *version flag* which indicates both the consistency criteria and the versioning view. The different types of consistency criteria handled by Mistore are the read/write mutal exclusion criteria, the atomicity, and the eventual monotonic-read consistency one. If the read operation must be handled by the primary or the backup POPs then a single one is chosen (through a random

choice). Figure 9 shows the code executed by HGs to handle a read operation, Figure 10 the code executed by a POP when it receives a read operation from a HG located in its region. Figure 11 shows the code executed by a POP to retrieve a requested object from its cache or from the HGs in its region and finally Figure 12 describes the code executed by a HG to retrieve an object from its local disk. In the following we present how the different consistency criteria are handled.

Read/write mutual exclusion: When a read operation is invoked with the read/write mutual exclusion flag, a read lock must be provided. When a POP receives this request from a HG, it maintains the read lock active until the requested object has been received by the HG that invoked the read operation. A read lock contains the last version number of an object o and ensures that as long as the read lock is active, no concurrent write operation on o will be executed. Thus, the read operation can read the object from the cache of any POP if it contains the last version of o, otherwise the request must be forwarded to the primary or the backup of the object.

Atomic consistency: When a read operation is invoked with the atomic consistency flag, no read lock must be provided. The read operation is forwarded to the primary or the backup of the object.

Eventual monotonic-read: When a read operation is invoked with the eventual monotonic-read flag, no read lock must be provided but the version flag must contain the most recent version number known by the client. The read operation can read the object from the cache of any POP if it contains a version equal to or newer than the version flag, otherwise the request must be forwarded to the primary or the backup of the object.

Versioning view: An object may be read from the cache of any POP or HG if it contains a version of the object equal to the one requested with the version flag, otherwise the request must be forwarded to the primary or the backup of the object.

Algorithm 9 Home Gateway: Function `READ`.
read(o_{id}, versionFlag)

Input: The identifier of the object `o` and a version flag specifying the desired object version.
Output: The requested object identifier, the object, and its version number.

```
1: if isCached(o_id, versionFlag) then
2:     [o_id, o, o_version] ← self.readOnDisk(o_id, versionFlag)
3: else
4:     myPOP ← self.getPOP()
5:     [o_id, o, o_version] ← myPOP.read(o_id, versionFlag)
6: end if
7: updateCache(o_id, o_version, o)
8: return [o_id, o, o_version]
```

Algorithm 10 Point of Presence: Function `READ`.
read(o_{id}, versionFlag)

Input: The identifier of the object `o` and a version flag specifying the desired object version.
Output: The requested object identifier, the object, and its version number.

```
1: if checkReadLock(versionFlag.readLock) then
2:     keepAlive(versionFlag.readLock)
3: end if
4: if isCached(o_id, versionFlag) then
5:     [o_id, o, o_version] ← self.readOnDisk(o_id, versionFlag)
6: else
7:     POPReplica ← random(primaryOf(o_id), backupOf(o_id))
8:     [o_id, o, o_version] ← POPReplica.retrieve(o_id, versionFlag)
9: end if
10: updateCache(o_id, o_version, o)
11: return [o_id, o, o_version]
```

Algorithm 11 Point of Presence: Function `RETRIEVE`.
retrieve(o_{id}, versionFlag)

Input: The identifier of the object `o` and a version flag specifying the desired object version.
Output: The requested object identifier, the object, and its version number.

1: **if** isCached(o_{id}, versionFlag) **then**
2: [o_{id}, o, $o_{version}$] ← self.readOnDisk(o_{id}, versionFlag)
3: **else**
4: HGReplica ← getHGReplica(o_{id}, versionFlag)
5: [o_{id}, o, $o_{version}$] ← HGReplica.retrieve(o_{id}, versionFlag)
6: **end if**
7: updateCache(o_{id}, $o_{version}$, o)
8: **return** [o_{id}, o, $o_{version}$]

Algorithm 12 Home gateway: Function `RETRIEVE`.
retrieve(o_{id}, versionFlag)

Input: The identifier of the object `o` and a version flag specifying the desired object version.
Output: The requested object identifier, the object, and its version number.

1: [o_{id}, o, $o_{version}$] ← self.readOnDisk(o_{id}, versionFlag)
2: updateCache(o_{id}, $o_{version}$, o)
3: **return** [o_{id}, o, $o_{version}$]

4.6 Evaluation

We have implemented Mistore on the Peersim simulator and performed all the simulations based on network parameters observed in a typical ISP infrastructure (see Table 4.1).

Table 4.1: Network parameters used in simulation

Network path	Upload (MB/s)	Download (MB/s)	Latency (ms)
HG - POP	5	20	75
POP1 - POP2	40000	40000	5
POP - DC	10000	10000	5

Figure 4.2 illustrates different write requests schemes. We observe that latencies are the lowest when objects are written synchronously on both their primary and backup POPs, the best case being when the write requests originate from the primary or the backup region. This gives us the insight that the primary and backup POPs of an object should be the POPs of the regions from which originate most of the write requests on the object. We also observe that writing synchronously an object in a remote HG shows the worst performance due to the low users network bandwidth. Both observations validate our choice to replicate objects on their primary and backup POPs in a synchronous way, and to replicate them asynchronously in the HGs of their regions. However the performance of writing objects in HGs can be considerably improved when a data stripping method is applied on objects (*i.e.* the object is fragmented in several blocks that are stored in parallel in different HGs). Nevertheless we observe that gains obtained by using data stripping are more substantial on big size objects than on smaller ones.

Figure 4.3 illustrates performances of different read requests scenarios on remote nodes. First, latencies are the lowest when objects are read on a POP, the best case being the one when this POP is the one of the region from where the read request comes. Reading an object entirely on a remote HG is in general slow due to the low upload bandwidth of users. However, the latencies can be reduced when objects are read from several HGs in parallel. This motivates to use data stripping

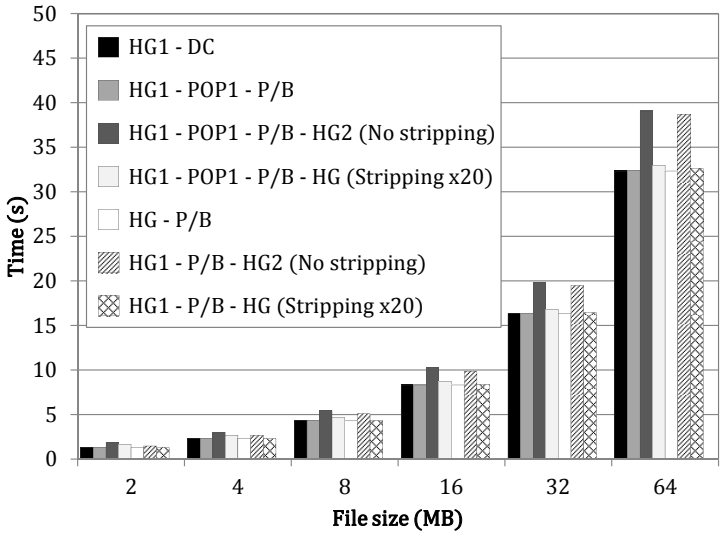


Figure 4.2: Write latencies. `HG-DC:` writing on the datacenter. `HG-POP-P/B:` writing an object on both its primary and backup POPs from a region different from these POPs region. `HG-POP-P/B-HG:` writing an object on HGs of both its primary and backup POPs from a region different from the primary and backup regions. `HG-P/B:` writing an object on both its primary and backup POPs from one these POPs region. `HG-P/B-HG:` writing on HGs of both the primary and backup POPs of an object from one of these POPs regions. `No stripping:` an object replica is written entirely in one HG. `Stripping x20:` an object replica is fragmented in 20 blocks stored in parallel in 20 different HGs.

methods when storing objects such that they can be retrieved by aggregating the bandwidth of several HGs. Nevertheless, as for write requests, data stripping is more efficient on big size data objects.

These results give us some insights for cache replacement policies. Actually, differences, regarding latencies, between storing small objects on their primary and backup or synchronously in remote HGs is not very large so we believe that big objects should have the priority to be kept in cache. Moreover, small objects may also be stored synchronously in remote HGs without a big impact on performance.

We have also evaluated the impact of HGs and POPs failures on write and read operation latencies. Prior to describing simulation results, we briefly present how the system reacts to failures.

Crash of HGs: When a HG fails by crashing, the list of objects it stores is retrieved from the index maintained by the POP in its region. To keep a given degree of redundancy, these objects may need to be recovered from other replicas and replicated in another HG. The results of the pending requests of a lost HG are cached in the POP in its region. Thus, when the HG recovers, acknowledgments are sent to the HG for write requests, and read requests are satisfied by accessing the cache of the POP in the region.

Crash of POPs: The failure of a POP means no Internet connection for the clients in its region. Thus we cannot not evaluate read and write latencies for users in a region where the POP is unavailable. Moreover, the crash of a POP means the crash of the primary or the backup of several objects. Without the primary of an object, its consistency can not be ensured so a new primary must be elected. As soon as the crash of a primary of an object is detected, its backup becomes the new primary and the datacenter becomes the new backup when an operation requiring a lock is activated. The backup can then trigger the lease of the locks it holds and takes over the service to the point where the old primary crashed. A crash of a POP is considered to be transient so a primary takes over the service when it recovers.

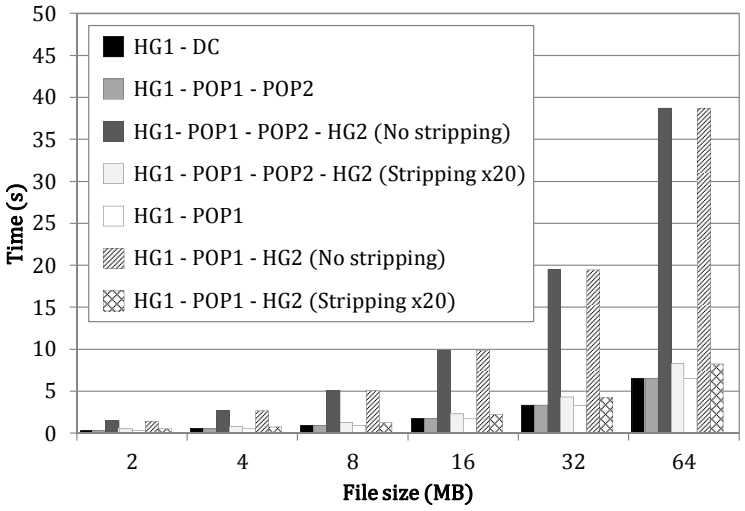


Figure 4.3: Read latencies. `HG-DC:` reading on the datacenter. `HG1-POP1-POP2:` reading on the POP of a region different from the one where the request is issued. `HG1-POP1-POP2-HG2:` reading on one HG connected to the POP of a region different from the one where the request is issued. `HG1-POP1:` reading on the POP of a region from where the request is issued. `No stripping:` the object is read entirely in 1 HG. `Stripping x20:` the 20 blocks stored in different HGs composing the object are read in parallel.

Figure 4.4 shows that during a write operation, if either the primary or the backup POP of the object becomes unavailable, the operation can finish without incurring a large latency overhead. This is due to the fact that the replication occurs in the IP core network, and thus the low user bandwidth is not involved when an object has to be replicated to a datacenter due to the unavailability of one of its primary or backup POP.

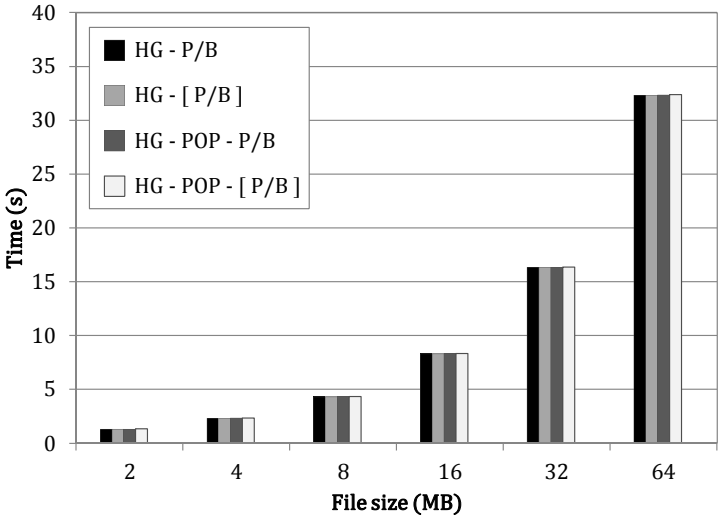


Figure 4.4: Write latencies in presence of crashes. See Figure 4.2 for meanings of `HG-P/B` and `HG-POP-P/B`. The element surrounded by brackets is the failed one. `HG-[P/B]:` writing from the primary POP region of an object and the primary or the backup POP is unavailable. `HG-POP-[P/B]:` writing of an object from a region different from its primary and backup regions while either the primary or the backup POP is unavailable.

Figure 4.5 shows the results of read operations during POPs and HGs unavailability. As argued before, we do not plot scenarios where the closest POP is unavailable — as it cuts users Internet connection — nor scenarios where the primary or the backup is unavailable — it does not incur any overhead as the object may be read on the available primary or backup POP. Similarly, we do not plot scenarios where HGs are unavailable since the POP knows which HGs are available and then will choose the available HGs to retrieve the requested objects. Now, when the POP (primary or backup) becomes unavailable after retrieving objects from the HGs and before forwarding them to their requester, we can observe that the overhead regarding latencies is very large when objects are not stripped over several HGs.

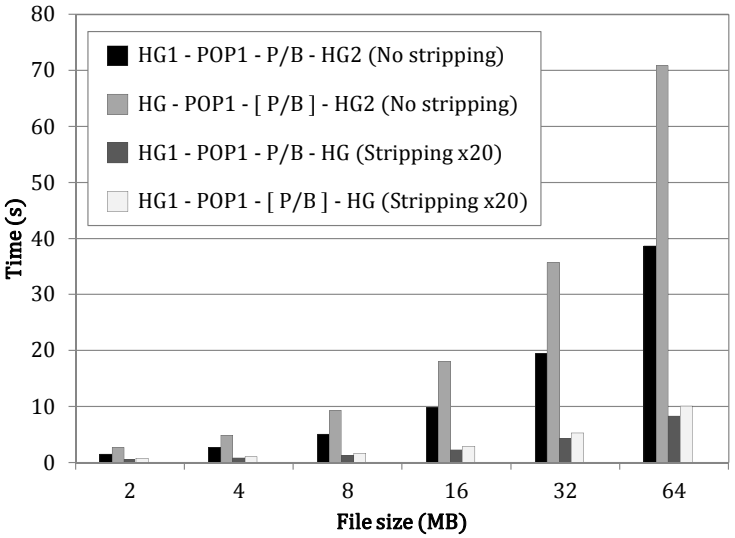


Figure 4.5: Read latencies in presence of crashes. See Figure 4.3 for the meanings of `HG1-POP1-P/B-HG2 (No stripping)` and `HG1-POP1-P/B-HG2 (Stripping x20)`. The element surrounded by brackets is the failed one. `HG1-POP1-P/B-[HG2] (No Stripping):` attempt to read an object in a HG located in a remote region but this remote HG is unavailable. `HG1-POP1-[P/B]-HG2 (No Stripping):` attempt to read an object in a remote HG located in a remote region but the POP of that region (primary or backup) becomes unavailable before transmitting the retrieved object. `HG1-POP1-[P/B]-HG2 (Stripping x20):` the same as the previous one but the object is retrieved by aggregating the object from 20 HGs in parallel before their associated POP becomes unavailable.

4.7 Related works

Architecture: Recently, some works have investigated new architecture designs to face the dynamicity issues of pure P2P systems [17] and the cost on large data centers [27]. Most of them adopt an hybrid approach combining peers and datacenters. [129] targets online backup applications. [138, 57] are social storage services. For example, in [57], users can decide to store their content on friends devices, in the cloud, or a mix of cloud and friends devices. This peer-assisted approach is also used for content distribution purpose [123]. An other approach is to rely only on distributed and stable small datacenters [27, 133]. The use of POPs in Mistore shares some similarities with the field of Content Delivery Network (CDNs) [99, 40, 98] where cache servers are deployed at specific locations in the Internet infrastructure as close as possible to the users for a fast content delivery. In contrast to those approaches, Mistore is dedicated to an ISP which is an advantage over other proposals because an ISP fully masters its network infrastructure and thus can push contents closer to its users.

Multiple consistency criteria: It is well known that strong consistency models have lower performance than weak consistency models in terms of availability, latency, and scalability [134, 2]. Most of storage systems provide the eventual consistency forms [134]. However, even if it is sufficient for a large panel of applications some others require strong consistency criteria [36, 74]. Moreover, a strong consistency is a desirable property for application programmers as it is easy to reason about. As a consequence, several systems target strong consistency while others aim to add strong consistency features to provide multiple consistency criteria in their system [29, 102]. Mistore follows this vision by giving the possibility to choose the consistency criteria adapted to the application needs.

4.8 Summary and perspectives

In this chapter we have presented the main principles of Mistore, a distributed storage system dedicated to users who access Internet via a DSL technology. Mistore aims at guaranteeing data availability, durability, and low access latency by exploiting home gateways storage capacities, POPs physical localizations, and the datacenter. Simulation results have shown that Mistore replication strategy decreases data access latencies, and stripping method works better on large objects.

A direction for future work is to leverage the DCs knowledge on the system to implement multi- and auto-tiered storage features which can refer to an automated placement of data on nodes to optimize performance, availability, and recovery [6]. Indeed, we have nodes of different types and locations that could be distinguished in three tiers for storing cold, warm, and hot data while allowing data to move from one tier to another depending on data access patterns.

- **Cold data** are infrequently or never accessed. Datacenters would be suitable to handle the steadily growing amount of cold data in a cost efficient manner using inexpensive and energy efficient storage devices with large capacities. In fact, it is more flexible to add storage capacities in DCs than in regions which requires adding new HGs or changing old ones with HGs with more storage capacity.
- **Warm data** are data that have been recently accessed in the storage system and present a higher probability to be accessed than cold data. They could be stored in regions from where their users access the storage system to reduce access latencies. It could also localize traffic within regions and minimize network traffic across regions.
- **Hot data** are frequently accessed and could be stored in POPs and distributed like in a CDN. It would allow to answer users requests while avoiding to overload HGs that have low upload bandwidths or datacenters that may exhibit large access latencies.

We are also convinced that our architecture could also be suitable for user-generated content distribution and storage costs reduction via a multi-tiered data management. Evaluation of those points would be interesting to pursue in future work.

5

Data placement and maintenance in Mistore

In the previous chapter we presented the architecture of Mistore. In this architecture, the data placement method used in regions is a table-based one. That is, the Point of Presence of a region maintains all the metadata to locate or store data within the region. Table-based data placement methods are simple to deploy however they lack in scalability as the amount of data and metadata to manage increase. In this chapter, we present a more scalable and flexible data placement scheme in which the amount of metadata to maintain depends on the number of home gateways in the region and not on the potentially enormous amount of data to manage.

Contents

5.1 Introduction

In Chapter 4 we presented the architecture of Mistore in which the data placement and maintenance was managed by the regions POPs. The method was a directory-based method. Each POP maintained a list of home gateways and the data stored within them. As we saw in Section 3.2.3, this type of data placement is flexible but lack in scalability.

In this chapter we propose a solution to improve the data placement and maintenance in the regions of the architecture of Mistore and design it to be more scalable.

We choose to design a pseudo-random placement technique to manage the data placement and maintenance as this kind of placement method promises more scalability than directory-based methods.

Our solution aims to be scalable by reducing the amount of metadata location to maintain and allowing the data location to be found by computation instead of using metadata table/directory traversal methods. Our solution uses notions of services, data pools, and placement groups in order to be more flexible (we describe these notions in Section 5.2). As all the data are not the same, these notions allows to parameterize and thus to diversify the way data are managed. These parameters can be the methods to retrieve, store data (*e.g.* using data stripping or not), or the redundancy method (*e.g.* replication or erasure coding).

In the remaining of this Chapter we describe in Section 5.2 the concepts used in our solution, in Section 5.3, we describe how the placement method in our scheme works, and in Section 5.4 we describe the data maintenance in our scheme. We conclude and discuss future works in Section 5.6.

5.2 Definitions and concepts

In this section we define the different concepts that we use in our solution. We borrow some concepts from the state of art in data placement and maintenance methods [90, 136, 137, 25, 67, 72, 142].

Service

A service is an application to which is allocated storage capacity within the infrastructure of the provider. In this context, the provider is an ISP and we consider storage capacity allocated in the home gateways in the network infrastructure of the ISP. Examples of services can be a photo backup application or a personal cloud storage application.

The pools

A pool is actually the storage space allocated to a service. A service may have several data pools (See Figure 5.1). Since we consider storage space of home gateways in the network infrastructure of the ISP, we can not know in advance when storage space capacity is added or removed from the infrastructure. The reasons of adding/removing capacity are multiple (*e.g.* the subscription of a new customer so a new home gateway is added in the infrastructure, the reconnection of a home gateway in the infrastructure, the unsubscription of a customer so its home gateway is removed from the infrastructure, a home gateway crash or simply a disconnection, etc). In this work, we assume that the storage space allocated to all the services is less than the global storage space available in the infrastructure. We also consider the capability to extend or reduce the storage space allocated to a service (*i.e.* storage space allocated to a pool or the number of pools) without violating our assumption. Note that to not allocate all the available storage space in the infrastructure allows to keep some storage space that can be used to extends services or pools storage space, or replace crashed home gateways that were associated to pools.

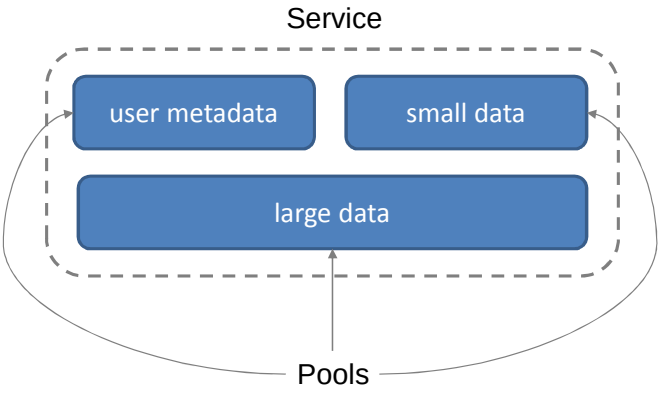


Figure 5.1: A service can have several pools.

Pools can be used to manage separately the way different types of data are stored. A service may have a pool for storing images and another for storing videos. The idea is that data belonging to the same pool will be stored and retrieved in the same manner in terms of data replication, placement, and maintenance. To achieve this, we configure pools with some parameters.

At the creation of a pool $pool_i$ for a service, the pool size, the redundancy algorithm and its parameters, the read/write algorithms, and the data maintenance strategy are defined. To the ease of understanding we will consider redundancy through simple replication. So we define R_{pool_i} the redundancy degree that will be applied in $pool_i$ which represents the number of replica of each data objects stored in $pool_i$.

The redundancy degree R_{pool_i} depends on the estimated home gateway availability in the infrastructure and the data availability we want to achieve for this pool. The formula to compute the redundancy degree can be found in the state of the art whether redundancy relies on simple replication or erasure codes [135, 110, 81]). In the following, we consider that we know the value necessary to compute the redundancy degree.

Placement groups

A Placement Group (PG) is a logical representation of a group of home gateways in which a data object and its replicas are stored. Several objects can be replicated in the same placement group (See Figure 5.2). A placement group is associated to a pool and the way data are stored and replicated within a placement group is inherited from the parameters of the pool. Thus all the data items stored in a placement group are stored and retrieved in the same manner. In fact, once we know the placement group in which a data item is stored, we can determine the home gateways that compound this placement group and we can store or retrieve this data item.

Placement groups allow to not store all the metadata of where data and their replica are stored in the infrastructure. Instead, the amount of metadata that we need to maintain depends on the number of home gateways and placement groups and not on the number of data and their replica stored in the system that can be potentially enormous. Placement groups help to reduce the amount of location metadata to maintain in the system. They also ease the reverse lookup process as nodes of the same placement group stores the same data. Knowing the placement groups a node is involved in is enough to recover the data that were store on that node using the other nodes of its placement groups.

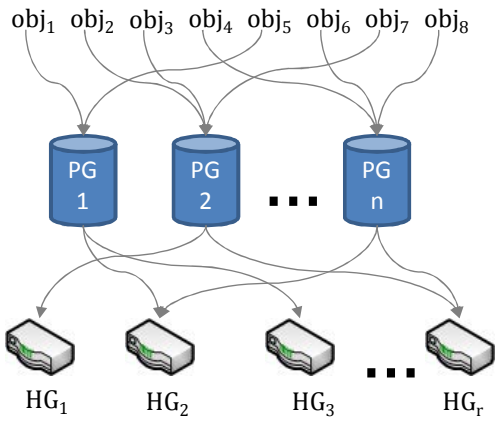


Figure 5.2: Several different objects can be stored in the same Placement Group (PG).

Fault domains

In order to improve data reliability, it is necessary to store the different data items and their replicas on different home gateways and ideally having independent sources of faults. To have a view of the potential sources of correlated failures between the home gateways, we introduce the notion of *Fault domains* (FD) on all the home gateways. Home gateways in the same FD share the same sources of failures. This notion of fault domains is useful when we create a placement group and we want to associated home gateways to this placement group. Using the concept of fault domains, we can form a placement group with home gateways belonging to different fault domains. We do so to reduce the impact to correlated faults among home gateways storing the same data.

In our infrastructure, we consider the connection to the DSLAM as a fault domain (See Figure 5.3). Thus all the home gateways connected to the same DSLAM belong to the same fault domain. Note that this consideration can be subject to evolution to take into account other criteria such as network bandwidth, electrical connection, the lifetime of the home gateways, etc.

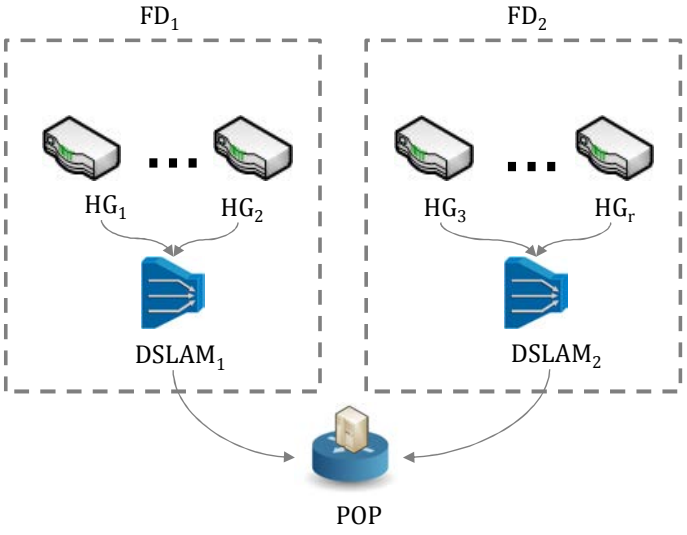


Figure 5.3: Home gateways (HGs) connected to the same DSLAM belong to the same Fault Domain (FD).

5.3 Data placement description

In this section we describe our data placement solution regarding data stored in a region in Mistore namely the different steps in the initialization of the systems, the `read/write` operation in a pool, and the maintenance of the placement groups of a pools.

System initialization

Figure 5.4 illustrates the different steps in the initialization of the system:

1. Creation of faults domains
2. Creation of the required number of PGs
3. Association of home gateways to placement groups
4. Placement groups organization

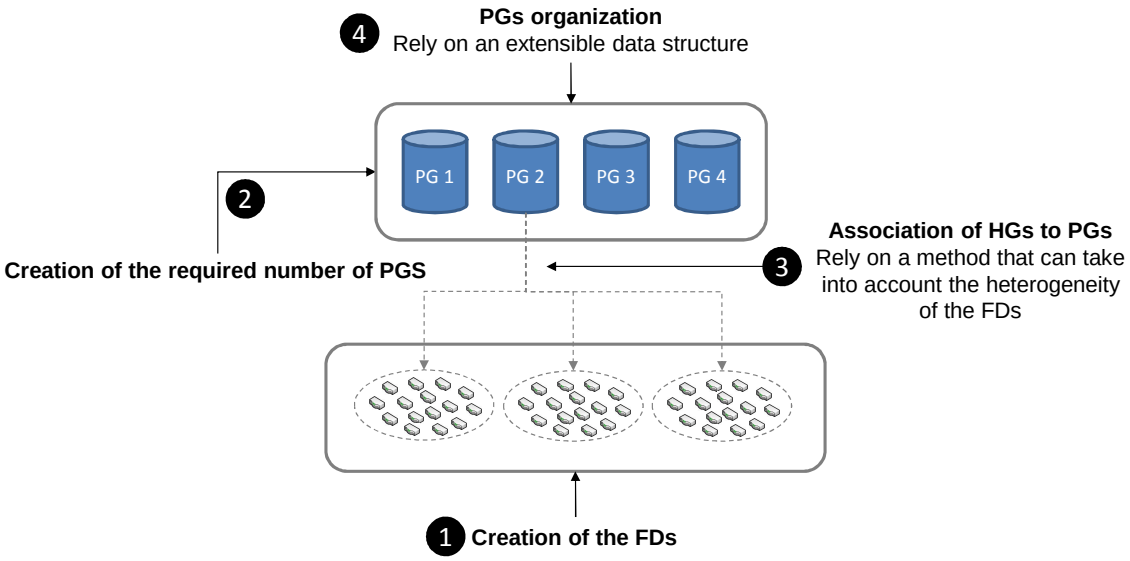


Figure 5.4: The different steps in the initialization of the placement system.

Creation of the fault domains

In this step, home gateways are classified regarding fault domains. All the home gateways connected to the same DSLAM are in the same fault domains. Then we compute the storage capacity of the domain fault by computing the sum of storage spaces available in its home gateways. We measure the storage capacity of a box in terms of placement groups and its computed as follows:

$$nbPG_{HG_i} = \frac{storageCapacity_{HG_i}}{PGSizeMin_{HG}} \tag{5.1}$$

With :

- $nbPG_{HG_i}$ the maximum number of placement groups of which the home gateway HG_i can be part of.

- $storageCapacity_{HG_i}$ the global storage capacity of the home gateway allocated to Mistore.
- $PGSizeMin_{HG}$ the minimal amount of storage space that can be allocated to a placement group by a home gateway.

The storage space capacity of a fault domain represents the maximum number of placement groups that can be created in this fault domain. keeping track of the fault domains capacity allows to associate home gateways to placement groups using the fault domains uniformly and proportionnaly to their capacity. In fact all the fault domains do not have necessarily the same capacity because they do not have the same number of home gateways. Since we consider a fixed number of DSLAMs, it is the same for number of fault domains. Note that the capacity of a fault domain may evolve depending on the departure and arrival of home gateways. The sum of all the capacity of the home gateways of a fault domain represents its capacity, it is expressed as follows:

$$nbPG_{FD_i} = \sum_{i=0}^{n} nbPG_{HG_i} \tag{5.2}$$

With:

- $nbPG_{FD_i}$ the storage capacity of a fault domain. It is expressed as the maximal number of placement groups that can be created in this fault domain.
- n the number of available home gateway in this fault domain.
- $nbPG_{HG_i}$ the maximum number of placement groups of which the home gateway HG_i can be part of.

Creation of the required number of PGs

When a pool $pool_i$ is created, a size and number of placement groups are attributed to that pool. The number of placement groups is computed as follows:

$$nbPG_{pool_i} = \frac{size_{pool_i}}{R_{pool_i} * PGSizeMin_{HG}} \tag{5.3}$$

With:

- $nbPG_{pool_i}$ the number of placement groups to created
- $size_{pool_i}$ the given raw size of the pool $pool_i$
- R_{pool_i} the redundancy level defined on that pool
- $PGSizeMin_{HG}$ the amount of space a home gateway allocates to one placement group. This value is a parameter of the system

Association of home gateways to placement groups

In this step, a number of R_{pool_i} are associated to the placement groups of a pool. Recall that the home gateway of a pool must belongs to different fault domains. Likely, not all the fault domains can provide the same storage space capacity to the system so it is convenient to associate the home gateway to the placement groups using a method that can achieve it in a uniform manner taking into account the heterogeneity in the storage capacity of the fault domains. Example of such techniques can be inspired from pseudo-randomized placement method [90, 136, 137, 25, 67, 72].

Note that the capacity of a fault domain decreases when some of its home gateways are allocated to placement groups. Similarly the capacity of a fault domain increases when new home gateways are added.

Placement groups organization

To be able to increase and decrease storage space allocated to a pool by adding or removing placement groups, we need to organize the placement groups using an extensible data structure in order to keep the data location lookup consistent despite placement groups adding or removal that may occurs. Several algorithms exist in the literature but we choose to base our solution on the random-slice algorithm [90] which has several desirable properties. This technique allows to place data on a subsets of nodes in a uniformly manner, it also minimizes data migration upon nodes addition or removal and keeps small the amount of location metadata to maintain.

In order to support redundancy and ensure that replica will be stored in home gateways belonging to different failure domains, we associated the random-slice method with the placement group concept. Actually, we apply the random slice method not on nodes but on placement groups as they may be viewed as virtual nodes with embedded redundancy taking into account failure domains. In the following we describe how we leverage this method on the placement groups of a pool.

A name space is defined on $pool_i$. Using the random-slice method, this name space is defined on an interval [0, 1[which is entirely distributed to the existing placement groups. All the placement groups will be responsible of a range of equal size. Placement groups can be responsible of several ranges but the sum of the length of these intervals should be equals for all the placement groups (See Figure 5.5).

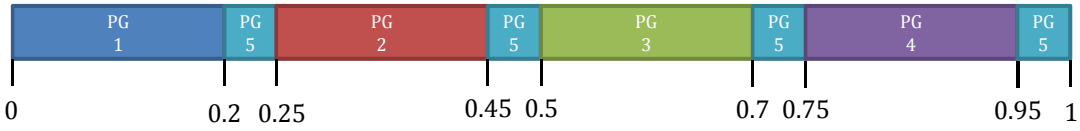


Figure 5.5: Representation of the placement groups name space of a pool

Adding placement group to a pool: Placement groups are added to a pool in order to add storage space to that pool. Recall that a pool has a name space in which each placement group is responsible of an interval. When the number of placement groups is determined (depending on the amount of storage space to add to the pool), the following operations are applied:

- The size of the interval each placement group should be responsible is computed.
- The size of the interval that each existing placement group should abandon to the new placement groups is computed
- The size of the existing placement groups are updated.
- The intervals that are free are distributed to the new placement groups. A placement group can be responsible of several intervals that are non contiguous but the sum of the length of these intervals is the same for all the placement groups.

Figure 5.6 illustrates the operation of adding a placement group in a pool. Objects that are already stored on the interval attributed to a new placement group should be migrated from their old placement group to their new placement group. To achieve this, the home gateway of rank i of the old placement group send the required objects to the box of rank i of the new placement group. Objects migration ensures that objects are uniformly distributed among all the placement group either old or new. It also ensures that data migration is only performed from old placement groups nodes to new placement groups nodes.

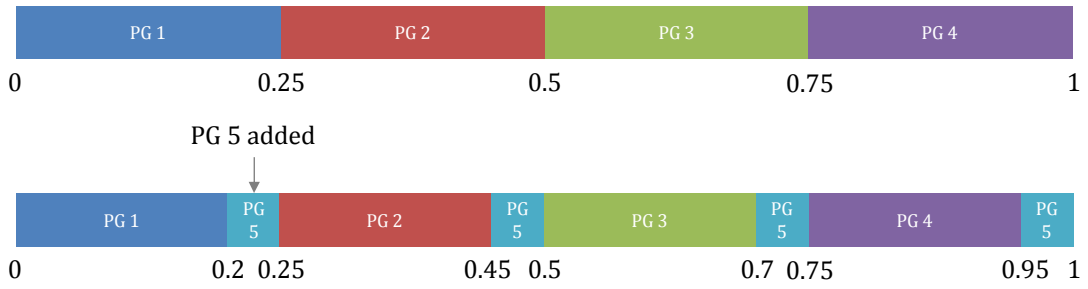


Figure 5.6: Adding a placement group

Removing placement group to a pool: Remove placement groups from a pool reduces the storage space allocated to a pool. Similarly to adding placement groups to a pool, the interval of the removed placement groups are distributed uniformly to remaining placement groups. The objects stored in the removed placement groups are then migrated to the new placement groups.

Read and Write operations

Write operations

Write requests are called trough the method named `PUT` which takes into parameters the object `obj`, its identifier $\texttt{ID}_{obj}$, and the pool in which the object will be stored. Figure 5.7 illustrates a `PUT` operation.

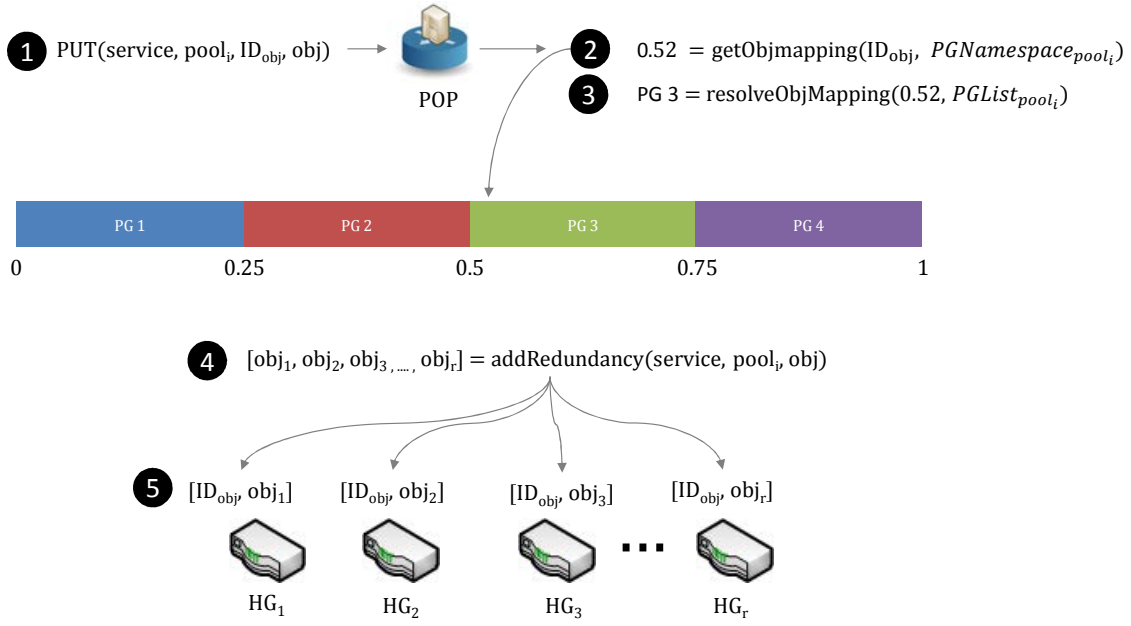


Figure 5.7: Write operation

Objects are mapped uniformly on $PGNamespace_{pool_i}$ the namespace of the pool $pool_i$. It is achieved though a method named $getObjMapping$ as follows:

$$mapping_{obj} = getObjMapping(ID_{obj}, PGNamespace_{pool_i}) \tag{5.4}$$

With:

- ID_{obj} the identifier of the object *obj*
- $PGNamespace_{pool_i}$ the data structure implementing the name space of $pool_i$
- $mapping_{obj}$ the mapping of the object *obj* in this name space.

The object *obj* will be stored in the placement group responsible of the interval where the object is mapped. This is achieved by a method named $resolveObjMapping$ as follows:

$$PG_i = resolveObjMapping(mapping_{obj}, PGList_{pool_i}) \tag{5.5}$$

With

- $mapping_{obj}$ the mapping value of the object *obj* in $PGNamespace_{pool_i}$ the name space of the pool $pool_i$.
- $PGList_{pool_i}$ the current list of placement groups in the pool $pool_i$.

When the placement group is determined, the POP applies the redundancy algorithm defined at the pool level in order to create the R_{pool_i} replicas or blocks of the object *obj* to be stored in the home gateways of the placement group. The home gateways are ordered in a placement group. The POP stores the replica or block i of *obj* in the home gateway with the rank i in the placement group.

Read operations

A read operation on an object *obj* is performed through a *get* method that takes into parameters the identifier of that object. Similarly to a write operation, first, the placement group where that object is stored have to be determined, then the home gateway in this placement group can be contacted to get the object. The way the object is retrieved depends on the redundancy and read method defined at the pool level. If the redundancy method is the replication, an approach may be to choose randomly a replica between the R_{pool_i} replicas in the placement group. An other approach, data stripping, may be can be used to retrieve different parts or blocks of the object in parallel in the R_{pool_i} replica in order to aggregate network bandwidth.

Read and write operations during migration

After a placement group is added or removed, the objects distribution on all the placement groups of a pool is still statistically uniform as the sum of all the intervals length each placement group is responsible of are equals.

The objects migration due to the add and removal of placement group are not urgent. In fact the durability of the concerned objects is not impacted since there are stored in the home gateway of these placement groups. It is thus not mandatory to perform these migration synchronously. In fact objects migration may create a pick in the bandwidth utilization in the system and reduce the performance for concurrent clients operations. We suggest to migrate objects asynchronously in order to not impact negatively the system performance. However we must ensure the availability of the object concerned by the migration *i.e.* we must be able to locate objects whether they are already migrated to they new placement groups or not.

The objects migration are ready to be performed when the new placement groups are created and initialized with home gateways. The POP maintains all placement group data structures until all the objects are migrated. When objects migration are finished, the data structures storing the old placement groups representation can be deleted. Depending on the state of the network, the POP can trigger the object migration from one placement group to an other. The POP can request a home gateway to migrate all the objects or a subset of these objects that are mapped to the

interval of the name space that need to be migrate to a new placement group. An other strategy is to get from the home gateways the lists of the objects that need to be migrated and organized the migration using these lists.

As home gateways are connected by the POP, objects migrated must go through the POP that can keep them in a cache. Thus, an object migration is divided in two steps. First the object is sent to the POP. Second the POP forwards the object to the home gateway of the new placement group where the object has to be stored. The POP may delay the second step depending on the load in the system. We claim that these steps must be performed when the activity in the system is low.

Write operations during migration: When a write operation is concurrent to the add or the removal of placement groups in a pool, the placement groups organization is used to determine where the object been written should be stored.

Read operations during migration: When the operation that is concurrent to the add or removal of placement in a group is a read, the POP, first, tries to read the object from its cache. If the object is not in the cache, the POP determines if the object is mapped to an interval impacted by the migration. If the object has already been migrated, the POP retrieves the object from its new placement group otherwise it retrieves it from the former placement group containing the requested object.

5.4 Data maintenance description

In this section we describe our data maintenance solution regarding the departure and arrival of home gateways in regions of Mistore.

A home gateway can be unavailable making also the data stored in this home gateway unavailable. The unavailability of a home gateway can be transient or permanent:

- **Transient unavailability:** The home gateway may just have been turned off by the user. In this case we can suppose that the durability of the data stored in this home gateway is maintained and that these data will be available for read and write operations as soon that the home gateway is powered and returns in the system.

- **Permanent Unavailablity:** It can be to a crash. In this case, all the data stored in this home gateway are lost. However replica of these data can be found in other home gateways of the placement groups in which these data are stored.

Monitoring service

Despite the home gateways unavailability that can occur in the system, the redundancy level of each object in the system must be maintain in order to ensure its availability and durability. For instance when a home gateway is permanently unavailable it can be necessary to create new copies replicas of the replica lost.

In order to propose a data maintenance method to deal with departure of nodes, a system monitoring is necessary. This monitoring system has the purpose to maintain a view of the home gateway in the system allowing to deduce the availability of the data stored. For instance knowing the number of home gateways that are available in a placement group, we can compute the availability of the data stored in this placement group regarding the redundancy degree defined at the pool level.

In the following we describe this monitoring service and our data maintenance proposition to ensure data durability and availability. This data maintenance relies on the information received by the monitoring service. These information can be related to the home gateway availability, their network bandwidth, their storage space capacity, etc. We will focus on the availability of the home gateways.

Home gateway availability: We describe three states to define the availability of a home gateway:

- State `up`: The home gateway is available.
- State `down`: The home gateway is assumed to be temporarily unavailable (the home gateway may just have been turned off). The POP does not forward requests (*i.e.* read and write operations) to a home gateway in this state. When a home gateway become in a down state, the POP checks if the availability degree of the data stored in the placement groups in which this home gateway is involved and proceeds to reconstruction of lost replica if needed.
- State `out`: The home gateway is considered definitively unavailable and no request is sent to this home gateway.

To monitor and maintain the state of the home gateways, each home gateway sends a message called `hearbeat` when it is (re)connected to the POP of its region. Heartbeat messages are sent periodically to the POP. The period of sending heartbeat messages named $t_{heartbeat}$ is predefined in the system. This heartbeat messages helps the POP to maintain the availability states of the home gateways.

The POP marks a home gateway `up` when it receives periodically heartbeat messages from that home gateway that can then receive read and write requests. After a period $t_{no\_heartbeat}$ if no heartbeat message is received by the POP from a home gateway, the POP marks that home gateway `down`. Note that a home gateway can be marked as `down` if that home gateway is unreachable when a request is sent to it. This can happen when the period $t_{no\_heartbeat}$ is not elapsed and the home gateway is unavailable. When a home gateway in the state `down` sends a heartbeat message before the predefined time t_{down_max} elapses, the POP marks that home gateway in a state `up` otherwise it is marked `out`. The availability degree of the data stored in the placement groups where that home gateway is checked to see if some data replicas reconstruction are needed.

The lazy data maintenance assisted by the POP

Our solution is based on the data maintenance methods described as **reactive** [26]. A basic reactive data maintenance whether it is **eager** or **lazy** uses a redundancy threshold on data. When an object availability degree reaches the defined threshold, new replicas are immediately created to avoid data lost. The main problem with this type of method is the state of emergency of the replica reconstruction when the threshold is reached. This may create a pick in the bandwidth utilization. Our method aims to reduce the emergency state during replica reconstruction by relying on our architecture.

We define three redundancy thresholds on the placement groups. These threshold correspond to the number of available home gateways in a placement group but also to the availability degree of the data stored in this placement group.

- Threshold T_{safe}: The initial threshold, it corresponds to the redundancy degree R_{pool} defined at the pool level. At this threshold no maintenance operation is needed. We assume the availability and durability of the objects stored in the placement group provided.
- Threshold T_{repair}: The threshold at which replica reconstruction is an emergency. When the threshold reaches this states, the POP re-creates redundancy to avoid data lost. The POP retrieves the necessary data to create new replica from the other available home gateways in

the placement group. The new replica can be kept in the cache of the POP before been stored in a new home gateway which will replace the older one that is definitively unavailable. The POP selects this home gateway in the same manner that home gateways are selected when placement groups are created (See Section 5.3). The new replica are available for read and write operations when there are in the cache of the POP so storing them in the new selected home gateways can be done asynchronously when the network bandwidth utilization is low in order to not disturb the quality of experience of the user.

- Threshold T_{cache}: This is a state between T_{safe} and T_{repair} such as $T_{repair} < T_{cache} < T_{safe}$. When the redundancy degree of an object reaches the state T_{cache}, the replica needed to reconstruct the object are uploaded from the remain available home gateways to be cached on the POP. Because this threshold is not critical to the availability and durability, cached these replicas are not done in an emergency state. Data can be sent to the POP in such a way to avoid load peaks in the network bandwidth utilization. For instance it can be done in periods of inactivity. Objects are kept in the cache of the POP during a defined period $Time_{cache}$. When some gateways leave the *down* state to the *up* state, the redundancy degree of their placement groups is updated. The cached objects can be deleted if the redundancy degree get superior to T_{cache}. Deleting objects from the cache may also depend on the object access pattern. In fact cached objects are available to read and write operations.

5.4.0.1 Home gateways arrival

A particular feature in our architecture is that we cannot explicitly add home gateways in the infrastructure. A home gateway can be added to the infrastructure due to the subscription of a new customer or the reconnection of a home gateway (*i.e* from state *out* to state *up*). A subscription policy may be set up to let the users decide whether their home gateway can be used by the storage system. In this thesis, we consider that every home gateway is enrolled in the storage system.

When a home gateway is added to the storage system, the first thing to do is to add it in a Fault Domain (See Section 5.3) of a region. Then the POP of that region manage the time when the home gateway will participate to the storage activity. In several systems new nodes automatically participate to the storage activity through a load balancing on a subset of nodes in the system [72] or on all the nodes [136, 137]. In our case, since the pools are already provisionned, to integrate new node in the storage activity does not need to be performed synchronously. We can integrate them asynchronously, in a lazy mode. We integrate new home gateway in the storage activity in the following cases:

- **Home gateway replacement:** When a home gateway is in the state *out*, the POP of the region selects others home gateways to replace it in the different placement groups in which the older home gateway was involved in. Selecting a home gateway to integrate in a placement group is performed just as in the system initialization (See Section 5.3).
- **Load balancing operation:** It can be performed to balance the storage load on more home gateways of a region. In this case, the operation is not an emergency, it is a maintenance operation that can be performed by the POP in a lazy mode (See Section 3.2.2), for example periodically taking into account the current workload in the region.

5.5 Related works

Several works on data placement methods has already been pursued in the literature focusing on different aspects such as load balancing, heterogeneity, redundancy, or reverse lookup support.

Redundant Array of Independent Disks (RAID), originally named as Redundant Array of Inexpensive disks [101], is a technique allowing to put together several physical disks to create a logical disk that may be larger, faster, more reliable dependent on the *RAID level* used. Different RAID levels optimize different aspects such as reliability by tolerating disks failure, performance by allowing

parallel data access on multiple disks, capacity by combining multiple disks, or a combination of them. RAID systems provides block storage as a single logical disk and can be used transparently by a file system on top of the RAID system. Compared to our solution, RAID systems only tolerate loss of disks. In our solution we are in a context of object storage and the solution is designed to tolerate lost of physical disks and storage nodes as well.

Controlled Replication Under Scalable Hashing (CRUSH) [137] is the data placement and maintenance algorithm used by Ceph [136] a distributed storage system. CRUSH and our solution share several similarities about the concepts used. CRUSH is a pseudo randomize method that distributes in a evenly manner objects on Object Storage Devices of a cluster, based on a storage device weighting system taking into account the heterogeneity of the storage devices. Ceph uses the concept of placement groups with CRUSH to manage redundancy and reverse lookup in the cluster however, in the first version of Ceph [136], the number of placement groups was determined and fixed at the creation of the cluster. Recent version of Ceph[1] allow to increase the number of placement groups but you can not decrease them. In contrast in our system, the number of placement groups can be increased or decreased.

Consistent hashing is a pseudo-randomize data placement using a namespace represented by a circle. Data and storage nodes are randomly and uniformly associated to a point in the namespace by a pseudo-randomize method taking node and data item identifiers as input. A storage node is responsible to store the data that are mapped between its point and the point of the following node on the circle. In case of an arrival, the new node takes responsibility of a range that were previously managed by its predecessor. That is the load in case of arrival/departure of node is not share among all the nodes in the system. Moreover, consistent hashing does not support nodes heterogeneity on its own. A solution to introduce heterogeneity in consistent hashing is to use the technique of virtual node [121] which consists of attributing several identifiers to a node that has more resources than the others. In that case, the node will be mapped to different points and eventually will manage more data than smaller nodes. In our work, we use a technique similar to a virtual node technique to allow a storage node to be part of several placement groups and make the storage node appear homogeneous in terms of storage space. And in contrast to consistent hashing technique, in our solution, when placement groups are added/removed, the nodes is share among all the placement groups of the system.

Random slice [90] is a placement method that distribute data among storage node of a cluster taking into account their heterogeneity. To achieve it, Random slice uses a name space represented by a segment defined on an interval of [0, 1[. To achieve heterogeneity, nodes are responsible of ranges of size that are proportional to their capacity. The ranges of a node can be contiguous or not. Data are mapped to ranges and are associated to the node responsible of the range. To keep the system balanced, when a node is added/removed, each node releases/acquires a range, proportionally of their capacity, of the name space to/from new/removed node.

SPOCA [25] is a pseudo randomize placement method aiming to distribute requests to a list of servers. It supports load balancing, heterogeneity. Each servers is assigned, proportionally to their capacity, a segment on a namespace represented by an interval. Not all the namespace is allocated to the servers, some intervals of the namespace are allocated to no servers. Requests are mapped on the namespace and routed to the server responsible of the interval in which the request has been mapped. Note that requests are remapped to the namespace until they are mapped to in an interval handled by a server. This allows to keep the system balanced in the case of servers arrivals and departures. However SPOCA presents some scalability and efficiency issues. In fact, in SPOCA, the size of the namespace is immutable so if the size of the namespace is small, the system lacks in scalability as the number of servers we can add will be limited. On the other side, if the namespace is very large, we may loose in efficiency if there is a large number of segments that are not allocated to servers.

Asura [67], is a data placement based on SPOCA algorithm and brings some enhancements to solve the scalability and efficiency issues in SPOCA. Asura achieves this by allowing to reduce and extend the namespace size or interval without causing unnecessary data migration. To do so, Asura

[1] http://ceph.com/

makes use of pseudorandom number generators. A pseudo random number generator generates a sequence of random numbers for a data item identifier in the interval of the namespace. When the namespace is extended, a new random number generator is added to generate random numbers on the new namespace. For each data, the sequences of the different random number generators are merged in a so called Asura random numbers. A data item will be stored on the first number of the Asura sequence numbers that falls in a segment that is attributed to a server. To reduce a namespace size, only unnecessary (*i.e.* those covering wider namespace) pseudo random number generators just have to be deleted.

While our method leverages clever design choices from the previous methods, our solution may be considered as a data placement framework that can be implemented using the previous algorithms when they present the required properties. For instance, a virtual node technique is used to deal with nodes heterogeneity, placement groups are used to provide reverse lookup capability, algorithm to provide redundancy can be parametrized. We also structured the placement groups using a random slice structure but other methods can also be used.

5.6 Summary and perspectives

In this Chapter, we described a data placement and maintenance scheme. This scheme aims to reduce the amount of metadata about the location of data in regions of Mistore. We design this scheme to be flexible so that one can configure elements such as the redundancy degree, redundancy techniques, the read and write methods, the cache retention period and the schedule of the data migration. Future work would be to implement this scheme and also make its easily pluggable in an existing system needing to locate, place, and maintain resources in a lightweight and scalable manner. Another direction may be to support several fault domain and heterogeneity characteristics.

6

A secure two-phase data deduplication scheme

Data grows at the impressive rate and a study by the IDC shows that 75% of the digital world is a copy [51]. Although keeping multiple copies of data is necessary to guarantee their availability and long term durability, in many situations the amount of data redundancy is immoderate. By keeping a single copy of repeated data, data deduplication is considered as one of the most promising solutions to reduce the storage costs, and improve users experience by saving network bandwidth and reducing backup time. However, this technology still suffers from many security issues that have been discussed in the literature. In this chapter, we target the attacks from malicious clients and present a method that we called two-phase data deduplication contributing to build a storage system that is secure against attacks from malicious clients. Our solution remains effective in terms of storage space and network bandwidth economy.

Contents

6.1 Introduction

The amount of data to be stored by cloud storage systems increases extremely fast. It is thus of utmost importance for Cloud Storage Providers (CSPs) to find means to reduce the cost for storing all the created data. A promising approach to achieve this objective is through data deduplication [47, 112].

By storing a single copy of redundant data, deduplication is one of the most promising technologies in storage systems. Storing a single copy of redundant data allows to save storage space and therefore it reduces the storage cost. Due to its straightforward economical advantages, data deduplication is gaining popularity in both commercial and research storage systems [47, 112, 68, 89, 18, 5, 141, 59, 44, 93, 86].

In cloud storage services, it also improves the clients user experience when it is applied prior to uploading the data the storage server. In fact, it allows to save network bandwitdh, *(i)* the users does not have to send more than one time the same data and *(ii)* the storage server does not have to receive several time the same data (See Section 6.1.1 for more information).

However, several works have recently revealed important security issues leading to information leakage to malicious clients [60, 62, 94, 122]. These security concerns arise especially in systems performing an inter-user and client-side deduplication which is unfortunately the kind of deduplication that provides the best savings in terms of network bandwidth and storage space. These vulnerabilities has been discovered on popular cloud storage services such as Dropbox [44], Memopal [86], and Mozy [93] Although several solutions exist in the literature, we observe deduplication still suffer from these vulnerabilities. In fact, most of the time these solutions only solve a subset of the attacks possible against deduplication.

In this chapter, we propose a solution allowing to perform an inter-user and client side deduplication that is secure against all the existing attacks from malicious clients against a system performing deduplication. In the following of this section we describe in Section 6.1.1 how deduplication works and in Section 6.1.2 we review the different confidentiality issues in data deduplication.

6.1.1 Data deduplication

Data deduplication is an emerging storage technology allowing to achieve storage space capacity optimization and is considered as the promising approach in storage technologies [47, 112].

Put simply, data deduplication keeps a single copy of repeated data. When a client wishes to store some piece of data, and if a copy of this data has already been saved in the system, then only a reference to this existing copy is stored at the storage server. No duplicate is created. This reduces the storage space consumption as references are in general smaller than data. It has been showed that depending on the application, data deduplication can save around 80% to 90% of storage space compared to the same application with no deduplication [47, 112].

Data deduplication can be distinguished at the level at which it is applied. Hereafter, we briefly describe the diverse forms of deduplication.

User granularity

Data deduplication can be classified according to its user granularity:

- **Intra-user:** Deduplication is performed only regarding the data previously stored by a user. For example, suppose a user Alice has already stored a data item D in the system, if another user Bob stores the same data item D in the system, a new copy of D will be stored. However, for subsequent requests for storing D by Alice or Bob, the deduplication will be applied.

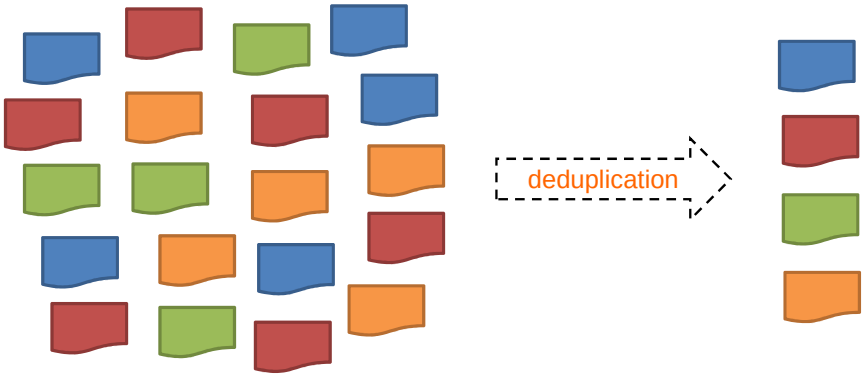


Figure 6.1: Deduplication: Only one copy of redundant data are stored in the system.

- **Inter-user:** In this case, the deduplication is performed on all the data stored in the system independently from the owner of the data items. For example, when Alice stores a data item D in the system, further, if Bob stores the same data item D, the deduplication will be applied. The inter-user deduplication is attractive because the storage space savings is higher than with an intra-user deduplication.

Data granularity

Data deduplication can also be applied at various level:

- **File level:** This method considers a whole file as a data item and only a single instance of a file is stored in the system [59, 18, 5]. However, this method does not take advantage of the similarities between different files. Assume a user who has stored a file in the system, modifies a little bit this file and stores the new version, this will result in two files saved in the system despite their similarities that could be leveraged by the deduplication for more storage space savings.

- **Block level:** This solution breaks a file into small blocks of fix-length [105] or variable-length [9, 95, 141] in order to detect duplicated blocks in the system. For example, when a user stores a file in the system, further, if he modifies this file and stores the new version, only the changed blocks of the file will be saved in the system. It allows more storage space savings than the file level deduplication because it takes advantage of similarities between files [48, 89]. The smaller the blocks are, the more storage space savings can be expected. However, this solution requires more resources because the indexes of blocks identifiers to maintain are much larger than in a file level deduplication [80, 144].

- **Content aware:** In contrast to a block level method, a content aware method breaks files into blocks taking into account the semantic of this file. With the understanding of the files semantics, creating blocks is more efficient and consumes less computing resources than a block level method. For example, to deduplicate emails, while a method not aware of the semantic of a mail may consume many resources to create multiple blocks, a method aware of the semantic of a mail can efficiently create blocks corresponding to headers, body, and attachments.

Location

Data deduplication can be executed at different locations, generally at the client-side or the server-side:

- **Server-side:** In this case, clients send data to the storage server which performs all the deduplication process in order to remove duplicates and stores only a single instance of each data item.
- **Client-side:** In this case, the deduplication is performed by the client before sending data to the storage server. Actually, clients compute and send the identifiers of the data items to the storage server. The storage server checks whether or not these identifiers exist in its index. If an identifier exists in the storage server index, it means that a copy of the corresponding data item exists in the system and the item is deduplicated. Otherwise, the client has to send the data to be stored. The client-side deduplication is very attractive because alongside the storage space savings, it also saves network bandwidth as data items are not sent if a copy already exists in the system.

Timing

Data deduplication may also differ depending at the moment when it is applied.

- **In-line**: The deduplication is applied while storing data in the system. It can be applied on the client-side or on the server-side.
- **Post-process:** In this case, data are deduplicated after they have been stored on storage nodes. Thus, it is applied on the server-side. The post-process avoid the backups latency that may create in-line deduplication on the storage server side due to the process to detect redundancy. However a post-process requires more available storage space on the storage server than an in-line deduplication as duplicates need to be stored for a certain time before the deduplication process starts [80].

6.1.2 Deduplication and confidentiality issues

A number of various attacks that malicious client can perform on system using data deduplication has been discussed in the literature [60, 62, 94, 122]. To better understand where the vulnerabilities of deduplication stand in order to propose a solution, we have studied these attacks and came up with the conclusion that these attacks can be summarized in three categories that we describe hereafter.

6.1.2.1 Manipulation of data identifiers

A common technique to deduplicate data is by hashing the content of the data and using this unique hashed value as the identifier of the data. Then the client sends this identifier to the storage server to determine whether such an identifier already exists in the system. An attacker can easily exploit this technique to perform various attacks

An attack based on data identifier manipulation is the *CDN attack* [62] that turns a cloud storage system into a Content Delivery Network (CDN) (See Figure 6.2). Suppose that Bob wants to share a file F with Alice. Bob uploads F to a cloud storage system and sends F identifier to Alice. When Alice receives it, she tries to upload F to the same cloud storage system by sending F identifier. The storage system will detect that this identifier already exists. Consequently, solely a reference meaning that Alice also owns file F will be stored in the system which is actually wrong. At this point, when Alice wants to download F, she just needs to request it from the cloud storage system. This attack has been popularized on Dropbox [44] by Dropship [43] a tool allowing to distributes files from between Dropbox accounts using only file identifiers. Note that Dropship is not functional anymore as Dropbox made some changes on their system because of this problem.

Another example of this kind of attack is the one called *poison attack* or *targeted collision* [122] in which, a malicious client uploads a piece of data (*e.g*, a file) that does not correspond to the claimed identifier (See Figure 6.3). Suppose that Bob wants to cheat Alice. If no control is

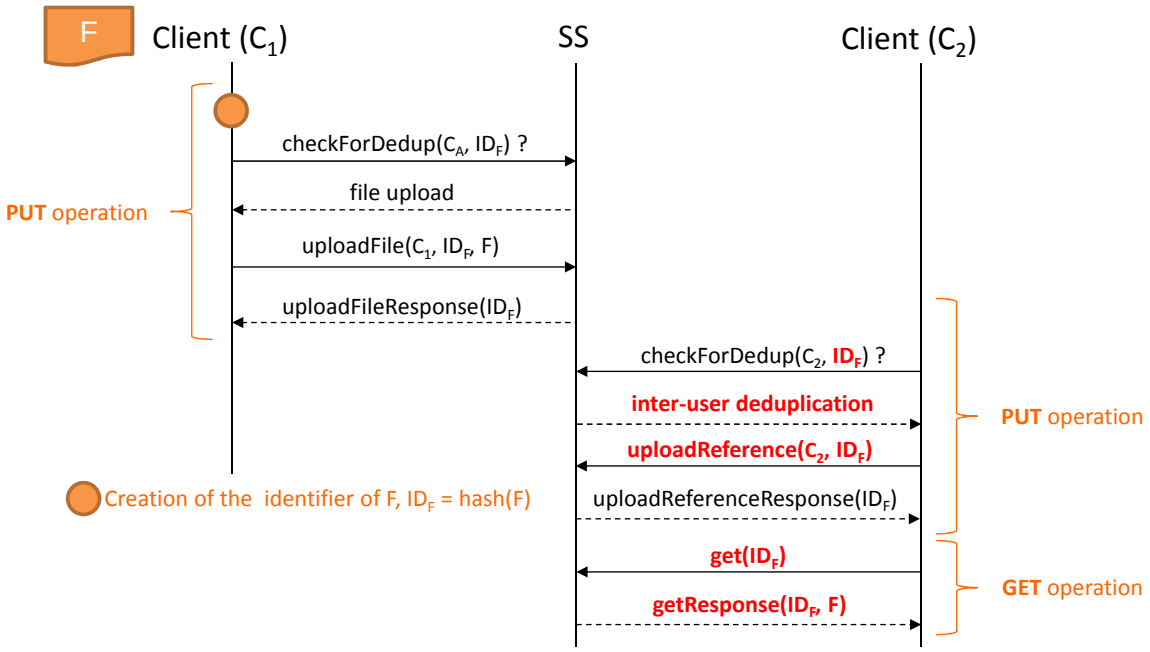


Figure 6.2: The CDN attack turns a cloud storage service into a Content Delivery Service using for free the resources of the CSP. The issue is that the storage service assumes that the client 2 has the file F just because he/she presents F identifier to the system.

made by the cloud storage system, Bob can upload a file F_1 with the identifier of a file F_2. Then, if Alice wishes to upload F_2 with its real identifier, the system will detect the existence of F_2 identifier in the system and will not store F_2. Rather, the system will store a reference meaning that Alice also owns file F_1 which is the file corresponding to the identifier of F_2 in the system. Later, when Alice will request the file corresponding to the identifier of F_2, the system will send F_1 to Alice.

6.1.2.2 Network traffic observation

Monitoring the network traffic on the client side gives an attacker the capability to determine whether an inter-user deduplication has been applied on a given piece of data (See Figure 6.4). Such an attacker can exploit this knowledge to identify data items stored in a cloud storage system by other users or even learn the content of these data items [12, 62]. For instance, to determine whether a file F is stored in a system, the attacker just tries to upload F and observes the network traffic from his/her device toward the storage server. If F is uploaded, it means that F does not exist in the storage system. The attacker is not even obliged to upload the complete file. He/she can prematurely abort the upload, so that the attack can be repeated later. On the other hand, if the file is not uploaded, the attacker can conclude that the file already exists in the system. This fact makes the storage system vulnerable to brute force attacks on file contents [12, 62]. For instance, to determine which copies of a movie exist on a specific cloud storage system, an attacker can upload each format or version of such a movie, and show that those that are not uploaded already exist in the storage system.

6.1.2.3 Backup time observation

Detecting an inter-user deduplication by observing the backup duration gives an attacker the ability to perform the same attacks as the ones done with network traffic observations (See Figure 6.4). This kind of attacks, also called *timing attacks*, have been used in several contexts such as hardware security tokens, network-based cryptosystems, Information Centric Networks (ICN), memory deduplication [73, 38, 114, 21, 139, 124, 92, 78, 49]. Depending on the context, timing attacks al-

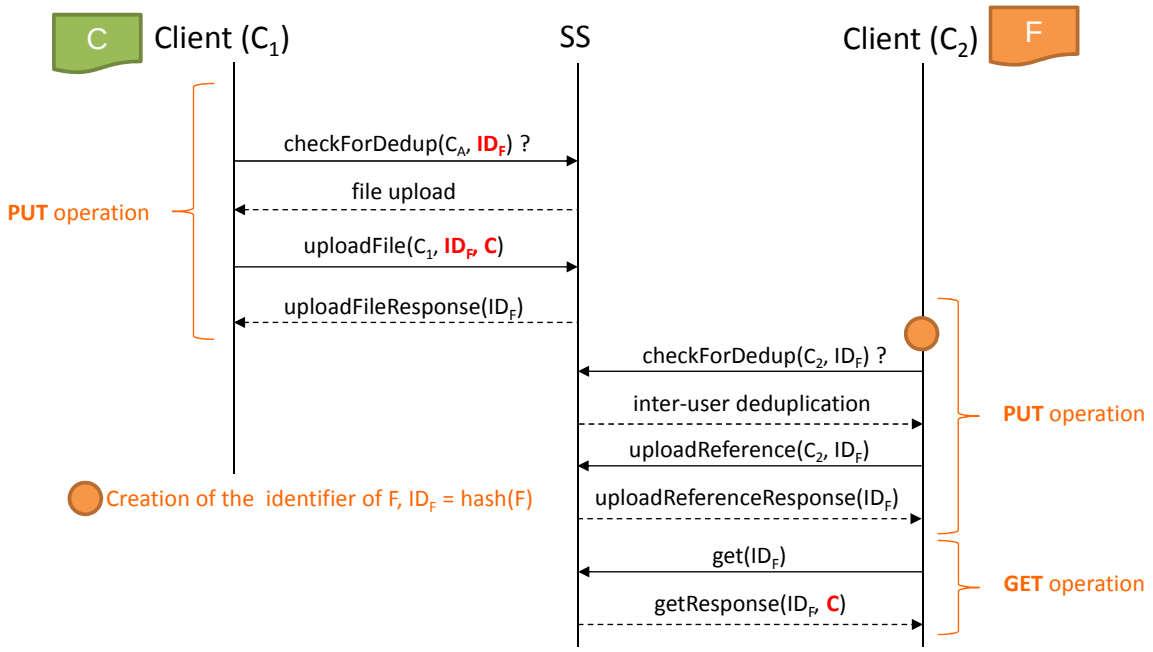


Figure 6.3: With the Poison attack also called targeted-collision attack, the malicious client can store a data item with the identifier of an other to create a collision. The problem is that there is no control on the correspondence of the file and its identifier.

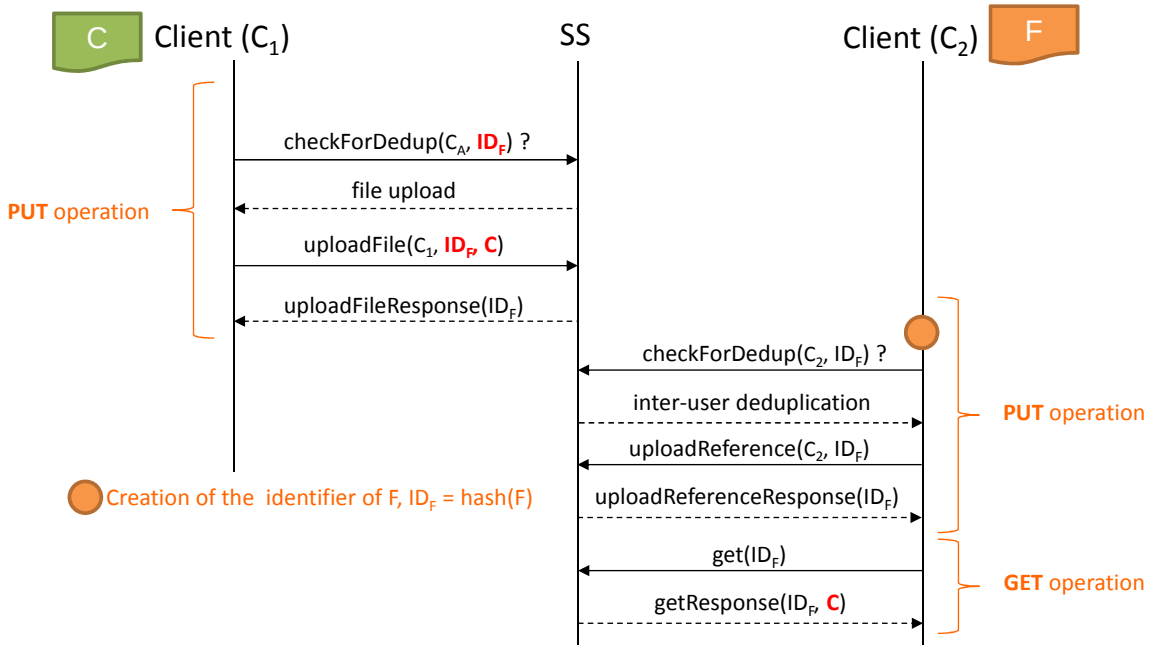


Figure 6.4: It is simple to identify files stored in the storage service just by trying to store a file and to observe the network traffic to see if the file is actually uploaded or by measuring the duration of the PUT operation. The problem here is that the inter-user deduplication is not transparent to clients.

lows to the attacker to compromise secret keys or users/systems privacy. Actually, systems where malicious clients can perform accurate timing measurements are potentially at risk against timing attacks. However, in a deduplication system, observing the duration of a backup operation is less

accurate than a network traffic monitoring as it depends on the size of the file and the state of the network. For small files, observation may not be accurate, while for larger ones, it gains in accuracy.

6.1.3 Our contribution

There have been lots of efforts devoted to deal with these issues and provide a secure and efficient data deduplication [12, 13, 39, 60, 62, 63, 82, 94, 96, 122, 140, 143]. However, we are not aware of any single solution that is capable of addressing, at the same time, the three types of attacks that malicious users can attempt on the deduplication system. This is our objective in our solution.

Our work takes place in the context of an Internet Service Provider (ISP) providing also the storage system[1]. That is to say, the ISP which is also the CSP has strong economical reasons to *(i)* save storage space and network bandwidth as it masters all the network and storage infrastructure, and *(ii)* provide a secure storage service to its consumers. Making clients save their network bandwidth is also a strong incentive as it improves the user Quality of Experience (QoE) of users. We propose a deduplication solution to build a storage system that positively answers the following three questions:

1. **Can we guarantee that the identifier used to trigger deduplication corresponds to the claimed data item?** In fact, it is important to avoid target-collision attacks. When the system deduplicates a data item we have to be sure that we are not introducing or creating a data collision. The natural approach in deduplication is to create data identifiers with a hash function that is pre-image resistant (*i.e.* given a hash value, it should be hard to find any file that hashes to that hash value), second pre-image resistant (*i.e.* given a file F_1, it should be hard to find a different file F_2 such that F_1 and F_2 hash to the same value), and collision resistant (*i.e.* it should be hard to find two different files F_1 and F_2 that hash to the same value). While this approach is needed to avoid or limit natural collision, it is not sufficient against targeted-collision attack.

2. **Can we determine that a client actually owns the data item corresponding to the identifier issued to the storage system?** It is important to provide a solution that answers positively to that question. In fact during a deduplication operation, if the client does not send the data item been deduplicated, it is impossible to know if that user were just in possession of the data item of just its identifier. If we ensure that a user really owns a data item before its deduplication, we can mitigate attacks such as the CDN one. In the literature, several solutions address this problem through a Proof of Ownership (POW) [39, 60, 94, 96, 143] that ensures that the client owns the data item corresponding to the identifier he/she presents to the storage system. With a POW, the client can prove to the system that he/she owns a data item without sending the entire data item to the storage server. However, we found that a POW alone is not sufficient as it answers only to one problem. For instance, a simple POW does not answer positively to the next question since a client knows when an inter-user deduplication is performed.

3. **Can we make the inter-user deduplication transparent (*i.e.* unnoticeable even in the case of backup time or network traffic observation) to clients and still provide network bandwidth savings?** Several solutions in the literature try to provide a secure deduplication solution while performing the inter-user deduplication on the client-side. We argue that this is not a good approach since having the capability to know whether an inter-user deduplication has been applied leaks some information about the file already stored in the cloud by other users. This leads to various type of attacks. In our work, we envision a solution where we can get the benefits of inter-user deduplication (*i.e.* the global storage space savings), and still get the benefits from network bandwidth savings. We took a different approach from the other solutions that consider only two locations where to perform dedu-

[1] For instance the french ISP Orange provides various cloud storage services.

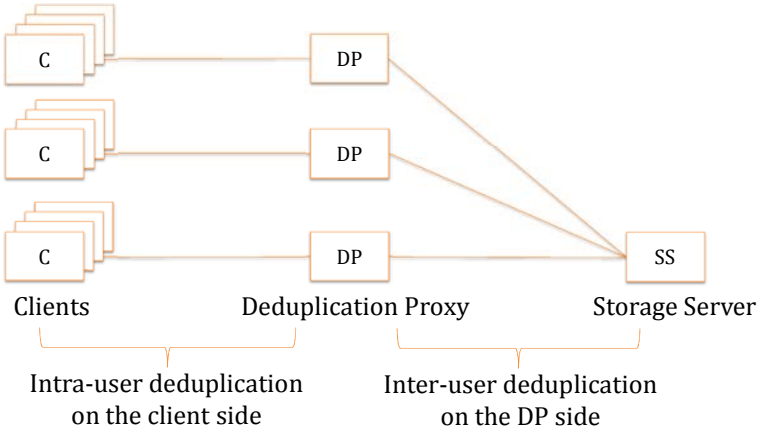


Figure 6.5: Architecture of the two-phase deduplication scheme.

plication (*i.e.* client-side and server-side). Thanks to the work on Mistore (See Chapter 4) we know that it is possible to put some storage intelligence in the network path between clients and a server, and we leveraged this fact to answer positively to that question.

Our deduplication scheme aims to be simple and robust against the aforementioned attacks while remaining efficient in terms of storage space and bandwidth savings for both clients and the CSP. We consider data deduplication at a file level granularity but our solution can be extended to the block level as well.

Specifically our approach is a two-phase deduplication that leverages and combines both intra- and inter-user deduplication techniques by introducing *deduplication proxies* (DPs) between the clients and the Storage Server (SS). Communications from clients go through these DPs to reach the SS which allows to split the deduplication process as illustrated in Figure 6.5.

The intra-user deduplication: This phase is performed by clients. It aims at saving clients network bandwidth by guaranteeing that each client uploads no more than once each of his/her file to a DP (*i.e.* per-client savings). In addition, we strengthen data confidentiality by letting clients encrypt their files before uploading them. Regarding security, we exploit the fact that the scope of the intra-user deduplication is limited to the files previously stored by the client trying to store a file. It allows the storage system to be protected against both CDN and network traffic observations attacks. Briefly, in the former case, each client uploads each of his/her files to his/her associated DP exactly once even if the same files already exist in the storage system (due to a prior upload by an other client). This accordingly prevents any client from determining whether a file already exists in the storage system through network traffic observations.

The inter-user deduplication: This phase is performed by the DPs. Alongside the economical benefits offered by this scheme thanks to global storage space savings (*i.e.* only a single copy of a file is stored independently of the number of owners of this file), it provides also global network bandwidth savings between DPs and the SS as a single copy of any file has to be sent to the SS. Note that to make indistinguishable the upload of a file to the SS from the sending of a reference — when the referred file already exists in the storage system — the concerned DP may introduce an extra delay before sending the notification to the client. The goal of adding this delay is to make similar the duration of storing a reference and storing the entire file. This aims to protect the system against attacks based on backup time observation.

Consistency of files and their identifiers: To protect the storage system against targeted-collision attacks, consistency between a file and its identifier is checked by the DPs before sending the file to the SS and before applying the inter-user deduplication (*i.e.* sending only a reference to the SS).

The remainder of this chapter is organized as follows. Section 6.2 details the system model and the assumptions we made about the system. Section 6.3 focuses on the two fundamental operations that allow to store and retrieve files in the system. Section 6.4 presents a performance evaluation of our solution, and Section 6.5 a discussion on security issues. Section 6.6 exposes the related work. Conclusion and future work are presented in Section 6.7.

6.2 System model

In this section, we describe the decision we took on the system model and the assumption we made about our solution in order to keep the focus on our contribution and make it easily comprehensible. Our system consists of the following three types of components:

- **Client (C):** Any authenticated user accessing the storage system.
- **Storage Server (SS):** A server in charge of storing and serving clients files. The storage server also maintains an index of all the files stored in the storage system and their owners.
- **Deduplication Proxy (DP):** A server associated to a given number of clients. Clients communicate with the storage server via their associated deduplication proxy. A deduplication proxy is involved in both the intra-user and the inter-user deduplication (See Section 6.3).

The orchestration of the proposed architecture is illustrated in Figure 6.5. The SS and the DPs are operated by the CSP.

6.2.1 Threat model

In order to focus on confidentiality issues, we suppose that the CSP is trusted to behave correctly to ensure the clients files availability, integrity, and long term durability. However it follows the honest but curious adversary model. Namely, it may take actions to disclose file contents.

Regarding clients, we assume that some of them are malicious. They may try to perform the types of attacks mentioned in Section 6.1.2. We suppose that there is no coalition between the entities provided by the CSP (*i.e.* the SS and the DPs) and the clients.

Finally, we consider that communication channels are reliable through the use of cryptographic protocols such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS). It avoids to deal with attackers that are not involved in the storage system so we can focus on the deduplication mechanism and the malicious clients of the storage system.

6.2.2 Data encryption

We make the hypothesis that entities have access to a hash function, denoted by `hash` in the following, that is pre-image resistant (*i.e.* given a hash value, it should be hard to find any file that hashes to that hash value), second pre-image resistant (*i.e.* given a file F_1, it should be hard to find a different file F_2 such that F_1 and F_2 hash to the same value), and collision resistant (*i.e.* it should be hard to find two different files F_1 and F_2 that hash to the same value). This hash function will be used to create data identifiers for example. We consider that there is no hash collision.

We assume that clients have the ability to encrypt and decrypt data with both asymmetric ($\texttt{encrypt}_{asym}$ and $\texttt{decrypt}_{asym}$) and symmetric ($\texttt{encrypt}_{sym}$ and $\texttt{decrypt}_{sym}$) cryptographic functions and manage their own private and public personal keys for the asymmetric encryption scheme.

We allow clients to encrypt their files before sending them over the network to be stored. However, there is some complications when client encrypt their files using a classic encryption scheme. In fact, if clients encrypt their files with personal keys, then the same file encrypted by several clients will result in different ciphertexts of the same file. Therefore, the CSP will not be able to detect duplicates and an inter-user deduplication would be impossible. To allow the inter-user deduplication on encrypted data and improve the storage space saving, an other encryption scheme is needed.

Fortunately there exists an encryption scheme called *convergent encryption* [12, 13, 42, 82, 140] that allows the inter-user deduplication on encrypted data. Specifically, to encrypt a file F with this scheme, the hashed value of F is used as an encryption key to encrypt F using a cryptographic symmetric function. Thus, the same file encrypted by different clients will result in equal ciphertexts. This allows the CSP to detect duplicates and apply an inter-user deduplication on encrypted files without any knowledge of their plaintext content.

In our approach, clients encrypt their files using the *convergent encryption* scheme to let clients encrypt files before uploading them.

6.3 System design

This section describes the `put` and `get` operations allowing clients to respectively store and retrieve their files from the cloud storage.

6.3.1 The `put` operation

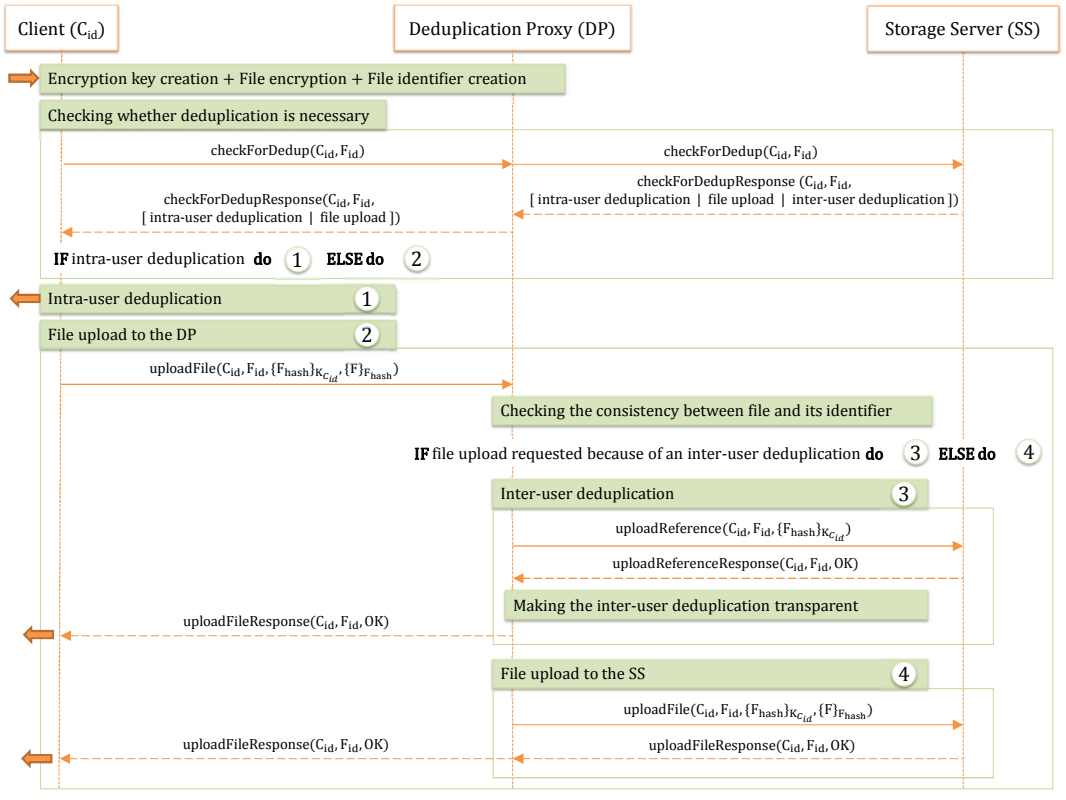


Figure 6.6: A `PUT` operation

When a client wishes to store a file in the storage system, he/she triggers a `put` operation. This operation, illustrated in Figure 6.6, involves two types of interactions, the first one between the client and a DP, and the second one between this DP and the SS. Table 6.1 gives the signatures and descriptions of the functions used in a `PUT` operation. In the following, the client invoking the `put` operation for a file F is denoted by its identifier C_{id}. Algorithm 13 gives the pseudocode of a `PUT` operation performed by the client. Before interacting with the DP, C_{id} builds the different parameters of the functions of the `put` operations. First, it creates the *encryption key* by hashing F content as follows,

$$F_{hash} \leftarrow hash(F). \text{ [Line 1 in Algorithm 13]}$$

Algorithm 13 Client: Function Put.
Put (C_{id}, F, K_C)

Input: The client identifier C_{id}, the public key of client, and the file F to store in the system.
Output: The client identifier C_{id}, the identifier associated to the file F, and the result of the operation.

1: $F_{hash} \leftarrow$ hash(F)
2: $\{F\}_{F_{hash}} \leftarrow encrypt_{sym}(F_{hash}, F)$
3: $F_{id} \leftarrow$ hash($\{F\}_{F_{hash}}$)
4: DP $\leftarrow$ getDP()
5: checkForDedupResponse $\leftarrow$ DP.checkForDedup(C_{id} , F_{id})
6: **if** checkForDedupResponse.result == file_upload **then**
7: $\{F_{hash}\}_{K_C} \leftarrow encrypt_{asym}(K_C, F_{hash})$
8: **return** DP.put(C_{id}, F_{id}, $\{F_{hash}\}_{K_C}$, $\{F\}_{F_{hash}}$)
9: **end if**
10: **return** {C_{id}, F_{id}, OK}

Then, C_{id} encrypts the file F by applying a symmetric encryption function on F using F_{hash} as a key leading to the ciphertext $\{F\}_{F_{hash}}$. We have

$$\{F\}_{F_{hash}} \leftarrow encrypt_{sym}(F_{hash}, F). \text{ [Line 2 in Algorithm 13]}$$

Finally, C_{id} creates F identifier F_{id} as

$$F_{id} \leftarrow hash(\{F\}_{F_{hash}}). \text{ [Line 3 in Algorithm 13]}$$

The previous steps allow different clients to create the same identifier for a given file, and enable to check the consistency between an encrypted file and its identifier by just comparing the hashed value of the encrypted file and the received identifier. Once client C_{id} has created the parameters of the `put` operation, he/she sends a `checkForDedup` request to the DP to inform it that he/she wants to store F. Algorithms 14 and 15 gives the pseudocode executed, during a `checkForDedup` request, respectively by the DP and the SS. This request is directly forwarded to the SS that checks whether a data deduplication and which kind of deduplication must be applied (*i.e.* either an intra-user deduplication or an inter-user one). This is achieved by looking for F_{id} and its potential owners in the file ownership index maintained by the SS. According to the outcome of this search, the SS sends one of the following three responses to the DP.

1. (C_{id}, F_{id}, `Intra-user deduplication`). C_{id} has already stored the file corresponding to F_{id}. Thus, by construction of our deduplication process, the file upload is not necessary; the DP directly forwards the response to C_{id}.

2. (C_{id}, F_{id}, `File upload`). In this case, the file corresponding to F_{id} does not exist in the storage system, which requires that C_{id} must upload it; the DP directly forwards the response to the client.

3. (C_{id}, F_{id}, `Inter-user deduplication`). The file corresponding to F_{id} has already been stored in the storage system by a client different from C_{id}. Thus, an inter-user deduplication must be applied by the DP. However, in order to hide this inter-user deduplication

Functions	Description
`hash(file)`	Creates a hash of the file passed in parameter
`encrypt`$_{sym}$`(secretKey, file)`	A symmetric encryption function taking as parameters the secret encryption key and the file to encrypt
`encrypt`$_{asym}$`(publicKey, file)`	An asymmetric encryption function taking as parameters the public encryption key and the file to encrypt
`getDP()`	Returns the deduplication proxy associated to the client executing this function. The DP returned assumed to be the closest one.
`checkForDedup(clientID, fileID)`	Checks whether deduplication (and which kind) is required when a client want to store a file. This function takes the client and the file identifiers as parameter.
`getFileIndex(clientID)`	Returns the list of files identifiers owned by a client. Its takes the client ID as parameter.
`put(clientID, fileID, encryptedKey, encryptedFile)`	Stores a file in the system. It takes as parameters the client and file identifiers, the encrypted key used to encrypt the plaintext file and the encrypted file.
`put(clientID, fileID, encryptedKey, encryptedFile)`	Stores a file in the system. It takes as parameters the client and file identifiers, the encrypted key used to encrypt the plaintext file and the encrypted file.
`getSS()`	Returns the Storage Server where to store a file.
`cache(checkForDedupResponse)`	Stores on the deduplication proxy the result of a checkForDedup request. It takes as parameter the response to the checkForDedup request.
`getCheckForDedupResponse(clientID, fileID)`	Gets the result of a checkForDedup request previously stored in the cache of the DP.
`getTransferTime(file, SS)`	Returns an average time required to transfer the given file to a given SS.
`putReference(clientID, fileID, encryptedKey)`	Stores in the system a reference of a file, namely the client and file identifier, and the encrypted encryption key.
`waitDelay(delay)`	Waits a delay of length *delay*
`update(clientID, fileID, encryptedKey)`	Updates the index of the system with the client and file identifier, and the corresponding encrypted encryption key

Table 6.1: API `put` operation

to the client, instead of directly forwarding this response to C_{id}, the DP sends him/her a `file upload` response, and memorizes that this upload is requested due to an inter-user deduplication (Line 4 in Algorithm 14).

Algorithm 14 Deduplication Proxy: Function checkForDedup.
checkForDedup (C_{id} , F_{id})

Input: The client identifier C_{id} and the identifier associated to the file F to store in the system.
Output: The client identifier C_{id} , the identifier associated to the file F, and `intra_user_deduplication` if the client already owns this file in the system, `file_upload` otherwise.

```
1: SS ← getSS()
2: checkForDedupResponse ← SS.checkForDedup(C_id, F_id)
3: if checkForDedupResponse.result in {inter_user_deduplication, file_upload} then
4:     cache(checkForDedupResponse)
5:     return {C_id, F_id, file_upload}
6: end if
7: return {C_id, F_id, intra_user_deduplication}
```

Algorithm 15 Storage Server: Function checkForDedup.
checkForDedup (C_{id}, F_{id})

Input: The client identifier C_{id} and the identifier associated to the file F to store in the system.
Output: The client identifier C_{id}, the identifier associated to the file F, and `intra_user_deduplication` if the client already owns this file in the system, `inter_user_deduplication` if the client does not own this file in the system but an other has already stored it, or `file_upload` if the file does not exist in the system.

```
1: if F_id not in index then
2:     return {C_id, F_id, file_upload}
3: else if F_id in getFileIndex(C_id) then
4:     return {C_id, F_id, intra_user_deduplication}
5: else
6:     return {C_id, F_id, inter_user_deduplication}
7: end if
```

Upon receipt of the DP message in response to a `checkForDedup` request, client C_{id} behaves as follows:

1. Receipt of an `Intra-user deduplication` response. It ends the `put` operation.
2. Receipt of a `File upload` response. The client uploads to the DP his/her client identifier, the encrypted file $\{F\}_{F_{hash}}$, the file identifier F_{id}, and the decryption key encrypted with its public key as follows,

$$\{F_{hash}\}_{K_{C_{id}}} \leftarrow encrypt_{asym}(K_{C_{id}}, F_{hash}) \text{ (Line 7 in Algorithm 13)},$$

which guarantees that $\{F_{hash}\}_{K_{C_{id}}}$ can only be decrypted using C_{id} private key. Client C_{id} waits for the response of the DP.

When the DP receives the `uploadFile` message from C_{id}, it applies a consistency check in order to prevent target-collision attacks. This is shown by the pseudocode in Algorithm 16. A targeted-collision is detected when the hash value of the encrypted file is not equal to the identifier received (Line 1 in Algorithm 16). In that case, the operation is aborted, and a notification is sent via an `uploadFileResponse` to client C_{id}. This ends the `put` operation. If the consistency check is successful and the file upload has been requested because of an inter-user deduplication, then only a reference (*i.e.* the client identifier C_{id}, the file identifier F_{id}, and the encrypted decryption key $\{F_{hash}\}_{K_{C_{id}}}$) is uploaded to the SS to be stored (Line 5 in Algorithm 16). Recall that the DP

has to ensure that this inter-user deduplication is unnoticeable to the client. Thus, before sending the notification (*i.e.* an `uploadFileResponse` message) to C_{id}, the DP delays it to make the duration of the whole `put` operation similar to a transmission of the file to the SS (Line 6 and 7 in Algorithm 16). The added delay is thus a function of the size of the file and the state of the network. On the other hand, if the file upload has been requested because that file does not exist in the storage system, then the `uploadFile` message is simply forwarded to the SS in the case where the consistency check has been successfully passed (Line 9 in Algorithm 16). Algorithms 17 and 18 shows the pseudocodes executed by the SS for storing a reference and a file. In both cases, the SS updates the index of the list of files belonging to the client with the new reference.

Algorithm 16 Deduplication Proxy: Function Put.
Put (C_{id}, F_{id}, $\{F_{hash}\}_{K_C}$, $\{F\}_{F_{hash}}$)

Input: The client identifier, the file identifier, the encrypted decryption key of the file F, and the file F encrypted.
Output: The identifier associated to the file F.

1: **if** F_{id} == hash($\{F\}_{F_{hash}}$ **then**
2: SS ← getSS()
3: checkForDedupResponse ← getCheckForDedupResponse(C_{id}, F_{id})
4: **if** checkForDedupResponse == inter_user_deduplication **then**
5: SS.putReference(C_{id}, F_{id}, $\{F_{hash}\}_{K_C}$)
6: $Time_{avg}$ ← getTransferTime($\{F\}_{F_{hash}}$, SS)
7: waitDelay($Time_{avg}$)
8: **else**
9: SS.put(C_{id}, F_{id}, $\{F_{hash}\}_{K_C}$, $\{F\}_{F_{hash}}$)
10: **end if**
11: **return** {C_{id}, F_{id}, OK}
12: **end if**
13: **return** {C_{id}, F_{id}, ERROR}

Algorithm 17 Storage Server: Function PutReference.
PutReference(C_{id} , F_{id}, $\{F_{hash}\}_{K_C}$)

Input: The client identifier, the file identifier, and the encrypted decryption key of the file F.
Output: The client identifier, the file identifier, and the result of the operation.

1: index.update(C_{id}, F_{id}, $\{F_{hash}\}_{K_C}$)
2: **return** {C_{id}, F_{id}, OK}

Algorithm 18 Storage Server: Function Put
Put (C_{id}, F_{id}, $\{F_{hash}\}_{K_C}$, $\{F\}_{F_{hash}}$)

Input: The client identifier, the file identifier, the encrypted decryption key of the file F, and the file F encrypted.
Output: The client identifier, the file identifier, and the result of the operation.

1: putReference(C_{id}, F_{id}, $\{F_{hash}\}_{K_C}$)
2: storageNodes.store(F_{id}, $\{F\}_{F_{hash}}$)
3: **return** {C_{id}, F_{id}, OK}

6.3.2 The `get` operation

The `GET` operation is invoked by clients to retrieve their own files from the storage system. Figure 6.7 illustrates the process of a `GET` operation. Our deduplication scheme pushes all the complexity to the `PUT` operation. Algorithms 19, 20, and 21 show the pseudocodes executed, during a `GET` operation, respectively by the client, the DP, and the SS. In `GET` operations, the DPs

Functions	Description
`getDP()`	Returns the deduplication proxy associated to the client executing this function. The DP returned is assumed to be the closest one.
`get(clientID, fileID)`	Get the file a client desires. It takes as parameters the client and the file identifiers.
`decrypt`$_{sym}$`(secretKey, file)`	An asymmetric decryption function taking as parameters the secret decryption key and the file to decrypt
`decrypt`$_{asym}$`(privateKey, file)`	An asymmetric decryption function taking as parameters the private encryption key and the file to decrypt
`getFileIndex(clientID)`	Gets a list of all the file identifiers of the files belonging to the client. It takes the client identifier as parameter.
`retrieve(fileID)`	Retrieves a file from the storage nodes. It takes the file identifier as parameter.
`getKey(clientID, fileID)`	Get the encrypted encryption key of a file with identifier `fileID` corresponding to the client with identifier `clientID`.

Table 6.2: API `get` operation

only forward the requests and responses they respectively receives from the clients and the SS. Specifically, upon receipt of a `GET` request from a client C_{id} about a file of identifier F_{id}, the DP simply forwards it to the SS which looks for F_{id} in its file ownerships index. If F_{id} is not found among the identifiers of files belonging to C_{id}, the SS sends a `getResponse` message to the DP with an error notification to terminate the `get` request. Otherwise, the SS sends a `getResponse` containing the ciphertext $\{F\}_{F_{hash}}$ corresponding to the identifier F_{id} and the encrypted decryption key $\{F_{hash}\}_{K_{C_{id}}}$ corresponding to the client C_{id}. In both cases, the DP only forwards the `getResponse` to the client C_{id} who will apply an asymmetric decryption function on the encrypted key using its private key $K^{-1}_{C_{id}}$ to recover key F_{hash}, where

$$F_{hash} \leftarrow decrypt_{asym}(K^{-1}_{C_{id}}, \{F_{hash}\}_{K_{C_{id}}}).$$

Finally, the plaintext of F is obtained with

$$F \leftarrow decrypt_{sym}(F_{hash}, \{F\}_{F_{hash}}).$$

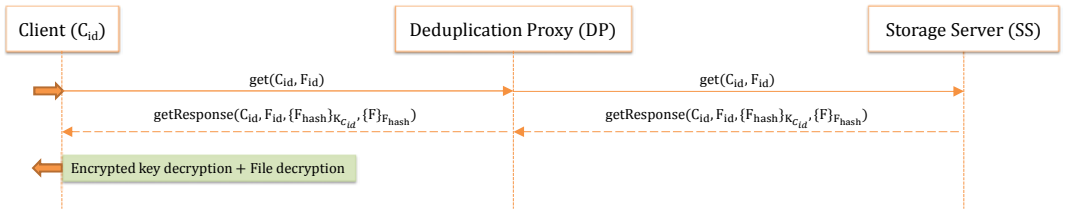


Figure 6.7: A `GET` operation.

Algorithm 19 Client: Function Get.
Get (F_{id})

Input: The identifier of the desired file.
Output: The file F.

1: DP ← getDP()
2: C_{id}, F_{id}, $\{F\}_{F_{hash}}$, $\{F_{hash}\}_{K_C}$ ← DP.get(C_{id}, F_{id})
3: F_{hash} ← decrypt$_{asym}$(K_C^{-1}, $\{F_{hash}\}_{K_C}$)
4: F ← decrypt$_{sym}$(F_{hash}, $\{F\}_{F_{hash}}$)
5: **return** F

Algorithm 20 Deduplication Proxy: Function Get.
Get (C_{id}, F_{id})

Input: The client identifier and the identifier of the desired file.
Output: The file identifier, the encrypted decryption key, and the file F encrypted. The function returns *NULL* if the file is not found.

SS ← getSS(C_{id}, F_{id})
1: **return** SS.get(C_{id} , F_{id})

Algorithm 21 Storage Server: Function Get.
Get (C_{id} , F_{id})

Input: The client identifier and the identifier of the desired file.
Output: The identifier of the file F, the encrypted decryption key, and the file F encrypted.

1: **if** F_{id} **in** getFileIndex(C_{id}) **then**
2: $\{F\}_{F_{hash}}$ ← storageNodes.retrieve(F_{id})
3: $\{F_{hash}\}_{K_C}$ ← index.getKey(C_{id}, F_{id})
4: **return** $\{C_{id}, F_{id}, \{F_{hash}\}_{K_C}, \{F\}_{F_{hash}}\}$
5: **end if**
6: **return** $\{C_{id}, F_{id}, \text{error}\}$

6.4 Implementation

We have implemented two storage system prototypes to compare the performance overhead of our proposition with respect to a classic storage system with no data encryption. Specifically, we have developed a classic storage system with a client and a storage server software modules, and one implementing our proposition with a client, a deduplication proxy and a storage server software modules. All these software modules are implemented in python 2.7.6 and access the pycrypto library for the cryptographic operations. We use the SHA256 algorithm as the hash function, RSA1024 for asymmetric encryption operations, and AES for the symmetric encryption operations with keys that are 256 bits long. The storage servers use the MongoDB[2] database to store the metadata of the stored files as well as files owners. The software modules are executed on three different virtual machines (VMs) running on Ubuntu 12.04.4 LTS with 1GB of memory and an AMD Opteron(TM) octa-core Processor 6220@3GHz. The network topology is as follows: the VM executing the DP software is located on the network path between the VM running the different clients modules and the one executing the different SSs modules.

Figure 6.8 and Figure 6.9 compare the different costs induced by the `put` and `get` operations in a classic storage system and in the one we propose. Each result regarding both `put` and `get` operations is the average of 1000 experiments run with files whose sizes range from 2MB to 64MB. The first straightforward observation that can be drawn from both figures is that the cost of both operations fully depends on the size of the stored files. In `put` operations, the overhead in

[2]http://www.mongodb.org/

our scheme is due to the encryption key creation, the encryption key encryption[3], the file encryption, the consistency check of a file and its identifier performed by the DP module, and the communication overhead introduced by handling the file by the DP. On the other hand, for `get` operations, the overhead in our proposition comes from the encryption key decryption[3], the file decryption, and the communication overhead of the DP. However, this overhead is small, as it entails around 0.5s on average per file of 64MB length. Actually, most of the overhead incurred in our solution is due to file encryption during a `put` operation and file decryption during a `get` operation. Indeed, up to 12-13s are needed on average for encrypting/decrypting a file that is 64MB length.

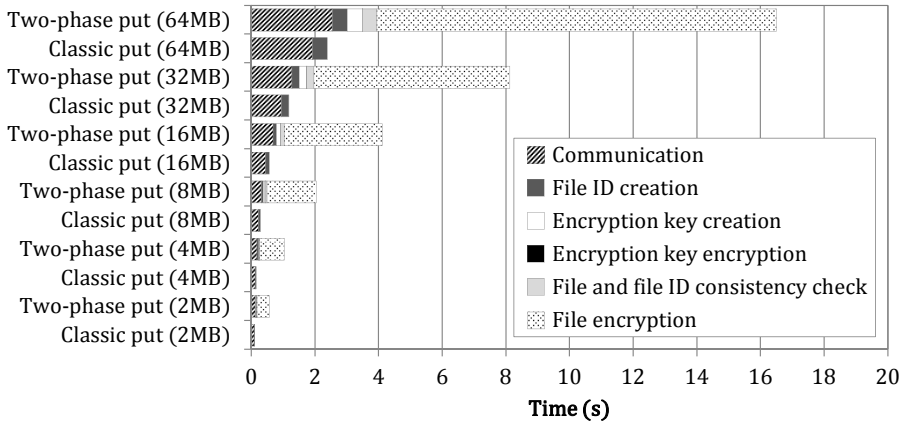


Figure 6.8: Average delays observed during a `put` operation in a classic system with no data encryption and in our system with data encryption. Most of the overhead introduced by our solution is due to cryptographic operations which are necessary to ensure data confidentiality.

Figure 6.10 gives a performance comparison between an inter-user deduplication applied in a classic scheme and the intra- and inter-user deduplication applied in our proposition. The inter-user deduplication experiments consist in storing files at a given client and having one thousand clients accessing them through the `put` operation. The intra-user deduplication experiment consists in storing thousand times the same files at a single client. The results plotted in Figure 6.10 represent the average duration values of the these operations. We observe that the intra-user deduplication in our scheme consumes more communication resources than an inter-user deduplication in a classic scheme due to the use of a DP (around 18ms of overhead for a file of 64MB length). However most of the overhead comes from the encryption key creation, the encryption key encryption[3] and the file encryption which depends on the file size. We also observe that operations involving the inter-user deduplication in our scheme have higher durations than the ones involving an intra-user deduplication. This is due to *(i)* the overhead of communications when applying the inter-user deduplication because the file has to be uploaded to the DP and then a reference is uploaded to the SS and *(ii)* the added delay by the DP to make unnoticeable the inter-user deduplication to client. Actually in our scheme, a `put` operation involving an inter-user deduplication has a similar duration than a file upload from the client to the SS.

6.5 Discussion

In this section, we discuss the impact of the proposed solution regarding the data security and the network bandwidth savings.

[3]The encryption key encryption/decryption is negligible (around 0.75 ms) and is not visible on Figures 6.8-6.10 because of the figure scale.

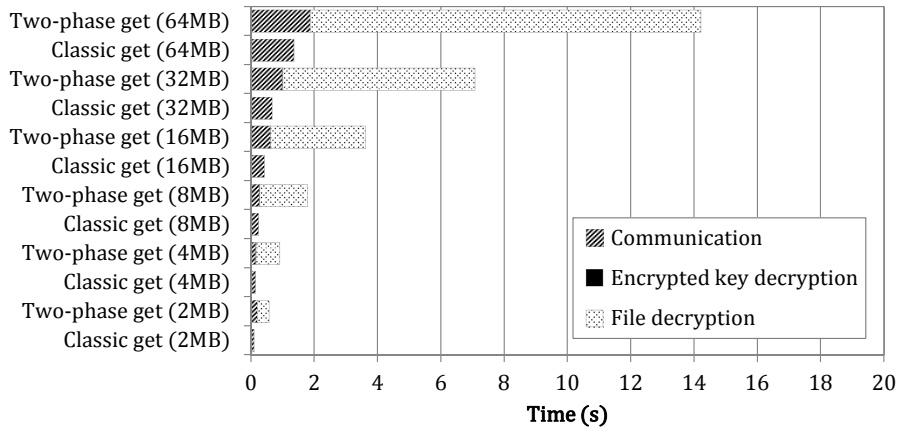


Figure 6.9: Average delays observed during a `get` operation in a classic system with no data encryption and in our system with data encryption. Most of the overhead introduced by our solution is due to cryptographic operations which are necessary to ensure data confidentiality.

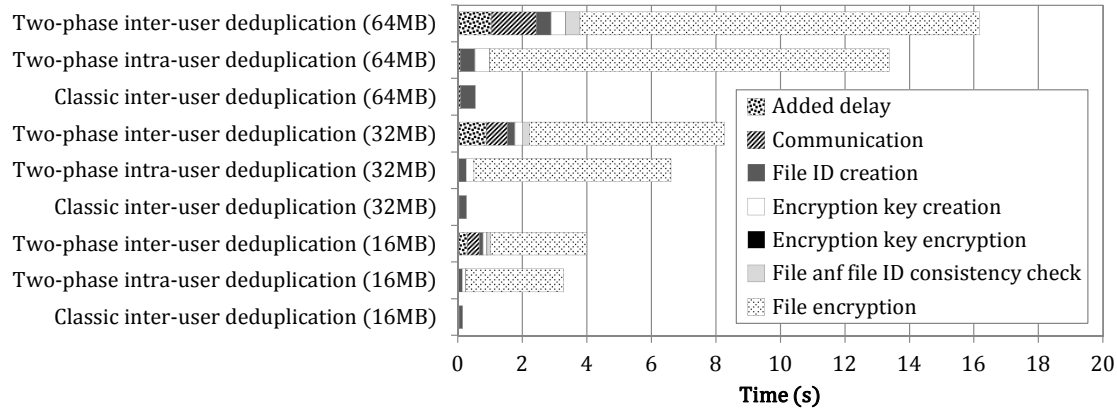


Figure 6.10: Performance comparison between a classic system with no data encryption where an inter-user deduplication is applied upon a `put` operation and our proposition with data encryption where an intra- and inter-user deduplication are applied. Most of the overhead introduced by our solution is due to cryptographic operations which are necessary to ensure data confidentiality.

6.5.1 Security against malicious clients

As previously described our solution aims at protecting the storage system against the manipulation of data identifiers, network traffic observation and backup time observation (See Section 6.1). This is achieved by placing deduplication proxies between clients and the storage server, and by running a two-phase deduplication scheme.

Now, by taking advantage of the intra-user deduplication performed between clients and their DP, from a client point of view, the visibility of what is stored in the storage system is limited to his/her own account. If a client has not previously stored its file then he/she must upload it to his/her associated DP, which protects the system against the CDN attacks and the attacks based on network traffic monitoring.

In order to prevent the targeted-collision attacks, the DP performs a consistency control on both the file and the identifier received from a client before sending this file to the storage server or before applying the inter-user deduplication.

To make the inter-user deduplication unnoticeable to clients in order to face the attacks based on backup time observation, the DP adds some delay before terminating the `put` operation (*i.e.* before sending the `uploadFileResponse`). Recall that a delay may be added only when an inter-user

deduplication is applied. In addition, by using a convergent encryption scheme, the CSP can perform an inter-user deduplication on client data without any access to their plaintext and achieve global storage space savings.

6.5.2 Impact of the localization of the deduplication proxy

The benefits of our proposition regarding the network bandwidth savings highly depends on the localization of the DPs in the storage infrastructure which may also impact the level of security of the solution. In the following, we briefly discuss the pros and cons of the DPs localization.

Deduplication proxies collocated with clients devices: If deduplication proxies are hosted by clients, then both intra- and inter-user deduplication need to be applied at the client sides, which provides global network bandwidth savings. However, this gain comes at the expense of data security. Indeed, clients may observe the network traffic between the DP and the SS, or have the opportunity to corrupt or replace the DP software. Such a solution protects efficiently the storage system solely against attacks based on backup time observation.

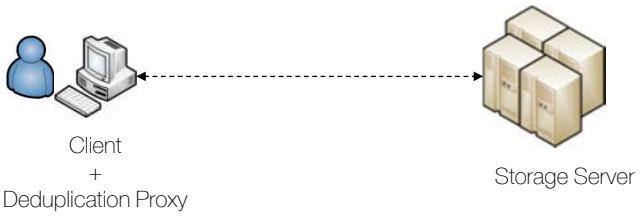


Figure 6.11: If the Deduplication Proxy is located on a client device there is some security issues as it is possible to observe the network traffic between the Deduplication Proxy and the Storage Server.

A deduplication proxy collocated with the storage server: If the DP is physically collocated with the storage server, then even if clients do not upload more than once the same file to that location thanks to the intra-user deduplication, duplicates of a file owned by other clients will still have to be transferred over the network to reach that location before the inter-user deduplication can be applied. This is clearly not the most efficient choice in terms of bandwidth savings. On the other hand, data security is not affected by this solution.

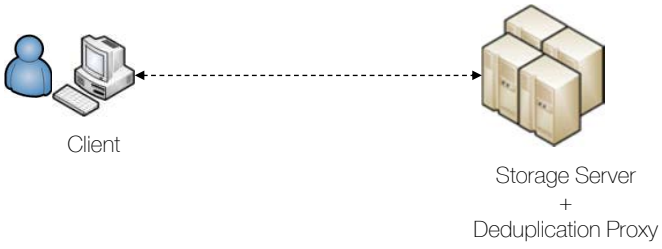


Figure 6.12: If the Deduplication Proxy is located on the Storage Server, there is some security issues as it possible to observe the network traffic between the Deduplication Proxy and the Storage Server.

Deduplication proxies hosted by independent nodes: We suggest that the two-phase deduplication relies on independent nodes to host the DPs. We argue that such nodes should be secure, physically out of reach of clients so that they can not observe the network traffic between a DP and the SS, but at the same time close enough to clients in order to improve the network bandwidth

savings. We believe that to locate such nodes one could adopt a similar strategy to what is done in the field of Content Delivery Network (CDN) [99]. In order to minimize data access latencies and network bandwidth consumption toward content servers, cache servers are deployed in strategic locations in the Internet infrastructure as close as possible to the clients to satisfy their requests close to them [65]. For instance, an interesting approach would be to use the Points of Presence (POPs) of the ISP as secure nodes to host the DPs and associate them to clients accessing to Internet through them [88]. This might also allow CSPs to cache files in the DPs in order to minimize the access latencies of subsequent `get` operations on these files.

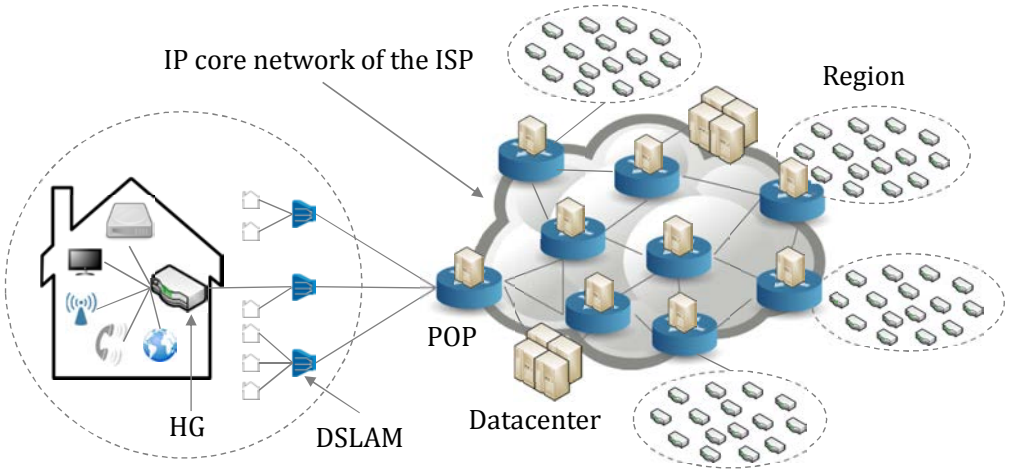


Figure 6.13: Simple overview of a DSL infrastructure of an ISP.

6.6 Related works

6.6.1 Storage cost optimization

Several techniques of storage capacity optimization have been introduced in order to reduce the storage cost. We can distinguish two approaches, the first one is focused on a better use of the available storage space in the architecture with technique such as storage tiering or thin provisioning, and the other approach targets the reduction of the amount of data to be stored with technique such as data compression, delta encoding, or deduplication. Data reduction techniques also improve the users Quality of Experience (QoE) as they can also reduce the amount of data that have to be transferred over the network.

The data deduplication differs from other storage cost optimization techniques such as data compression [77] and delta compression [66] as it can be applied not only on one file but also on all the file stored in the system without regards of their owner (*i.e.* the inter-user deduplication). However, these techniques are compatible and it is possible to combine them in a system, for instance, a file may be compressed before been deduplicated.

Here after we present briefly some of these techniques.

Thin provisioning

In a data management system, the provisioning allocates storage space for an application or service. Traditionally, full storage devices are allocated even if the application does not use immediately all the storage space that is allocated. So if several applications are located in the same architecture, this results in a large storage capacity that is most of the time unused. A thin provisioning technique combines different storage devices into a logical or virtual device so you no longer have to consider

a maximum or minimum number of storage devices allocated to an application or service [45]. It allocates only the storage space needed to an application and in the case of a need of more storage space, more storage space is allocated transparently. Thin provisioning actually leverages unused storage space to be used by many applications.

Storage tiering

According to an SNIA study, in 100TB of data, 80% are inactive or rarely accessed. In fact when data get old, they become rarely accessed and old data are the fastest growing part in a storage system. However active and inactive data may have to be kept in the storage architecture and the retention period may be long in order to meet regulatory requirements for example or to be used for trend analysis purpose.

Storage tiering actually relies on a well thinking data placement on nodes to optimize performance, availability, and recovery while minimizing the cost of the overall storage architecture [6]. To achieve this, it is essential to know the usage characteristics of the data in order to create a classification of them. Usually it is the applications using these data that drive this classification and the tiers configuration which is based on access speed, capacity, cost. Note that depending on the life-cycle of the data, they can be moved from one tier to an other. In general data that are critical and the most accessed are stored in a tier with fast and expensive storage devices such as Solid State Drives (SSD) and data that are infrequently accessed are located on tiers with slower and cheaper storage device but have larger storage capacity such as tape. Figure 6.14 shows different configuration of tiers and the reduction of the cost using storage tiering. The storage tiering is a cost-efficient alternative to simply add the same disks in the storage architecture to face the increasing storage capacity demand.

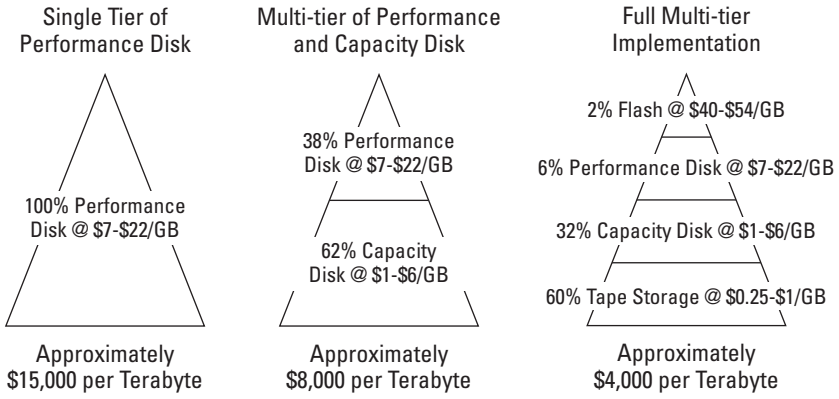


Figure 6.14: Tiered-storage cost savings: *Source Horison Information Strategies 2010.*

Erasure coding

Storage systems usually rely on data redundancy to provide reliability, namely, availability and long term durability for clients data [26]. A promising alternative to simply create additional identical copies of data is the technique of erasure code coming from the mathematical fields. The interesting part of erasure code is that they can provide the same level of reliability as a simple replication while requiring less storage space. Basically, to create redundancy with an erasure code, a data is fragmented, let say in k source fragments. Then these k fragments are encoded in n fragments that will be stored in the storage system with n >= k. To recover the original data, a decoding process is required in which only a subset of the n encoded fragments, generally a minimum number of k,

is needed. Erasure coding is more complex to implement than the simple replication but due to its economic advantages over the simple replication in terms of storage space consumption, it is gaining in popularity in both commercial and research storage systems.

Data compression

It is a technique that try reduce the size of a file on a write operation, this is the compression phase [77]. To achieve this, redundant parts of a file are replaced with smaller representations, links or references for example. Thus storage space when storing the file on disks as the compressed file is smaller and in the case where the file has to be sent over the network, data compression helps to save network bandwidth. Compression may be applied on files or on blocks of files. Data compression can be lossy or lossless [113].

A Lossless data compression creates smaller data from original data with a write operation. A lossless data compression is reversible. To recover the original data, usually in a read operation, a decompression phase is required in order to transform the compressed file into the original file.

A lossy data compression is irreversible. This technique actually eliminates nonessential information in a given file in order to get a compact and smaller representation of the original file. The file must tolerate some loss of information. Lossy data compression algorithms are sometimes used to compressed image, audio, or video (JPEG, MP3, MPEG, etc) where loss of information may not be detectable to human.

Delta-encoding

This technique is also known as delta compression or differential compression and most of the times used to save storage space and network bandwidth when files are to be sent between entities that may already own similar or identical files [66]. Then, only the differences will be sent and stored. Delta-encoding is widely used in distribution of software update or revision control systems, remote files synchronization, and also to improve the latency of web accesses [128, 131, 103, 23, 126, 108]. Delta encoding has also been combined with data compression to improve web performance [91].

6.6.2 Deduplication and data security

To deal with the security issues introduced by a client-side and inter-user deduplication, a method called randomized solution is proposed in [62]. This solution defines a random threshold for each file in the storage system which corresponds to the number of different clients that have uploaded this file. Each time a client uploads for the first time a file, a counter associated with this file is created and initialized to 1. As long as the counter is under the threshold, deduplication is not applied. When the counter reaches the threshold, the inter-user deduplication is activated and applied to all the subsequent uploads of that file. This solution protects a file as long as its threshold is not reached. As soon as the system enables deduplication on a file, it becomes vulnerable.

In [122], the authors propose a deduplication method that is secure against targeted-collision attacks. Similarly to our method, a file identifier is a hash value of the file content so it is simple to check the consistency of a file and its claimed identifier. However, this method does not address the others types of attacks mentioned in Section 6.1.2.

In [63], the authors propose a gateway-based deduplication scheme which relies on users gateways to run the deduplication process. In order to mitigate the attacks based on network traffic monitoring, gateways send a random network traffic compounded of encrypted packets with a Time-To-Live (TTL) parameter equal to 1. On the other hand, this method does not take into account neither backup time observation attacks, nor the possible corruption of the deduplication software module by the gateway owner.

Several additional works suggest the notion of Proof Of Ownership (POW) [39, 60, 94, 96, 143] to face the attacks based on manipulation of data identifiers. A proof of ownership allows a client to prove that he/she actually owns the file corresponding to the identifier presented to the storage server. However, it does not address the issues caused by attacks based on network traffic or backup time observation that are performed by adversaries that actually own the files corresponding to the identifiers that they present to the storage server.

Some works rely on specific encryption schemes to provide data confidentiality and deduplication against honest but curious CSPs [12, 13, 82, 140]. Table. 6.3 shows a comparison of some related work and our proposition regarding the storage space and network bandwidth savings, and the protection against the attacks mentionned in Section 6.1.2.

Compared to all the previously cited works, our two-phase deduplication method addresses all the attacks presented in Section 6.1.2, and to the best of our knowledge this is the first one to achieve it.

6.7 Summary and perspectives

In this chapter, we have presented a two-phase deduplication scheme that *(i)* ensures that a client actually owns the file he/she wants to store by applying an intra-user deduplication on the client side *(ii)* ensures that a file corresponds to its claimed identifier through a control by a deduplication proxy located between clients and the storage server and *(iii)* applies an inter-user deduplication on the deduplication proxy side that makes this inter-user deduplication unnoticeable to clients by adding some delay to `put` operations so that the length of a file upload is indistinguishable from an upload of its reference. Our method provides protection against attacks from malicious clients, global storage space savings to the CSPs thanks to the inter-user deduplication, per-client bandwidth network savings between clients and the deduplication proxies, and global network bandwidth savings between the deduplication proxies and the storage server.

In the scheme presented in this chapter, clients send their decryption key encrypted with personal key to be stored in the system. So, for a file owned by n clients for example, the system will have to store n ciphertext which represent the same decryption key. While these keys are small, we think that for future works, it would be interesting to extend our solution so that the encrypted decryption keys could also be deduplicated without jeopardizing security properties.

For future works, it would also be interesting to extend the deduplication in our scheme to a block level granularity. An other direction for future works is to address the confidentiality issues against attacks that can be performed by a CSP that is honest but curious because there is a potential threat regarding data confidentiality when an inter-user deduplication is provided. In fact allowing the inter-user deduplication on encrypted files gives the opportunity to determine that two equal ciphertexts originate from the same plaintext. This can be exploited by the CSP since given a file or just its identifier, the CSP can determine whether this file already exists in the storage system and identify the clients who have uploaded it (See Figure 6.15). Moreover the CSP is able to perform brute force attacks on file contents even if their files are encrypted with a convergent encryption [12, 140]. Recall that in our proposition, the inter-user deduplication is unnoticeable to clients so these limitations are only applicable to the CSP. Nevertheless, one may find in such an approach an interesting property for the CSP to detect illegal files in its storage system or/and to respond to judicial inquiries for example.

A strategy to protect files against honest but curious CSP would be to encrypt files using private encryption schemes. For example, individuals may use personal keys in the file encryption process. In the case of a group of clients or an enterprise, a key server that does not belong to the CSP can be used to manage the encryption keys so that all the clients of the key server will encrypt a given file with the same specific key [12, 82]. However it will make possible the inter-user deduplication solely within the members of the group or the enterprise and would reduce the storage space savings

Table 6.3: A comparison between some related works and our proposition.

Approaches	**Storage space savings**	**Network bandwidth savings**	**Protection against clients attacks based on data identifier manipulation**	**Protection against clients attacks based on network traffic observation**	**Protection against clients attacks based on backup time observation**	**Protection against honest but curious CSP**
Harnik *et al.* [62]	Global savings	Global savings	No more protection when the inter-user deduplication is enabled	No more protection when the inter-user deduplication is enabled	No more protection when the inter-user deduplication is enabled	NO
Storer *et al.* [122]	Global savings	Global savings	Protection only against targeted collision attacks	NO	NO	NO
Proof of ownership [39, 60, 96, 143]	Global savings	Global savings	YES	NO	NO	NO
Heen *et al.* [63]	Global savings	No bandwidth savings between the client and the gateway but global bandwidth savings between the gateway and the storage server	Limited to the ability of attackers to monitor the traffic going through the gateway	Limited to the ability of attackers to corrupt or replace the software module in the gateway	NO	NO
Bellare *et al.* [12]	Global savings only for the group of users using their client solution	Global savings only for the group of users using their client solution	NO	Online brute force attacks are slowed using a requests rate limitation	Online brute force attacks are slowed using a requests rate limitation	YES
Xu *et al.* [140]	Global savings	Global savings	YES	NO	NO	YES
Liu *et al.* [82]	Global savings only for the group of users using their client solution	Global savings only for the group of users using their client solution	NO	NO	NO	YES
Our proposition with DP on the client	Global savings	Global savings	NO	NO	YES	NO
Our proposition with DP on the storage server	Global savings	Per-client savings	YES	YES	YES	NO
Our proposition with DP on an independent node	Global savings	Per-client savings between clients and the DP, and global savings between the DP and the storage server	YES	YES	YES	NO

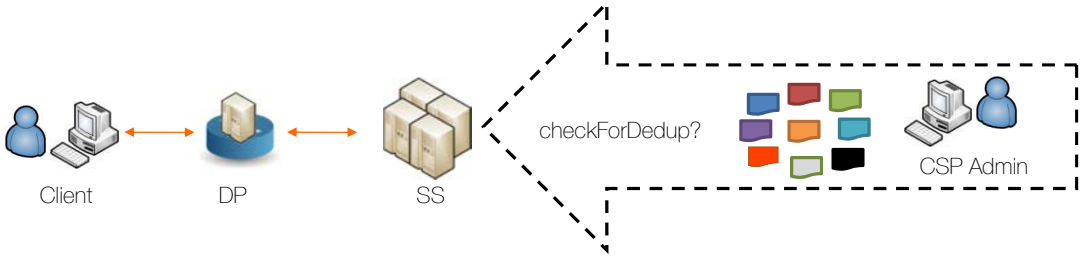


Figure 6.15: Security issues against an honest but curious CSP.

of the CSP since a given file encrypted with different keys would result in different ciphertexts. A data deduplication process independent of the owners (*i.e.* individuals, groups or enterprises using different key servers) would be impossible.

7

An enterprise secure outsourcing method with deduplication support

While the adoption of cloud storage services to outsource every type of data is steadily growing, enterprises are still reluctant to outsource completely all their data to the cloud because its benefits come with some inherent risks implied by the fact full control on data is given to the CSPs. In this chapter, we present a solution allowing enterprises to outsource data to the cloud while keeping control on them. Our solution leverages a cloud gateway located at the enterprise premises that aims to provide control on data stored in cloud, improve data access performance, optimize storage and network bandwidth usage, and provide data confidentiality. To achieve that this cloud gateway implements a metadata management on-premises as well as features such as caching, flexible data placement over multiple cloud storage services, encryption, and the secure two-phase data deduplication presented in Chapter 6.

Contents

7.1 Introduction

While the adoption of cloud storage services to outsource every type of data is steadily growing, its benefits come with some inherent risks implied by the fact full control on data is given to the Cloud Storage Providers (CSPs) [71, 107]. Here after some factors reducing the adoption of cloud storage services:

- **Security issues:** Outsourcing data gives a full control to the CSPs over the data being stored. Clients (*e.g.* individual, scientists, enterprises, governmental agencies, etc) may want to restrict the control or access of the CSP to their sensitive data in order to preserve their confidentiality.

- **High data access latencies:** Moving data to the cloud implies that data access will be performed over the Internet so the performance are reduced due to network latencies. This can be a major drawbacks especially for data that are intensively shared and used in the enterprise.

- **Vendor locking:** Not all the CSPs use technologies that are seamlessly interoperable so when the service of a specific CSP is chosen, over the time, clients may become locked into to a specific solution. Migrate data from one CSP solution to another may be difficult. Another concern when using a single CSPs is that clients rely on the reliability of that CSP solution. In fact, even if CSPs claim to provide an always on availability, outages and unavailability may happen making clients not able to access to their data [1]. Cloud storage services may also be shutdown (*e.g.* Nirvanix and Wuala) requiring clients to retrieve their data or migrate them to another Cloud Storage service which can be complex to perform depending on the amount of data being involved.

In this chapter we describe an enterprise data outsourcing strategy dealing with the issues previously mentioned. This solution aims to provide full control on the outsourcing strategy to the enterprise (*i.e.* CSP selection, how data are stored on these public CSPs, etc) and aims to be transparent to the clients (*i.e.* the enterprise employees).

Our solution relies on a physical appliance called Cloud Gateway (CG) located at the enterprise premises. The Cloud Gateway implements a caching system as well as the two-phase data deduplication to improve data access performance and reduce storage and network bandwidth costs. The two-phase data deduplication protects the system against malicious clients. The Cloud Gateway deals also against timing attacks that may be attempted by malicious clients. Timing attacks have been used in several contexts such as hardware security tokens, network-based cryptosystems, Information Centric Networks, memory deduplication (ICN) [73, 38, 114, 21, 139, 124, 92, 78, 49]. Depending on the context, timing attacks allow the attacker to compromise secret keys or users/systems privacy. Systems where malicious clients can perform accurate timing measurements of read/write operations are potentially at risk against timing attacks. To prevent these attacks, in our system, the Cloud Gateway adds an extra delay if necessary on read operations to protect the system against timing attacks on cached data.

The Cloud Gateway also adds an additional encryption layer for data outsourced in public clouds and keeps the associated metadata (*i.e.* information about data and clients) on-premises. The additional layer of encryption/decryption is transparent to clients. This allows to separate the storage of data in public CSPs and their metadata to improve the confidentiality against honest but curious CSPs as they will manage outsourced data without knowledge of their associated metadata (*e.g.* clients, data type, data name, etc.).

The remainder of this chapter is organized as follows. Section 7.2 describes the system model assumed. Section 7.3 describes the architecture of our solution. Description of `put` and `get` operations is given respectively in Section 7.4 and Section 7.5. Related works are presented in Section 7.6. We summarize this chapter and give perspectives in Section 7.7.

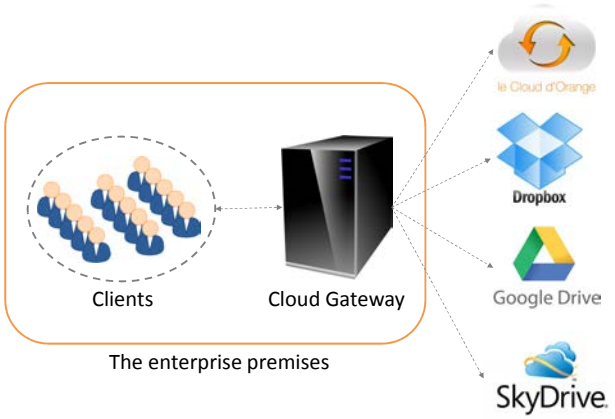


Figure 7.1: The propose architecture

7.2 System model

We suppose that CSPs follows the honest but curious adversary model. Namely, CSPs may take actions to disclose file contents. Regarding clients, we assume that some of them are malicious. They may try to perform the types of attacks mentioned in Section 6.1. We suppose that there is no coalition between CSPs and the clients. Finally, we consider that communication channels are reliable through the use of cryptographic protocols such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

We make the hypothesis that clients and the Cloud Gateway have access to a hash function, denoted by `hash` in the following, that is pre-image resistant (*i.e.*, given a hash value, it should be hard to find any file that hashes to that hash value), second pre-image resistant (*i.e.*, given a file F_1, it should be hard to find a different file F_2 such that F_1 and F_2 hash to the same value), and collision resistant (*i.e.*, it should be hard to find two different files F_1 and F_2 that hash to the same value). Thus we consider that there is no hash collision. We assume that entities have the ability to encrypt and decrypt data with both asymmetric ($\texttt{encrypt}_{asym}$ and $\texttt{decrypt}_{asym}$) and symmetric ($\texttt{encrypt}_{sym}$ and $\texttt{decrypt}_{sym}$) cryptographic functions and manage their own private and public personal keys for the asymmetric encryption scheme.

In our approach, clients may encrypt their files using a convergent encryption scheme [42, 140, 82, 12, 13]. Specifically, to encrypt a file F with this scheme, the hashed value of F is used as an encryption key to encrypt F using a cryptographic symmetric function. Thus, the same file encrypted by different clients will result in equal ciphertexts. This allows to detect duplicates and apply an inter-user deduplication on encrypted files without any knowledge of their plaintext content. Clients may also store their data using plaintext or using a private encryption scheme. On the other side, the Cloud Gateway only uses a private encryption scheme in the additional encryption layer for outsourced data.

7.3 System architecture

We consider an environment consisting of tree types of components, namely the clients, the public clouds, and the cloud gateway (See Figure 7.1). These components are described in the following.

7.3.1 Clients

A client is a member or employee of the enterprise deploying the data outsourcing system. We assume that clients are authenticated when they use the storage service. In the remainder of the chapter a client identifier stands for a unique member or employee identity.

7.3.2 Public cloud storage providers

We envision an outsourcing method supporting any public cloud storage providers (*e.g.*, Dropbox[1], Amazon S3[2], Microsoft OneDrive[3], etc). In the remainder of the paper, only the `put` and `get` operations of public cloud storage providers are used to illustrate our solution.

7.3.3 Cloud Gateway

This is the key on which relies our proposition. The Cloud Gateway may be hosted by a server or a cluster of servers for more reliability. As we can find different applications using different storage access technologies, the Cloud Gateway aims to be a unified storage system, actually it is an object storage system supporting file and block storage interfaces (See Section 2.3). In the following, we consider a Cloud Gateway hosted on a single server in order to focus on the contribution of our method and keep its description simple. The Cloud Gateway implements the following software components:

The Storage Interface Adapter (SIA)

The Storage Interface Adapter gives to the Cloud Gateway the ability to provide a unified storage. It actually aims to implement various storage interfaces such as block, file, objects interfaces as well as public CSPs APIs. This component aims to be extensible so that new storage interfaces can be added to support a new public CSP Application Programming Interface (API) for example.

The Data Formatter (DF)

This component is responsible of data formatting operations. What we call data formatting operations includes data stripping/reconstruction, encoding/decoding, and encryption/decryption.

The Storage Proxy (SP)

All the `put` and `get` operations from clients are processed by this component. The Storage Proxy implements the Deduplication Proxy feature of the two-phase deduplication which allows to perform an inter-user deduplication while ensuring confidentiality and security against potential malicious users (See Section 6 for more information about the two-phase deduplication solution). Performing data deduplication on-premises reduces the amount of data to be outsourced. It also reduces the outsourcing data cost as it avoids to outsource duplicates and reduces data transfer from/to outsourcing locations.

The Storage Accelerator (SA)

The role of the Storage Accelerator is to speed up data access by implementing on-premises caching features on `put` and `get` operations. We assume that the Storage Accelerator is made of high performance components in order to accelerate the storage operations and provide fast data accesses. We assume that this component provides means to manage the data it contains (*e.g.* the ability to verify if a data item is stored in it using its unique identifier) and executes the cache placement/replacement strategy.

On `get` operations, the Storage Accelerator keeps the data read by clients for a chosen laps of time t defined by the enterprise in order to speed up their subsequent accesses. The cache placement or replacement strategy is defined by the enterprise according to its requirements. We describe `put` operations in Section 7.5.

[1] http://www.dropbox.com
[2] http://aws.amazon.com
[3] http://onedrive.com

On `put` operations, data are not meant to be kept durably on the Storage Accelerator. Data stored at the Storage Accelerator by the Storage Proxy are aimed to be outsourced to public CSPs or in the Private Data Store (See the next point for more information about this component). The data outsourcing operations can be performed synchronously on `put` operations from clients. In this case, as soon as a data item is received by the Cloud Gateway, this data item is outsourced and the client storing the data item is acknowledged to end the `put` operation. This data item is also stored at the Storage Accelerator to speed up its subsequent accesses. In order to optimize the Internet bandwidth usage in period of normal or high activity, another strategy is to outsource data asynchronously, periodically, in periods of low activity of the enterprise, in the nights or the weekends for example. In that case, upon a `put` operation, the data item is stored at the Storage Accelerator and the client is notified to end the `put` operation. The data item is later outsourced to public CSPs or in the Private Data Store (See the next point for more information about this component). The `put` operation is described in Section 7.4. Outsourcing data asynchronously may be suggested to data loss if some storage components of the Storage Accelerator are lost because of a crash for example before all the data have been outsourced. In order to enhance the reliability of the Storage Accelerator, the Storage Accelerator should implements some data reliability mechanisms to mitigate such data loss. We assume that the Storage Accelerator has data redundancy built-in.

The Private Data Store (PDS)

Depending on the data sensitivity or the enterprise security and privacy policy, some data, may not be supposed to be outsourced to public CSPs and should be kept on-premises. In that case, instead of being outsourced to Public CSPs, they will be sent to the Private Data Store. We assume that the Private Data Store is trusted and provides means to keep data durably.

The Data Outsourcing Controller (DOC)

This component implements the data outsourcing strategy of the enterprise. The Data Outsourcing Controller is parametrized with the different enterprise accounts in public CSPs and the different methods to store and retrieve data from them. Our solution aims to provide flexibility and full control to the enterprise on this point. The Data Outsourcing Controller relies on the Storage Interface Adapter to be compatible with the different APIs provided by the public CSPs, and on the Data Formatter to format data before the outsourcing operations. The Data Outsourcing Controller has also access to the cryptographic tools necessary to the outsourcing operations. Data may be stored using several strategies (mix of them may also be imagined). Data can be *(i)* stored on only one public CSP which is the simplest choice, *(ii)* fragmented and distributed to different clouds, or *(iii)* replicated over different CSPs using a simple copy of an erasure code which improves reliability as its tolerates CSPs unavailability depending on the redundancy degree. Data redundancy over several CSPs also allows to avoid CSP vendor lock-in but it consumes more storage space. For instance, using data redundancy by simple copy, 1GB of data requires 3GB of storage space in the cloud to tolerate the unavailability of two public CSPs. Note that regarding the data sensitivity or the enterprise security and privacy policy, some data may be stored in the Private Data Store instead of public CSPs.

The Metadata server (MDS)

It maintains all the information about data (*e.g.,* name, type, format, etc), the clients, the ownership relations between data and clients, and the data access history. We distinguish two types of metadata, 1) the *private metadata* that are the real information about data such as their name, owner, type, etc, and 2) the *public metadata* that are the metadata attributed to data when they are outsourced to public CSPs. A public metadata hides the real information about a data item

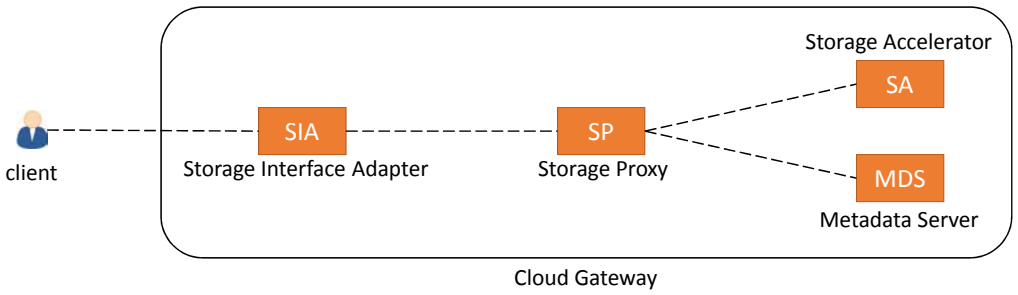


Figure 7.2: Data storage and caching on-premises

in order to strengthen confidentiality of outsourced data. Public metadata is essentially the name given to the outsourced data which is a cryptographic hash value of the encrypted outsourced data and its location.

7.4 The put operation

We describe in this section the way `put` operations are performed. Such operation compounds two steps, a data storage/caching on-premises at the Storage Accelerator and a data outsourcing process to public CSPs or to the Private Data Store.

7.4.1 Storage and caching on-premises

Figure 7.2 illustrates data storage and caching on-premises. Prior to store a file, users may encrypt it using a private encryption scheme (*i.e.* with a personal key) or a convergent encryption scheme (*i.e.* with a key derived from the file content). Data may also be stored in plaintext.

In the following, a user invoking a `put` operation on a file F will be called client and denoted by its identifier C_{id} and we consider that clients encrypt their files with a convergent encryption scheme and encryption keys are encrypted with an asymmetric encryption scheme and also stored by the Cloud Gateway. We assume that encryption keys are stored in the Private Data Store.

- C_{id} request is received by the Cloud Gateway in the Storage Interface Adapter components which translates it into the storage protocol used internally by the Cloud Gateway. The request is then forwarded to the Storage Proxy.
- The Storage Proxy applies the two-phase deduplication scheme on F.
- F is kept in the Storage Accelerator if it does not already exist.
- The Storage Proxy updates the MDS with the mapping about C_{id} and F_{id} meaning that this client also owns the file F. If F is not a duplicate, the Storage Proxy also updates F in the MDS with a tag meaning that F has not been outsourced yet. The outsourcing process is explained in the next point (See Section 7.4.2). The Storage Proxy also updates the operations history of F. It keeps a trace of the `put` operation of C_{id} on F.

7.4.2 Data outsourcing

Data can be outsourced synchronously as soon as they are received by the Cloud Gateway. An other strategy may be to outsource data periodically (by small data item groups) or only in low activity periods of the enterprise. In this work, we consider that outsourcing operations are performed asynchronously and in low activity periods of the enterprise. Figure 7.3 illustrates the data outsourcing process. The following steps are performed in an outsourcing operation.

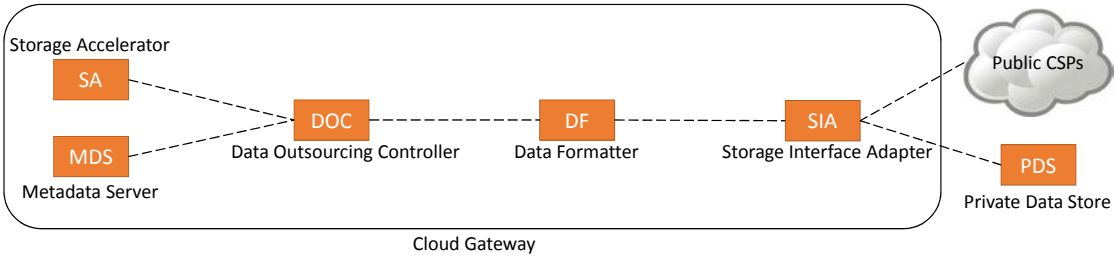


Figure 7.3: Data storage outsourcing

- **Get list of data to outsource:** The Data Outsourcing Controller gets from the MDS a list of a data items that should be outsourced. Note that if the outsourcing operation has to be performed synchronously (*e.g.* in the case where the Storage Accelerator is completely unavailable or removed), upon a `put` operation, the outsourcing operation will be triggered by the Storage Proxy that will pass the data item to be outsourced to the Data Outsourcing Controller.

- **Outsourcing location selection:** Depending on the sensitivity of a data item and the enterprise outsourcing policy, the Data Outsourcing Controller selects the Private Data Store, or one or several public CSPs.

- **Redundancy method selection:** Depending on the sensitivity of a data item, its outsourcing location, an algorithm defining how the data item will be stored is chosen among several strategies such as storing the data in the Private Data Store, in a single CSP, in multiple CSPs using simple replication, or using erasure coding over multiple CSPs, etc.

- **Data formatting:** This step is executed by the Data Formatter and it prepares the data to be outsourced. They may be encoded using a erasure coding technique for example. Data is also encrypted. Recall that this data item may have been already encrypted by its owner for his/his own confidentiality. Here, the additional encryption layer is added by the enterprise to improve the confidentiality and privacy against public CSPs. The Cloud Gateway will use secret enterprise keys to encrypt and decrypt outsourced data. We assume the secret enterprise keys safely kept in the Private Data Store. The Data Formatter also changes the encrypted data names before the outsourcing operation. A *public name* is given to outsourced data items which is the cryptographic hash value of the encrypted data item. This process aims to improve the privacy of the data against public CSPs by hiding their real name to public CSPs. The only information that are visible to CSPs are the public name of the data item and its size. It allows also to check the integrity of a data item by computing a cryptographic hash value of the data item received from public CSPs and comparing it with its public name.

- **File outsourcing:** When the location where to outsource a data item and the put algorithm are selected, and the encoding and encryption done, the data item is ready to be outsourced. The Data Outsourcing Controller uses the Storage Interface Adapter to send data using protocols/APIs supported by the selected outsource location.

- **MDS update:** When a data item has been successfully outsourced, the Data Outsourcing Controller updates the MDS with the means to retrieve this data item when needed. The necessary information to achieve it are the public name, the location of the data item or its fragments, and the method to retrieve/reconstruct the data item. The Data Outsourcing Controller also updates the data item record by removing the tag put in the phase where the client has stored it on-premises.

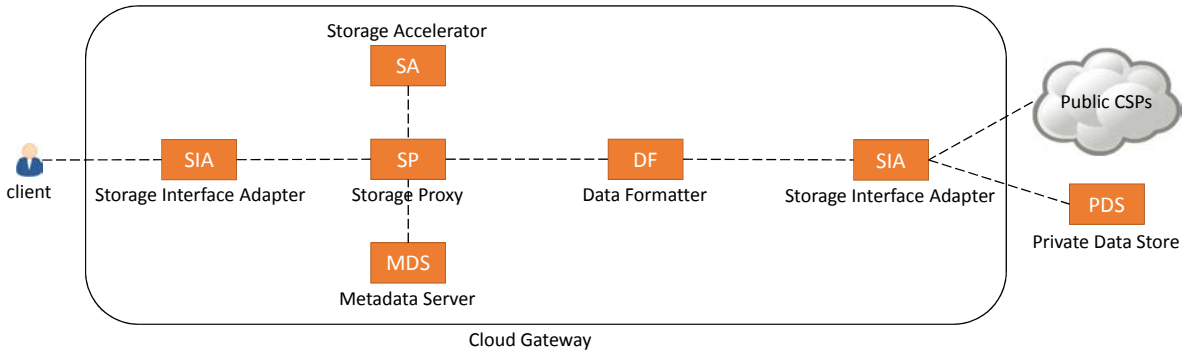


Figure 7.4: Get operation

7.5 The get operation

In this section we describe how `get` operations are performed in order to retrieve a file. The strategy to process `get` operations differs from whether the requested files are in the Storage Accelerator or not. In the following, a user invoking a `get` operation on a file F will be called client and denoted by its identifier C_{id}. Figure 7.4 illustrates a `get` operation.

7.5.1 A get operation on file located in Public CSPs

- C_{id} request is received by the Cloud Gateway in the Storage Interface Adapter components which translates it into the storage protocol used internally by the Cloud Gateway. The request is then forwarded to the Storage Proxy.
- The Storage Proxy retrieves from the MDS the necessary information to retrieve this file. These information are the public file name if the file is stored in public CSPs otherwise its private file identifier if the file is located in the Private Data Store. The Storage Proxy also retrieved the algorithm to retrieve and reconstruct the file, and its decryption keys (*i.e.* the enterprise decryption key and the encrypted client decryption key if existing).
- The Storage Proxy uses the Storage Interface Adapter component to translate the request into the protocol implemented by the Private Data Store or the public CSPs. The response is also translated in order to be processed by the Storage Proxy.
- When the Storage proxy receives the requested file from public CSPs, it decrypts it using the enterprise decryption key. Decryption process is not required when the file is retrieved from the Private Data Store. The Storage Proxy also gets from the MDS the local metadata of the file visible to that client (*e.g.*, the file name given by the client, the client decryption key if existing, the creation/modification date, etc) and sends it with the decrypted file to the client via the Storage Interface Adapter component to be compliant with the storage protocol used by the client. When the client receives them, he/she can decrypt the file if necessary.
- In parallel to sending the file to the client, the file is also stored in the Storage Accelerator to speed up its subsequent access and save Internet bandwidth.
- The Storage proxy updates the operation history of the file in the MDS.

7.5.2 A get operation on files cached in the Storage Accelerator

Before diving in the details of the `get` operation in the case where the requested file is located in the Storage Accelerator, we first describe the privacy risks of reading files from a cache server which is the role of the Storage Accelerator in this case.

Compared to a case where a file is retrieved from the public CSPs or the Private Data Store, retrieving the same file from the Storage Accelerator at the enterprise premises will reduce the access latency. Although reducing data access latency improves the clients quality of experience which is one the interest of implementing caching features, it leaks some information to clients. In fact, if the latency to access a data item is lower than expected, it reveals that the object was in the Storage Accelerator because an other client previously accessed to that data item.

In order to mitigate this information leakage, we designed a get algorithm that detect and prevent this possible information leakage upon a `get` operation. To achieve it, the Cloud Gateway maintains a data access history in the MDS. Each `put` and `get` operation of clients are recorded in this history. The record contains the identifier of the file accessed, the client identifier, the type and the time of the operation.

When a client performed a `get` operation, he/she sends his/her request to the Cloud Gateway.

- C_{id} request is received by the Cloud Gateway in the Storage Interface Adapter components which translates it into the storage protocol used internally by the Cloud Gateway. The request is then forwarded to the Storage Proxy.
- When the file is in the Storage Accelerator, the Storage proxy checks in the data access history of that file, if the client already accessed to that file in the past `cacheTime` which the duration of a data item in the cache if not accessed. If the client accessed to that file in that laps of time, the file is sent directly to him/her via the Storage Interface Adapter component. Otherwise, the Storage Proxy adds some delay before sending the data item.
- The Storage Proxy updates the operation history of the file in the MDS.

7.6 Related works

In the context of the adoption of the cloud storage system by enterprises, there have been several works in the literature dealing with the same issues we address in this chapter namely vendor security issues, high data access latency for data located in the cloud, and vendor locking [132, 31, 28, 16, 41, 15, 3, 64, 14]. The common approach, similar to ours, is to leverage multiple CSPs to avoid vendors locking and improve reliability against CSPs outages. Redundancy through replication or coding techniques are used to achieve it. Most of these works either use a cloud gateway/proxy approach or a multi-cloud approach. Recall that a multi-cloud approach refers to a service having the ability to leverage several cloud infrastructures. So in contrast to a multi-cloud approach, a cloud gateway approach have better potential to be easy integrated with different types of storage services with different storage access protocols while being transparent to clients. It allows to make evolve (*e.g.* adding new features, updates, etc.) the cloud gateway without modifying clients. Cloud Gateways seems to be more and more provided by commercial solutions that may support inter-connection of multiple cloud gateways to avoid Single Point of failure issues [132, 31, 28] . Compared to these solutions we consider the storage and bandwidth savings in storage operations (*i.e.* `put` operations) as well as in retrieval operations (*i.e.* `get` operations) thanks to the two-phase deduplication scheme and the on-premises caching feature.

7.7 Summary and perspectives

We presented in this Chapter a data outsourcing method architecture allowing enterprises to extend their on-premises storage capacity using public CSPs while keeping control on their data via a private on-premises metadata management system, improving confidentiality and privacy against public CSPs via the encryption of outsourced data. A secure two phase data deduplication also ensures security against potential malicious clients.

By caching data that are frequently accessed on-premises, this solution speeds up data access and reduces bandwidth cost to retrieve data from public CSPs. Performance to store data is also improved as data can be outsourced asynchronously. They are first stored in the Storage Accelerator and outsourced later in public CSPs or in Private Data Store. This solution also reduces the storage cost of data in public CSPs thanks to the deduplication performed on-premises, avoiding to store duplicates in the cloud. The deduplication performed being a two phase deduplication, it protects the system against potential malicious clients.

The Cloud Gateway on which relies our solution, via a Storage Interface Adapter, aims to implement various storage interfaces allowing to ease its integration with existing, or legacy, applications that use file, block, or object storage protocols. It also aims to be flexible enough so that data placement can be controlled.

We believe that the data placement scheme presented in Chapter 5 may be suitable to manage the data placement over the Private Data Store or public CSPs since we can find similitude in the architecture of a region in Mistore and in the enterprise Cloud Gateway proposed in this Chapter. In fact the Cloud Gateway is located on the path between the clients and the data outsourcing location. Integration of this data placement can be a direction for future work.

Future works can be to enhance our solution so that it supports multiple Cloud Gateways located in different enterprise sites and an everywhere data access capability. In fact more and more enterprises have several sites and their activities may require data access from outside the enterprise sites.

Another direction for future work may be to enhance the additional encryption operations so that data encryption can be changed without consuming lot a resources *i.e.* avoid to retrieve all the data from public CSPs, decrypt them and re-encrypt them with a new encryption key. The same issue is found in the case of a migration data from one CSPs to another.

8

Summary and perspectives

In this PhD thesis, we studied several topics such as Cloud, storage, or dependability. In the following we summarize and review our contributions and give some perspectives.

Contributions

- **Mistore:** In Chapter 4, we described Mistore, a distributed storage system that we designed to ensure data availability, durability, low access latency by leveraging the Digital Subscriber Line (xDSL) infrastructure of an ISP. Mistore uses the available storage resources of a large number of home gateways and Points of Presence for content storage and caching facilities. Mistore also targets data consistency by providing multiple types of consistency criteria on content and a versioning system.

- **Data placement and maintenance in Mistore:** This contribution, presented in Chapter 5 aims to improve the data placement and maintenance of our previous contribution, Mistore. In this work, we designed a scalable and flexible data placement scheme in which the amount of metadata to maintain depends only on the number of nodes in the system and not on the potentially enormous amount of data to manage.

- **The two phase data deduplication:** In this contribution, we designed a method allowing to build a storage system performing a client-side data deduplication that is secure against attacks from malicious clients. Our solution remains effective in terms of storage space and network bandwidth consumption and have been patented. We described the solution in Chapter 6.

- **An enterprise data outsourcing method:** In this contribution, we designed a solution allowing enterprises to outsource data to the cloud while keeping control on them. Our solution leverages a cloud gateway located at the enterprise premises that aims to provide control on data stored in cloud, improve data access performance, optimize storage and network bandwidth usage, and provide data confidentiality. To achieve that, this cloud gateway implements a metadata management on-premises as well as features such as caching, flexible data placement over multiple cloud storage services, data encryption. Our method also used the secure two-phase data deduplication to improve storage and network bandwidth consumption. We described this solution Chapter 7.

Perspectives

- **Data consistency flexibility:** Not all the applications requires the same level of data consistency. In our work about the Mistore architecture (See Chapter 4, we provided the flexibility to select a desired level of data consistency upon read operations. However, we adopted a single-writer multiple-readers model. In consequence, Mistore fit better for applications where concurrent write operations are not so frequent. In the future, mechanism to support multiple-writers on the same data will be interesting to be added in Mistore while providing the flexibility of choosing the required data consistency upon operations.

- **Multi- and auto-tiered storage:** Not all data are equal in terms of access frequency, access performance or retention duration requirements(*e.g.* hot, warm, and cold data). On the other side, nodes in an architecture are not necessarily equal in terms of storage capacity, performance, their proximity to the users and the access latency they may exhibit (*e.g.* Home Gateways, Points of Presence, and Datacenter in Mistore) so it can make sense to think about solutions that will leverage the data and nodes characteristics to optimize data management. In this context, we believe that it will be interesting to investigate in multi- and auto-tiered storage that refers to an automated placement of data on nodes to optimize performance, availability, recovery, and cost.

- **Distributed cloud gateway system:** In Chapter 7 we designed a solution based on an on-premises cloud gateway allowing enterprises to outsource data to the cloud while keeping control on them. However enterprises may have several sites geographically dispersed and having only one cloud gateway to serve clients all around the different enterprises site is not the best choice,especially in term of data access latency for clients that are distant form the cloud gateway. It will be interesting to investigate means to build a distributed cloud gateway system while maintaining the different advantages of the initial solution.
- **Encryption keys rotation and changes of CPSs:** While encryption brings confidentiality, it is also important to change or rotate the encryption keys to improve the security. Rotation key is good to consider for instance when some people who has access to the encryption keys leaves the organization or when the encryption keys are or supposed to be compromised. It will interesting to introduce additional encryption operations so that encryption keys can be changed without consuming lot a resources *i.e.* avoid to retrieve all the data from public CSPs, decrypt them, re-encrypt them with a new encryption key, and upload them. Due to the fact that a versionning system may be present, encryption key rotation may also be associated with some solutions to limit the access to data encrypted under previous encryption keys. Similarly it will be interesting to investigate flexible means to mobility or migration of data between several providers in a cost-effective way in term of resources consumption, performance, availability, and security.

To conclude, in this PhD thesis we had the opportunity to discover and study a large panel of topics regarding cloud, storage, architecture, dependability technologies. We believe that more challenges still have to be addressed or investigated. And with the growing usage and adoption of cloud solutions, proliferation of connected devices, more challenges will appear.

Bibliography

[1] The 10 biggest cloud outages of 2015 (so far). http://www.crn.com/slide-shows/cloud/300077635/the-10-biggest-cloud-outages-of-2015-so-far.htm/pgno/0/2. Accessed: 2015-11-26. 88

[2] Daniel J. Abadi. Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *Computer*, 45:37–42, 2012. 26, 33, 45

[3] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. Racs: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 229–240. ACM, 2010. 95

[4] Tolga Acar, Mira Belenkiy, Lan Nguyen, and Carl Ellison. Key management in distributed systems. Technical Report MSR-TR-2010-78, Microsoft Research, June 2010. Previously submitted to NDSS 2010. 28

[5] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th ACM Symposium on Operating Systems Design and Implementation (OSDI)*, 2002. 62, 63

[6] Marcos K. Aguilera, Kimberly Keeton, Arif Merchant, Kiran-Kumar Muniswamy-Reddy, and Mustafa Uysal. Improving recoverability in multi-tier storage systems. In *Procs of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007. 21, 45, 81

[7] Peter A. Alsberg and John D. Day. A principle for resilient sharing of distributed resources. In *Procs of the 2nd international conference on Software engineering (ICSE)*, 1976. 37

[8] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, December 2004. 23

[9] Siddhartha Annapureddy, Michael J. Freedman, and David Mazières. Shark: scaling file servers via cooperative caching. In *Proceedings of the 2nd USENIX Conference on Symposium on Networked Systems Design & Implementation (NSDI)*, 2005. 63

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010. 2

[11] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan 2004. 18

[12] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *Proceedings of the 22nd USENIX Conference on Security (SEC)*, 2013. 65, 67, 70, 83, 84, 89

[13] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013. 67, 70, 83, 89

[14] David Bermbach, Markus Klems, Stefan Tai, and Michael Menzel. Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 452–459. IEEE, 2011. 95

[15] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12, 2013. 95

[16] Alysson Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo. Scfs: a shared cloud-backed file system. In *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, pages 169–180. USENIX Association, 2014. 95

[17] Charles Blake and Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: pick two. In *Procs of the 9th conference on Hot Topics in Operating Systems (HOTOS)*, 2003. 19, 32, 44

[18] William J. Bolosky, Scott Corbin, David Goebel, and John R. Douceur. Single instance storage in Windows 2000. In *Proceedings of the 4th Conference on USENIX Windows Systems Symposium (WSS)*, 2000. 62, 63

[19] Peter J Braam et al. The lustre storage architecture, 2004. 22, 24

[20] Eric Brewer. Cap twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29, 2012. 26

[21] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005. 65, 88

[22] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 1–14. ACM, 2007. 9

[23] Scott Chacon. *Pro Git*. Apress, Berkely, CA, USA, 1st edition, 2009. 82

[24] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996. 19, 34

[25] Ashish Chawla, Benjamin Reed, Karl Juhnke, and Ghousuddin Syed. Semantics of caching with spoca: a stateless, proportional, optimally-consistent addressing algorithm. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIX ATC'11. USENIX Association, 2011. 22, 23, 48, 52, 59

[26] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Procs of the International Conference on Networked Systems Design & Implementation (NSDI)*, 2006. 18, 57, 81

[27] Kenneth Church, Albert Greenberg, and James Hamilton. On Delivering Embarrassingly Distributed Cloud Services. In *HotNets*, 2008. 44

[28] Panzura CloudFS. http://panzura.com/. 95

[29] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans A. Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *Procs of the VLDB Journal*, 1(2):1277–1288, 2008. 27, 45

[30] Peter F Corbett and Dror G Feitelson. The vesta parallel file system. *ACM Transactions on Computer Systems (TOCS)*, 14(3):225–264, 1996. 22, 24

[31] Ctera. http://www.ctera.com/. 95

[32] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a dht for low latency and high throughput. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation*, NSDI'04. USENIX Association, 2004. 19, 20

[33] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. 27

[34] Chris J Date and Hugh Darwen. *A guide to the SQL Standard: a user's guide to the standard relational language SQL*, volume 55822. Addison-Wesley Longman, 1993. 8

[35] Terry Daugherty, Matthew S. Eastin, and Laura Bright. Exploring consumer motivations for creating user-generated content. *Journal of Interactive Advertising*, 8(2):16–25, 2008. 9

[36] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Procs of the ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2007. 26, 27, 32, 45

[37] Serge Defrance, Anne-Marie Kermarrec, Erwan Le Merrer, Nicolas Le Scouarnec, Gilles Straub, and Alexandre Van Kempen. Efficient peer-to-peer backup services through buffering at the edge. In *Procs of the International Conference on Peer-to-Peer Computing (P2P)*, 2011. 32

[38] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *Proceedings of the The International Conference on Smart Card Research and Applications*, CARDIS '98, pages 167–182. Springer-Verlag, 2000. 65, 88

[39] Roberto Di Pietro and Alessandro Sorniotti. Boosting efficiency and security in proof of ownership for deduplication. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2012. 67, 83, 84

[40] J. Dilley, B. Maggs, J. Parikh, Harald Prokop, Ramesh Sitaraman, and B. Weihl. Globally distributed content delivery. *Internet Computing, IEEE*, 6(5):50–58, Sep 2002. 44

[41] Dan Dobre, Paolo Viotti, and Marko Vukolić. Hybris: Robust hybrid cloud storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014. 95

[42] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2002. 70, 89

[43] Driverdan. Dropship-dropbox api utilities. https://github.com/driverdan/dropship. 64

[44] Dropbox. http://www.dropbox.com. 62, 64

[45] B. Dufrasne, P. Kimmel, B. Wilson, and IBM Redbooks. *DS8000 Thin Provisioning*. IBM Redbooks, 2014. 81

[46] Alessandro Duminuco, Ernst Biersack, and Taoufik En-Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *Proceedings of the 2007 ACM CoNEXT Conference*, CoNEXT '07, pages 27:1–27:12. ACM, 2007. 20

[47] Mike Dutch. Understanding data deduplication ratios. *SNIA Data Management Forum*, 2008. 62

[48] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Oltean, Jin Li, and Sudipta Sengupta. Primary data deduplication-large scale study and system design. In *Proceedings of the USENIX conference on Annual Technical Conference (ATC)*, 2012. 63

[49] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*, pages 25–32. ACM, 2000. 65, 88

[50] Pierre Francois, Clarence Filsfils, John Evans, and Olivier Bonaventure. Achieving sub-second igp convergence in large ip networks. *ACM SIGCOMM Computers Communication Review*, 35(3):35–44, 2005. 34

[51] John Gantz and David Reinsel. The digital universe decade. are you ready? *IDC iView: IDC Analyze the future*, 2010. ii, vi, 61

[52] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2012. i, v

[53] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43. ACM, 2003. 18, 19

[54] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002. 25

[55] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361, August 2011. 18

[56] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Procs of the ACM SIGCOMM Conference*, 2011. 38

[57] R. Gracia-Tinedo, M. Sanchez-Artigas, A. Moreno-Martinez, and P. Garcia-Lopez. FriendBox: A hybrid F2F personal storage application. In *Procs of the IEEE International Conference onCloud Computing (CLOUD)*, 2012. 44

[58] C. Gray and D. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *Procs of the ACM Symposium on Operating Systems Principles (SOSP)*, 1989. 37

[59] Haryadi S. Gunawi, Nitin Agrawal, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Jiri Schindler. Deconstructing commodity storage clusters. *ACM SIGARCH Computer Architecture News*, 33, 2005. 62, 63

[60] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications security (CCS)*, 2011. 62, 64, 67, 83, 84

[61] Joseph Hall, Jason Hartline, Anna R Karlin, Jared Saia, and John Wilkes. On algorithms for efficient data migration. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 620–629. Society for Industrial and Applied Mathematics, 2001. 23

[62] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security Privacy*, 8, 2010. 62, 64, 65, 67, 82, 84

[63] O. Heen, C. Neumann, L. Montalvo, and S. Defrance. Improving the resistance to side-channel attacks on cloud storage services. In *Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS)*, 2012. 67, 82, 84

[64] Yuchong Hu, Henry CH Chen, Patrick PC Lee, and Yang Tang. Nccloud: applying network coding for the storage repair in a cloud-of-clouds. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, pages 21–21. USENIX Association, 2012. 95

[65] Cheng Huang, Angela Wang, Jin Li, and Keith W. Ross. Understanding hybrid CDN-P2P: why limelight needs its own Red Swoosh. In *Procs of the International ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2008. 33, 80

[66] James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy. Delta algorithms: An empirical analysis. *ACM Trans. Softw. Eng. Methodol.*, 7(2):192–214, April 1998. 80, 82

[67] Ken ichiro Ishikawa. Asura: Scalable and uniform data distribution algorithm for storage clusters. *CoRR*, abs/1309.7720, 2013. 22, 23, 24, 48, 52, 59

[68] Keren Jin and Ethan L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of ACM The Israeli Experimental Systems Conference (SYSTOR)*, 2009. 62

[69] J. Jonsson and B. Kaliski. Public-key cryptography standards (pkcs) #1: Rsa cryptography specifications version 2.1, 2003. 28

[70] Burt Kaliski. A survey of encryption standards. *IEEE Micro*, 13(6):74–81, November 1993. 27

[71] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security (FC)*, 2010. 5, 27, 88

[72] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663. ACM, 1997. 22, 24, 48, 52, 58

[73] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 104–113. Springer-Verlag, 1996. 65, 88

[74] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Operating Systems Review*, 44(2):35–40, 2010. 26, 27, 45

[75] Leslie Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, 1977. 36

[76] YoungSeok Lee, YounSoo Kim, and Hoon Choi. Conflict resolution of data synchronization in mobile environment. In *Computational Science and Its Applications âĂŞ ICCSA 2004*, volume 3044 of *Lecture Notes in Computer Science*, pages 196–205. Springer Berlin Heidelberg, 2004. 9

[77] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. *ACM Comput. Surv.*, 19(3):261–296, September 1987. 80, 82

[78] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems (extended abstract). In *Proceedings of the 8th International Financial Cryptography conference (FC)*, pages 251–265. Springer, 2004. 65, 88

[79] Wubin Li, P. Svard, J. Tordsson, and E. Elmroth. A general approach to service deployment in cloud environments. In *Second International Conference on Cloud and Green Computing (CGC)*, pages 17–24, Nov 2012. 4

[80] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezise, and Peter Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, 2009. 63, 64

[81] Lin, W. K. and Chiu, D. M. and Lee, Y. B. Erasure Code Replication Revisited. In *Procs of the International Conference on Peer-to-Peer Computing (P2P)*, 2004. 49

[82] Chuanyi Liu, Xiaojian Liu, and Lei Wan. Policy-based de-duplication in secure cloud storage. In *Trustworthy Computing and Services*, volume 320 of *Communications in Computer and Information Science*, pages 250–262. Springer Berlin Heidelberg, 2013. 67, 70, 83, 84, 89

[83] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005. 32

[84] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ritu Sharma, and Sharon Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, 7(2):72–93. 23

[85] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011. 2

[86] Memopal. http://www.memopal.com. 62

[87] Mike Mesnier, Gregory R Ganger, and Erik Riedel. Object-based storage. *Communications Magazine, IEEE*, 41(8):84–90, 2003. xv, 9, 13

[88] Pierre Meye, Philippe Raïpin, Frédéric Tronel, and Emmanuelle Anceaume. Toward a distributed storage system leveraging the DSL infrastructure of an ISP. In *Proceedings of the 11th IEEE Consumer Communications and Networking Conference (CCNC)*, 2014. 80

[89] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST)*, 2011. 62, 63

[90] A. Miranda, S. Effert, Y. Kang, E.L. Miller, A. Brinkmann, and T. Cortes. Reliable and randomized data distribution strategies for large scale storage systems. In *High Performance Computing (HiPC), 2011 18th International Conference on*, pages 1–10, 2011. 22, 23, 24, 48, 52, 53, 59

[91] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for http. *SIGCOMM Comput. Commun. Rev.*, 27(4):181–194, October 1997. 82

[92] Abedelaziz Mohaisen, Xinwen Zhang, Max Schuchard, Haiyong Xie, and Yongdae Kim. Protecting access privacy of cached contents in information centric networks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 173–178. ACM, 2013. 65, 88

[93] Mozy. http://mozy.com. 62

[94] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In *Proceedings of the 20th USENIX Conference on Security (SEC)*, 2011. 62, 64, 67, 83

[95] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. *ACM SIGOPS Operating Systems Review*, 35, 2001. 63

[96] Wee Keong Ng, Yonggang Wen, and Huafei Zhu. Private data deduplication protocols in cloud storage. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC)*, 2012. 67, 83, 84

[97] NISO. Understanding metadata. In *NISO Press*, 2004. 8

[98] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, July 2010. 44

[99] George Pallis and Athena Vakali. Insight and perspectives for content delivery networks. *Communications of the ACM*, 49(1), 2006. 33, 44, 80

[100] Andrea Passarella. A survey on content-centric technologies for the current Internet: CDN and P2P solutions. *Computers Communications*, 35(1):1–32, 2012. 33

[101] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 109–116. ACM, 1988. 58

[102] Stacy Patterson, Aaron J. Elmore, Faisal Nawab, Divyakant Agrawal, and Amr El Abbadi. Serializability, not serial: concurrency control and availability in multi-datacenter datastores. *Procs of the VLDB Endowment Journal*, 5(11):1459–1470, 2012. 27, 45

[103] Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004. 82

[104] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 395–403. IEEE, 2009. 27

[105] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *Proceedings of USENIX the Conference on File and Storage Technologies (FAST)*, 2002. 63

[106] Michel Raynal and AndrÃĺ Schiper. A suite of formal definitions for consistency criteria in distributed shared memories. In *Procs of the International Conference on Parallel and Distributed Computing Systems (PDCS)*, 1996. 34

[107] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *IEEE Internet Computing*, 16, 2012. 5, 27, 88

[108] Sean C. Rhea, Kevin Liang, and Eric Brewer. Value-based web caching. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 619–628. ACM, 2003. 82

[109] Francisco Rocha, Salvador Abreu, and Miguel Correia. The final frontier: Confidentiality and privacy in the cloud. *Computer*, 44(9):44–50, 2011. 27

[110] Rodrigo Rodrigues and Barbara Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *Procs of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005. 19, 49

[111] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The Internet Society (ISO)*, 2015. i, v

[112] Dave Russel. Data deduplication will be even bigger in 2010. *Gartner*, February 2010. 62

[113] David Salomon and Giovanni Motta. *Handbook of Data Compression (5. ed.)*. Springer, 2010. 82

[114] Werner Schindler. A timing attack against rsa with the chinese remainder theorem. In *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, pages 109–124. Springer-Verlag, 2000. 65, 88

[115] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07. USENIX Association, 2007. 18

[116] Beomjoo Seo. Survey on data placement and migration algorithms in distributed disk systems. In *Proceedings of 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2004. 23

[117] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10, May 2010. 19

[118] Gustavus J. Simmons. Symmetric and asymmetric encryption. *ACM Comput. Surv.*, 11(4):305–330, December 1979. 27

[119] Emil Sit, Andreas Haeberlen, Frank Dabek, Byung gon Chun, Hakim Weatherspoon, Robert Morris, M. Frans Kaashoek, and John Kubiatowicz. Proactive replication for data durability. In *In Proceedings of the 5th IntâĂŹl Workshop on Peer-to-Peer Systems (IPTPS*, 2006. 20

[120] Vision Solutions. Assessing the financial impact of downtime. *White Paper*. 18

[121] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001. 23, 24, 59

[122] Mark W. Storer, Kevin Greenan, Darrell D.E. Long, and Ethan L. Miller. Secure data deduplication. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability (StorageSS)*, 2008. 62, 64, 67, 82, 84

[123] Ye Sun, Fangming Liu, Bo Li, Baochun Li, and Xinyan Zhang. FS2You: Peer-assisted semi-persistent online storage at a large scale. In *Procs of the International IEEE INFOCOM Conference*, 2009. 44

[124] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. Memory deduplication as a threat to the guest os. In *Proceedings of the 4th ACM European Workshop on System Security (EUROSEC)*, 2011. 65, 88

[125] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the xfs file system. In *USENIX Annual Technical Conference*, volume 15, 1996. 22

[126] Dan Teodosiu, Nikolaj Bjorner, Joe Porkka, Mark Manasse, and Y. Gurevich. Optimizing file replication over limited-bandwidth networks using remote differential compression. Technical report, Microsoft Research, November 2006. 82

[127] Doug Terry. Replicated data consistency explained through baseball. *Commun. ACM*, 56(12):82–89, December 2013. 26

[128] Walter F. Tichy. Rcs – a system for version control. *Softw. Pract. Exper.*, 15(7):637–654, July 1985. 82

[129] L. Toka, M. Dell'Amico, and P. Michiardi. Online data backup: A peer-assisted approach. In *Procs of the Conference on Peer-to-Peer Computing (P2P)*, 2010. 32, 44

[130] Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv.*, 47(1):7, 2014. 4

[131] Andrew Tridgell and Paul Mackerras. The rsync algorithm. 1996. 82

[132] Nasuni UniFS. http://www.mongodb.org/. 95

[133] Vytautas Valancius, Nikolaos Laoutaris, Laurent Massoulié, Christophe Diot, and Pablo Rodriguez. Greening the internet with nano data centers. In *Procs of the International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009. 32, 44

[134] Werner Vogels. Eventually consistent. *Commununications of the ACM*, 52(1):40–44, 2009. 26, 33, 45

[135] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Procs of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002. 19, 49

[136] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: a scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 307–320. USENIX Association, 2006. 23, 24, 48, 52, 58, 59

[137] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. Crush: controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06. ACM, 2006. 22, 24, 48, 52, 58, 59

[138] Wuala. http://www.wuala.com. 32, 44

[139] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. Security implications of memory deduplication in a virtualized environment. In *Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, DSN '13, pages 1–12. IEEE Computer Society, 2013. 65, 88

[140] Jia Xu, Ee-Chien Chang, and Jianying Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIACCS)*, 2013. 67, 70, 83, 84, 89

[141] Lawrence L. You, Kristal T. Pollack, and Darrell D. E. Long. Deep store: An archival storage system architecture. In *Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE)*, 2005. 62, 63

[142] Junyao Zhang, Pengju Shang, and Jun Wang. A scalable reverse lookup scheme using group-based shifted declustering layout. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 604–615. IEEE, 2011. 22, 23, 48

[143] Qingji Zheng and Shouhuai Xu. Secure and efficient proof of storage with deduplication. In *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012. 67, 83, 84

[144] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, 2008. 63

[145] Huijun Zhu, Peng Gu, and Jun Wang. Shifted declustering: a placement-ideal layout scheme for multi-way replication storage architecture. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 134–144. ACM, 2008. 23

Résumé

La quantité de données stockées dans le monde ne cesse de croître posant des challenges aux fournisseurs de services de stockage qui doivent trouver des moyens de faire face à cette croissance de manière scalable, efficace, tout en optimisant les coûts. Nous nous sommes intéressés aux systèmes de stockage de données dans le nuage qui est une grande tendance dans les solutions de stockage de données. L'International Data Corporation (IDC) prédit notamment que d'ici 2020, environ 40% des données seront stockées et traitées dans le nuage. Cette thèse adresse les challenges liés aux performances d'accès aux données et à la sûreté de fonctionnement dans les systèmes de stockage dans le nuage. Nous avons proposé Mistore, un système de stockage distribué que nous avons conçu pour assurer la disponibilité des données, leur durabilité, ainsi que de faibles latences d'accès aux données en exploitant des zones de stockage dans les box, les Points de Présence (POP), et les centres de données dans une infrastructure Digital Subscriber Line (xDSL) d'un Fournisseur d'Accès à Internet (FAI). Dans Mistore, nous adressons aussi les problèmes de cohérence de données en fournissant plusieurs critères de cohérence de données ainsi qu'un système de versioning. Nous nous sommes aussi intéressés à la sécurité des données dans le contexte de systèmes de stockage appliquant une déduplication des données, qui est l'une des technologies les plus prometteuses pour réduire les coûts de stockage et de bande passante réseau. Nous avons conçu une méthode de déduplication sécurisée contre des attaques d'utilisateurs malicieux tout en étant efficace en termes d'économie de bande passante réseau et d'espace de stockage.

Mots-clés: Sûreté de fonctionnement; nuage de stockage; cloud storage; systèmes distribués; cohérence des données, placement des données; confidentialité des données; déduplication;

Abstract

The quantity of data in the world is steadily increasing bringing challenges to storage system providers to find ways to handle data efficiently in terms of dependability and in a cost-effectively manner. We have been interested in cloud storage which is a growing trend in data storage solution. For instance, the International Data Corporation (IDC) predicts that by 2020, nearly 40% of the data in the world will be stored or processed in a cloud. This thesis addressed challenges around data access latency and dependability in cloud storage. We proposed Mistore, a distributed storage system that we designed to ensure data availability, durability, low access latency by leveraging the Digital Subscriber Line (xDSL) infrastructure of an Internet Service Provider (ISP). Mistore uses the available storage resources of a large number of home gateways, Points of Presence, and datacenters for content storage and caching facilities. Mistore also targets data consistency by providing multiple types of data consistency criteria and a versioning system. We also considered the data security and confidentiality in the context of storage systems applying data deduplication which is becoming one of the most popular data technologies to reduce the storage cost and we design a data deduplication method that is secure against malicious clients while remaining efficient in terms of network bandwidth and storage space savings.

Mots-clés: Dependability; cloud storage; distributed systems; data consistency; data placement; data confidentiality; deduplication;