Leveraging content properties to optimize distributed storage systems

Konstantinos Kloudas

N° d'ordre : 00000 ANNÉE 2013

UNIVERSITÉ DE RENNES 1

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

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Konstantinos KLOUDAS

préparée à l'unité de recherche INRIA–Bretagne Atlantique
Institut National de Recherche en Informatique et Automatique
Université de Rennes 1

Leveraging Content Properties to Optimize Distributed Storage Systems.

Thèse soutenue à Rennes le 6 Mars 2013

devant le jury composé de :

Gaël THOMAS
Associate Professor at UPMC / *Rapporteur*
Pascal FELBER
Professor Univ. de Neuchâtel / *Rapporteur*
Paolo COSTA
Researcher Imperial College London / *Examinateur*
Davide FREY
Researcher INRIA / *Examinateur*
François TAÏANI
Professor Univ. Rennes 1 / *Examinateur*
Anne-Marie KERMARREC
Directrice de recherche INRIA / *Directrice de thèse*

Contents

Chapter 1

Introduction

Cloud service providers, social networks and data-management companies are witnessing a tremendous increase in the amount of data they receive every day. Recent studies [VOE11, GCM+08a] report that the size of data produced and stored has been increasing exponentially the last 5 years, as shown in Figure 1.1, and projecting this growth rate to the year 2020, they expect the size of our "digital world" to reach 35 ZBs, or 35 trillion GBs, by that time. For comparison, back in 2009, our whole digital ecosystem contained only 0.8 ZBs of data, which roughly corresponds to $1/44th$ of the volume expected in 2020, as shown in Figure 1.2.

The main reasons for this fast increase in data production can be traced back to (i) the expansion of the geographical limits of the Internet, as now users with a smartphone can connect to it from almost anywhere, and (ii) to two major changes in the way users interact with it. Focusing on these two changes, the first has to do with the fact that the user nowadays is placed in the centre of the content creation process. In the early years of the Internet, content production and publication was the privilege of a small number of individuals or companies. In fact, in the classic paper describing PageRank [BP98], S. Brin and L. Page, report that in 1994, one of the first web search engines, the World Wide Web Worm (WWWW) had an index of only $110,000$ web pages and web accessible documents. This was due to (i) the costs involved in accessing the internet and hosting one's content, and (ii) the technical knowledge required for doing so. Nowadays, both of the above barriers have been removed. The cost of having a good internet connection is no longer an issue in most of the developed countries. In addition, services like Facebook, YouTube, Flickr, Twitter and many others allow the user to publish anything, at any time for free and with almost no technical prerequisites. The impact of this revolution can only be compared to the impact that the invention of typography had on the publishing process back in the mid-15th century. The second change that had a severe impact on

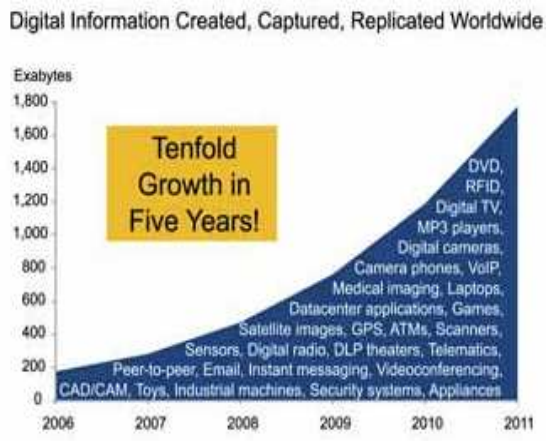


Figure 1.1: Data Growth Rate

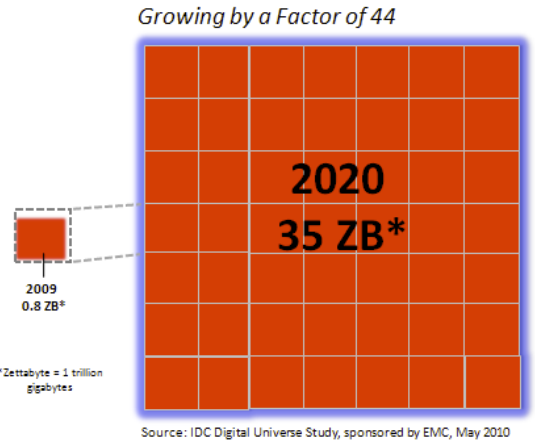


Figure 1.2: Data Size in 2020

the increase of produced data, has to do with the dramatic increase in devices located at the periphery of the network that produce data and have access to the internet. These include embedded sensors, smart-phones, and tablet computers.

All this data creates new opportunities to expand human knowledge in fields like human genetics, healthcare, city planning and human behavior and improve offered services like search, recommendation, and many others. It is not by accident that many academics but also public media refer to our era as the "Big Data" era. But these huge opportunities allowed by this unprecedented abundance of data come with the requirement for better data management systems that, on one hand, can safely accommodate this huge and constantly increasing volume of data and, on the other, serve them in a timely and useful manner so that applications can benefit from processing them. Failing to achieve either one of the above goals can drastically limit the potentials offered by "Big Data". Failing to protect data from dangers like hardware failures and natural disasters can lead to their loss or corruption. In addition, failing to keep up with the pace at which their volume increases, will inevitably lead to the deletion of data due to lack of storage space, which, in turn, may lead to "loss of memory" with important consequences [eco]. To illustrate the later one, a simple search for events like national elections in countries even as recent as five years ago, would reveal that many of the links that pointed to articles on the event are now dead and the corresponding article has been removed. The above phenomenon may lead to loss of human memory or, even worse, controlled writing of history, as only the information that "survived" will be available to the historians of the future. Finally, not being able to serve content in a timely and useful way, can (i) discourage users from using services, thus stopping providing data leading to service deterioration for services like recommendation and (ii) lead to severe system malfunctions, in case of time-critical

applications. To this end, a lot of effort from both the academia and the industry is being invested in "taming" this "Big Data" and the challenges that arise in both storing this content, as well as serving it.

This document focuses on the above two challenges that come with "Big Data". In our study, we focus on (i) backup storage as a means to safeguard data against a number of factors that may render them unavailable and (ii) on data placement on geographically distributed storage systems, with the goal that the latencies perceived from the end-user are minimized and the network and storage resources are efficiently utilized. As we will see in the following paragraphs, the requirement for safe storage, combined with the volume of "Big Data", makes the design of storage and backup systems able to accommodate them, a constantly challenging research problem. In addition, the never satisfied demand of users for better quality of the offered services, makes serving content an everyday engineering feat. Throughout our study, data are placed in the centre of our design choices as we try to leverage content properties for both placement and efficient storage.

1.1 Archival Storage and Data Deduplication

In a perfect world we would never have to worry about computers failing, nor would we have to worry about natural disasters. Unfortunately, we do not live in a perfect world, and when it comes to magnetic disks, which is the main means used to store all new information produced [LV03], hardware failures seem to be more frequent than one can imagine. To this end, and to protect data against the above dangers, individuals as well as organizations rely on frequent backups for safeguarding their data.

To illustrate the importance of backups, in this paragraph we provide statistics about "disk mortality" in order to show that disk failures are not as uncommon as one may think. In 2007, researchers from Google conducted a study on a large-scale deployment of PATA and SATA drives [PWB07]. These are the types of drives that home users buy for personal use and, apparently, the ones Google uses in its data centers. One of the main findings of this study was that magnetic disks fail more often than expected. According to their study, there is a non-negligible phenomenon of "infant mortality", where disks fail in the first three months of their lifetime and after that around 8% of the disks fail after 2 years of use and this percentage increases for 3 year old drives. Figure 1.3 presents the results from [PWB07]. The same study shows that apart from the very young and the very old drives, the level of utilization does not affect significantly the probability of a disk to break down. This implies that both home users and bigger organizations, that are expected to stress more their infrastructure, face an equal

danger of loosing their data. In another study of the same year [SG07], the authors do not limit their sample to PATA and SATA but they also include SCSI and FC interfaces that have much higher MTTFs, i.e. Mean-Time-To-Failure. Again, one of the main findings is that in the field, annual disk replacement rates typically exceed 1%. In fact, $2 - 4\%$ is a common case and this percentage can reach up to 13%, as the authors of [SG07] observed on some systems. In this context, regular backups are required to guarantee data recoverability.

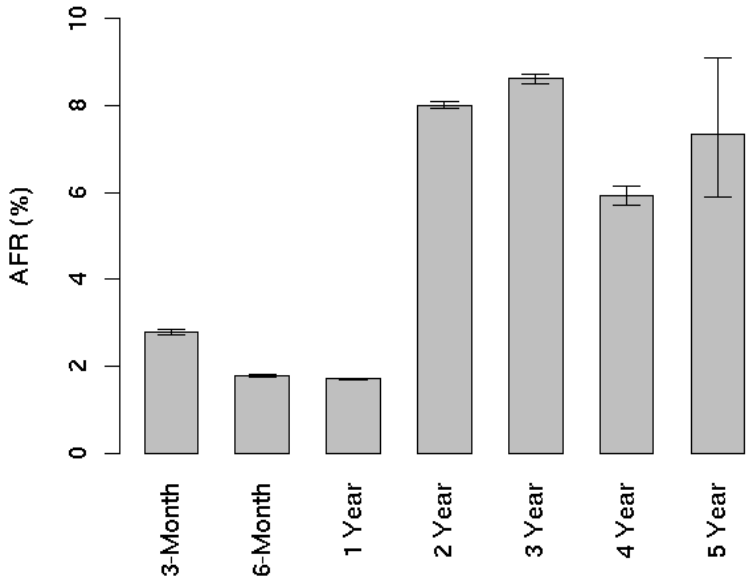


Figure 1.3: Annualized failure rates broken down by age groups

In this document we focus on large-scale backup system deployments that target either big organizations with PBs of data to back up or organizations that offer backup as a service, thus expect big volumes of data to be stored in their systems.

Traditional backup systems tend to combine data in huge tarballs and store them on tape-based systems. The reason for using tape is mainly related to their cost efficiency, compared to disk-based ones. Table 1.1[1] shows the costs related to the acquisition and operation of each type of system for a 5 year period. In the table, the operational costs contain the fees for the guarantee and the expected energy expenses throughout a 5 year usage. As we can see, tape-based systems are close to 4 times less expensive than disk-based ones.

Although the investment required for a tape-based system is smaller, this comes at the cost of reduced throughput. The reason is that tape is characterized by sequential access to data. While tape can provide a very high data

[1]Source: www.backupworks.com

	Tape	**Disk**
Acquisition cost	407, 000$	1, 620, 000$
Operational cost	205, 000$	573, 000$
Total cost	612, 000$	2, 193, 000$

Table 1.1: Cost comparison of Disk versus Tape based backup systems.

transfer rate for streaming long contiguous sequences of data, it takes 10s of seconds to reposition the tape head to an arbitrarily chosen place on the tape. This implies that random reads or writes are too expensive for this means of storage. In contrast, hard disks can perform the equivalent action in $10s$ of milliseconds, or 3 orders of magnitude faster, and can be thought of as offering random access to data. The importance of throughput in backup systems becomes more pronounced as the volume of data to be backed up increases and the time window available to perform the back up, termed backup window, decreases. This can be the case for data centers that receive big volumes of data by the minute and have to back them up, or medium to big size organizations that want to backup a day's or a week's work. In these cases, the system has to be fast enough to ensure the secure data storage while respecting the agreed time constraints.

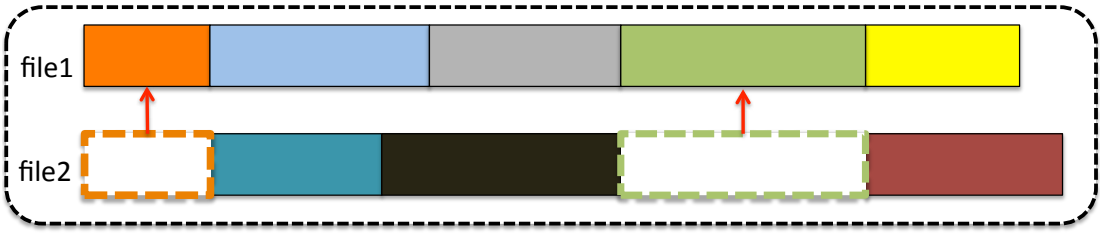


Figure 1.4: Deduplication Example

In this context, data deduplication provides an interesting alternative that can make disk-based backup solutions financially viable, even for small to medium sized organizations. Deduplication stands for the simple idea of replacing duplicate regions of data with references to already stored ones, thus reducing the space required to store a given workload. Figure 1.4 illustrates the process of storing a file in a deduplication system with an example. In the figure, `file1` is split into chunks of consecutive bytes and its chunks are stored in the system, while `file2` is about to be stored. After splitting `file2` in chunks, the system detects that the two files share the orange (first) and the green (fourth) chunks. In this case, instead of storing all the chunks belonging to `file2`, it only stores the unique ones and replaces the duplicate ones with references to the already stored orange and green chunks, that initially belonged only to `file1`.

Data deduplication can be seen as the waste management of the backup process. Most types of data tend to have whole regions of repeated bytes, and deduplication provides a way to get rid of this redundancy in order to reduce the space required to store them. It is not difficult to imagine that for an institution, backing up all its machines will result in keeping many copies of the operating systems that the company runs. Or in the case of e-mails, let's imagine that the human resources department sends to all personnel the new safety regulations as a `PDF` attachment. This would result in multiple copies of the attachment, when backing up the mail servers of the company. This phenomenon is even more pronounced in backup systems, where the workloads are, in most cases, updated versions of the same data. This implies that apart from the regions that were modified, the remaining data tend to be the same as in the previous version of the workload, thus leaving space for deduplication to reduce both the storage and the network needs of the backup process.

To further back up our claim that redundancy is ubiquitous in backup workloads, in [WDQ+12] the authors conduct a study of real backup workloads from EMC, the biggest company in data deduplication backup systems according to [pbb]. Their results show that, in their settings, deduplication rates range from 2.2 to 14, meaning that a given workload can reduce its space requirements up to 14 times by applying deduplication. In addition, the same company reports that if deduplication is combined with classic compression, the space needed for the backup of a given workload can be reduced up to 30 times [DDL+11]. To illustrate the importance of deduplication in the industry of backup services and also to illustrate the turn of clients' interest towards disk-based backup solutions, in [pbb], IDC reports that the worldwide market of data deduplication backup service experienced a growth of 43.4% in the year 2011 and the total revenue is estimated to 2.4 billion dollars.

Given the importance of backup systems and the potentials that deduplication has to offer in this context, in this document we focus on how to efficiently integrate deduplication in cluster-based storage systems. In Chapter 2, we discuss in details the challenges related to designing such systems, the tradeoffs involved and the solutions proposed so far, before presenting Produck, a cluster-based deduplication solution that combines novel contributions with state-of-the-art mechanisms to provide a more resource-friendly and efficient deduplication process.

1.2 Content Distribution

Safeguarding the content guarantees its integrity against dangers that threaten to corrupt it. But content is useful as long as it can be used by end-users, re-

searchers and organizations in order to extract "value" from it. This "value" can be new knowledge, as is the case for scientific applications, or user entertainment and service improvement, which is the case for social networks, video streaming, recommendation and many others. To this end, simply guaranteeing content integrity and (eventual) accessibility, is not enough. Access to the available content should be feasible in a timely manner.

Guaranteeing good service quality, both in terms of user-perceived latency and actual content quality to an increasing number of users all around the world is an open research problem for both academics and, of course, the industry. In addition, the huge and constantly increasing volume of data makes the problem even harder.

In this document we focus on User-Generated Content hosting sites, and more specifically YouTube, and we are investigating strategies to efficiently place videos on servers all around the world, in a way that will help the site to scale while providing good quality of service. We focus on YouTube for two main reasons. The first has to do with its magnitude. YouTube is the biggest user-generated short video sharing site in the world with 60 hours of video uploaded per minute and 4 billion videos watched per day. More statistics on YouTube can be found in Table 1.2[2]. In addition, traffic related to such services is expected to increase rapidly in the next few years. In fact, according to analysts [cis12], global internet video traffic is expected to be 55 percent of all consumer Internet traffic in 2016, up from 51 percent in 2011. And this estimation does not include video exchanged through peer-to-peer (P2P) file sharing. In addition, according to the same study, video-on-demand traffic will triple by 2016. The second reason, has to do with YouTube's "social" nature. The reason why we are interested in this type of workloads is that in recent years, online "social" networks have become one of the main information dissemination channels, influencing national election results, consumer behavior and, at the end, real social interactions. In addition, their success has led to many services integrating "social" aspects and we believe that this trend will continue for the following years. To this end, we believe that studying how to optimize infrastructure for this type of services is of utmost importance. YouTube, in this context, shares many characteristics with many of the most popular social networks, especially as far as content organization is concerned. To this end, we believe that focusing on YouTube is useful by itself but the conclusions can also be applied, to a certain extent, to other social networks.

The huge success of sites like YouTube creates huge resource needs in order for them to scale and be able to serve a constantly increasing and more and more demanding audience, spread all around the globe. To face this huge needs in storage and bandwidth, most UGC sharing sites (and not only) either maintain

[2]Source: http://www.youtube.com/t/press_statistics

60 **hours of video uploaded per minute**, 1 hour per second.
More than 4 **billion videos watched per day**.
More than 800 **millions users per month**.
More than 3 **billion hours of video watched per month**.
In 2011, YouTube had **more than** 1000 **billion views**.

Table 1.2: YouTube Statistics.

their own distributed infrastructure with servers all around the world, or employ Content Delivery Networks (CDNs) to offload some of the traffic from their servers and some of the costs related to infrastructure maintenance. CDNs are big distributed systems with servers deployed all around the world whose goal is to serve content to end-users on behalf of their clients with high availability and high performance. To do this, most of the CDNs deploy their servers within the data centers of different ISPs and the main question that they try to answer when storing an item is how to place it close to the users that are most likely to request it in the future. Accurately answering this question, allows CDNs to serve content with low latency and with minimal resource demands.

The goal of our study is to store videos on servers close to the users that are likely to request them in the future. This would allow the system to provide good service quality with minimal resource requirements. The principal property we leverage throughout our study is that items in YouTube's collection, but also in general for UGC sharing sites, are no longer independent the one from the other. By the time an item is uploaded to these sites, it becomes part of their content graph. This content graph may have different semantics, depending on the site in question but they all share some common characteristics. In Facebook users "befriend" other users, in Twitter they "follow" other users while in YouTube, a video is "related" to a number of other videos. Given the above relations, in YouTube the content graph consists of the videos as nodes and two videos being connected by an edge if they are considered as "related" by YouTube's recommendation mechanism. In Chapter 3, we verify previous findings that the position of an item in the content graph influences how users interact with it and we take this observation one step further by showing that it also influences the origins of the requests for a given video. Given this finding, we propose a proactive video placement mechanism that manages to place videos on servers located close to their future viewers.

1.3 Contributions

In Chapter 2 we focus on cluster-based deduplication systems. We start by presenting the state-of-the-art in duplicate detection techniques before focusing on

their application on storage systems. In addition to this, we also present a discussion on the challenges that system designers have to face when designing such systems. After this, we move on to describe Produck, a cluster-based deduplication solution that combines state-of-the-art techniques with novel ideas to provide good deduplication with minimal resource needs. In Produck we introduce a new set-intersection estimation mechanism that allows the system to efficiently detect duplicate regions of data with minimal overhead in terms of memory, network and computation. This set-intersection estimation technique, although it has been proposed in the field of the information retrieval, it is for the first time used in the context of storage systems. In addition, and to guarantee that load is equally spread among the nodes in the system, we introduce a novel, more deduplication-friendly load balancing mechanism that permits the system to be load balanced while keeping the deduplication efficiency high.

Chapter 3 focuses on content distribution. In this chapter we initially present the state-of-the-art in Content Delivery Networks and content placement techniques before diving into the specifics of user-generated content and how its structure can be leveraged to come up with novel placement strategies. In this context, we start by verifying results of previous studies that show that beyond traditional forms of locality that account for each content item independently, viewing patterns observed in UGC sharing sites are significantly influenced by the fact that content in these sites is no longer independent but is organized in a content graph [KZGZ11, ZKG10]. Pushing this finding one step further, we show that the position of a video in YouTube's content graph, plays an important role on where its future requests will come from. Leveraging this, we initially propose a mechanism that allows the system to predict where a video's future requests will come from and based on this mechanism, we propose a novel content placement mechanism that successfully manages to proactively place content close to where it is most likely to be requested. In addition, our mechanism makes minimal assumptions about the underlying infrastructure as it only assumes the existence of a distributed network of servers that host and serve content and an entity that forms the content graph, i.e. periodically runs YouTube's recommendation mechanism.

Finally, Chapter 4 summarizes the contributions of the above two chapters and presents some of the open problems related to them.

1.4 List of Publications

- Huguenin, K., Kermarrec, A.-M., Kloudas, K., and Taïani, F. **Content and Geographical Locality in User-Generated Content Sharing Systems.** Proceedings of the 22nd SIGMM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV) (2012).

- Frey, D., Kermarrec, A.-M., and Kloudas, K. **Probabilistic Deduplication for Cluster-Based Storage Systems.** ACM Symposium on Cloud Computing (SoCC) (2012).

Chapter 2

Probabilistic Deduplication for Cluster-Based Storage Systems

The need to backup huge quantities of data has led to the development of a number of distributed storage systems that leverage deduplication to reduce their network and storage space requirements. These systems aim to reproduce the operation of centralized, single-node backup systems in a cluster environment. At one extreme, stateful solutions rely on state stored for each node in the system to maximize deduplication. However the cost of these strategies in terms of computation and memory resources makes them unsuitable for large-scale deployments. At the other extreme, stateless strategies store data blocks based only on their content, without taking into account previous placement decisions, thus reducing the cost but also the effectiveness of deduplication.

In this chapter, we present, Produck, a stateful, yet lightweight cluster-based backup system that provides deduplication rates close to those of a single-node system at a very low computational cost and with minimal memory overhead. In doing so, we provide two main contributions: a lightweight probabilistic node-assignment mechanism and a new bucket-based load-balancing strategy. The former allows Produck to quickly identify the nodes that can provide the highest deduplication rates for a given data block. The latter efficiently spreads the load uniformly among the nodes. In our experiments, we compare Produck against state-of-the-art alternatives over a publicly available dataset consisting of 16 full Wikipedia backups, as well as over a private one consisting of images of the environments available for deployment on the Grid5000 experimental platform. Our results show that, on average, Produck provides (i) up to 18% better deduplication compared to a stateless minhash-based technique, and (ii) an 18-fold reduction in computational cost with respect to a stateful Bloom-filter-based solution.

2.1 Introduction

Cloud providers, social networks, data-management companies and on-line backup services are witnessing a tremendous increase in the amount of data they receive every day. This phenomenon implies huge storage needs that increase exponentially [GCM+08b] as more users join and already existing ones become more engaged in the offered services. For such companies, the data they store constitutes part of their business and potential data disruption could lead to decreased revenues due to loss of credibility or service deterioration. To minimize this danger and also for legal requirements [DDL+11], data centers need to periodically backup full copies of their data for periods of up to several years and have them available for retrieval upon request.

As mentioned in Chapter 1, until recently, most backup systems combined files into huge tarballs and stored them on tape in an effort to minimize costs [DDL+11]. However, the decrease in the cost of magnetic disks combined with techniques like deduplication that can make more efficient use of storage space have made disk-based backup solutions more popular. Deduplication identifies identical chunks of data and replaces them with references to a previously stored copy of the chunk [MB11], thereby reducing the required storage space. Using deduplication, single-node commercial backup systems are now capable of storing petabytes of data to disk [GE11]. Yet, the backup needs of modern data centers are already surpassing this limit [GCM+08b], and the amount of data to backup is bound to increase even further as more and more companies outsource much of their infrastructure to the cloud.

An appealing approach to address these increasing requirements is the design of cluster-based backup platforms. Cluster-based systems permit the capacity of the system to scale by adding more nodes, instead of replacing the whole system with a more powerful one. In addition, existing infrastructure can be used and expanded, when needed, using (almost) commodity hardware. Integrating deduplication into cluster-based solutions, however, is a challenging task [DDL+11, DGH+09, BELL09]. In a single-node system, the main difficulty is to identify at which granularity duplicate data regions should be detected (i.e. chunk size) so that deduplication is maximized while throughput stays high. In these systems, as we will see in Section 2.2, the main concern is how to structure the chunk index so that the disk is visited as few times as possible during the duplicate detection process, a problem termed the "disk bottleneck problem". A cluster, however, introduces a new challenge: assigning each chunk to a cluster node while (i) maximizing deduplication, and (ii) balancing the storage load on the available nodes. As we will see in Section 2.2, these two goal are at odds: the requirement for a load balanced system often leads to placing a given group of chunks on a different node than the one with whom it shares most of its chunks.

The solutions that researchers have proposed in the field of cluster-based deduplication systems can be broadly categorized into two categories. So-called stateless solutions [DDL+11, BELL09] assign data to nodes based only on properties of the contents of the data being stored. This, in many cases, provides a somewhat natural form of load balancing, due to its randomized nature, but it yields suboptimal deduplication performance as new assignment decisions do not take into account previous ones. At the other extreme, stateful approaches maintain information about the current state of each node in the system (i.e. what each node stores) in order to assign identical data blocks to the same node. This yields much better deduplication, but with two main drawbacks. First, stateful techniques require more complex load-balancing strategies to avoid storing everything on the same node. Second, existing solutions require significantly more computing and memory resources to index all the stored information. This leads deployed systems, such as [DDL+11] to operate according to a stateless model, at least until a practical stateful system becomes available.

In this document, we attempt to satisfy this need by proposing Produck, a lightweight cluster-based backup system that aims to make stateful deduplication usable in practice. Produck achieves deduplication rates that are close to those of a single-node system, where there is no requirement for load balancing, by combining state-of-the-art techniques with novel contributions. First, similar to [DDL+11], Produck addresses the tradeoff between small and large chunk sizes with a two-level chunking algorithm. It routes and stores data based on relatively large superchunks, composed of a number of smaller chunks that constitute the minimum information units for deduplication. Second, it incorporates two novel contributions that enable it (i) to achieve stateful superchunk assignments with minimal CPU and memory requirements, and (ii) to balance the load on cluster nodes without hampering deduplication performance.

The first of these contributions is a probabilistic similarity metric that makes it possible to identify the cluster node that currently stores the highest number of chunks out of those that appear in a given superchunk. This metric is based on a probabilistic technique for computing set intersection that originates from the field of information retrieval [MBN+06], and which, to our knowledge, has never been used in the context of storage systems. The second novel contribution of Produck is a deduplication-friendly bucket-based load-balancing strategy. Specifically, Produck uses fixed-size buckets to measure the deviation of a node's disk usage from the average. This facilitates the aggregation of similar superchunks, i.e. superchunks with many common chunks, on the same nodes even in the initial phases of a backup process, ultimately yielding better deduplication.

Produck can be deployed as a stand-alone system, but it can also be used as a middleware platform to integrate existing high-throughput single-node deduplication solutions. This allows users to easily integrate new techniques, leverage

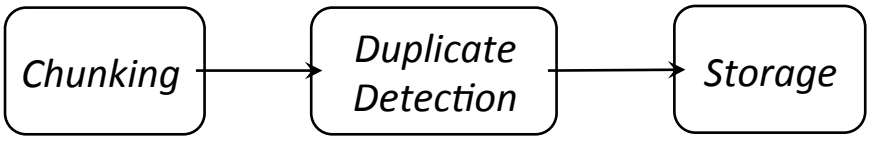


Figure 2.1: Storage Deduplication Workflow.

existing research results and encourages Produck's adoption in a wide range of backup services. In this chapter, we focus on the evaluation of Produck as a stand-alone system. We thus leave a study of its integration with existing platforms as future work.

2.2 Related work

In this section we present the existing work in the context of duplicate detection. We start by presenting the mechanisms used to efficiently detect identical regions of data, before diving into the application of duplicate detection and elimination in the context of storage systems. Figure 2.1 presents the workflow for the deduplication process, that can also serve as a roadmap for this section. Although the figure focuses on backup systems, which are the main focus of the chapter, the first two steps are the same for every system that uses duplicate detection to achieve its purpose. Before being deduplicated, data that are about to be stored, are initially split into chunks, which constitute the basic duplicate detection unit. After chunking, duplicate chunks are replaced by references to already existing ones. This is done during the duplicate detection process. Finally, the deduplicated data are stored in the system.

This section starts by presenting the different chunking algorithms that have been deployed so far, along with the advantages and the disadvantages of each of them. After chunking, we move on the application of deduplication in archival storage systems and we present the state-of-the-art in single-node and cluster-based systems. For each one of these types, we present the goals, the challenges to be faced, and the solutions that researchers have given to these challenges.

2.2.1 Duplicate Detection

Duplicate detection has been applied in many fields ranging from file systems [MCM01, UAA+10] and fast data transfers [PAK07] to copy-detection mechanisms [BDGM95]. As a consequence, multiple algorithms have been proposed to detect redundancy in large sequences of raw bytes, depending on the requirements of each specific application.

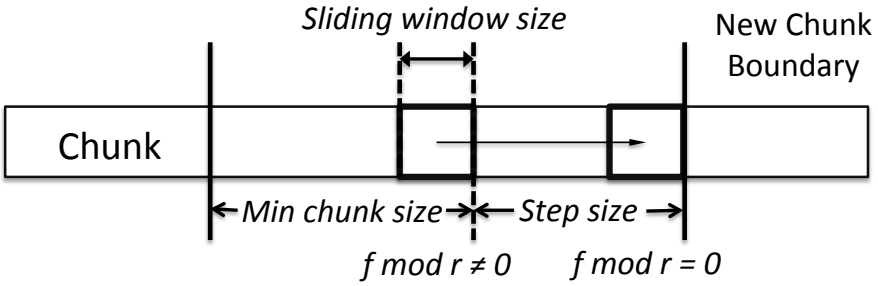


Figure 2.2: Content-Based Chunking Principles.

In the general methodology, a given workload is, at first, split into sequences of consecutive bytes that from now on we will call chunks. Chunks constitute the basic similarity unit, as unique chunks are stored only once and duplicates are replaced by references to already existing ones. To compare chunk contents efficiently, a collision resistant hash function, such as SHA-1, is applied on the contents of each chunk and its output is used as the chunk's signature or fingerprint. In the remaining of this chapter, the two terms, signature and fingerprint, will be used interchangeably. To see if two chunks are identical, it is sufficient to compare their signatures. Comparing the contents of two chunks based on their SHA-1 hash value is a lot more efficient than comparing the chunks byte by byte, as chunk sizes range from 4 to 64KB while the SHA-1 signature of a chunk accounts for only 20-bytes. In addition, relying on only the signatures to compare chunk contents can be considered safe, as the probability of two non-identical chunks having the same fingerprint is many orders of magnitude lower than that of a hardware error [QD02].

Initial solutions for duplicate detection included whole-file chunking [ABC+02] and fixed-size chunking [RCP08] algorithms. In the former, deduplication is performed on a file granularity while in the latter, files are split into chunks of fixed size. These solutions, in many cases, failed to accurately detect duplicate data regions. In the whole-file case, a small modification in the content of a file, even at the end of it, would change its signature, as this consists of the hash value of its contents, thus leading to consider the two versions of the file as totally different. While in the fixed-size chunking techniques, an insertion of even one byte would shift all the boundaries of the chunks following the region of the modification, resulting on all signatures being changed, thus not detecting the similarity of two files even if they share most of their contents.

To fight the drawbacks of the above solutions, in [MCM01] the authors introduce Content-Based Chunking (CBC). In this work, the authors use deduplication in the context of network file systems and they introduce CBC to reduce network traffic by detecting chunks of data that are already present on the client. In Pro-

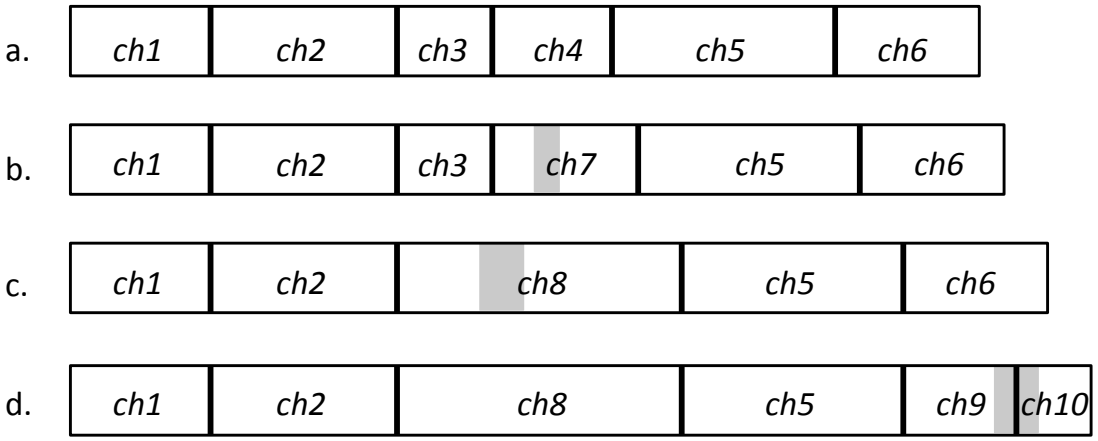


Figure 2.3: Chunks of a file after various modifications. Gray regions are the modifications while the vertical lines are the chunk boundaries.

duck, we use CBC to detect duplicates and Section 2.3.2 provides a more in-depth description of the implementation of the algorithm. Figure 2.2 presents the basic principles of CBC, so that the reader can compare it to previously described chunking algorithms. CBC uses Rabin fingerprinting over a sliding window to declare chunk boundaries. A minimum and maximum chunk size are provided as parameters along with parameter r, the size of the sliding window and the step size, as shown in the figure. A chunk boundary is declared when the lower order bits of the Rabin hash of the bytes in the sliding window, f, match a certain pattern ($mod\ r = 0$). Declaring boundaries based on byte patterns helps the algorithm be immune to small modifications.

To illustrate the resistance of CBC to small modifications, Figure 2.3 presents how chunks are affected by different edits of a given file. In Figure 2.3, a. shows the chunks of the file before the modifications. As we can see chunks do not have the same size as boundaries depend on data properties. In the case of b. some bytes are added in the middle of ch4. The result is that the modification is absorbed by ch4, creating ch7. In this case chunk boundaries are not affected and the same holds for their signatures, apart from ch7. In c., a modification in the file eliminates the boundary of ch3. The result is that ch3 and ch7 are combined in ch8 and, again, the signatures of the chunks not touched by the change, stay the same. Finally, in d., a modification introduces a new chunk boundary resulting in ch6 being split into ch9 and ch10.

Many studies [KDLT04, MB11, PP04] compare the above file-chunking strategies and the main finding is that CBC outperforms its counterparts in duplicate detection.

In CBC, one of the most important parameters is the chunk size used, as this

determines the granularity of the deduplicate detection process. Although in CBC the chunk size is not fixed, the system designer can define the average size of the chunks, by carefully choosing the values of the minimum and maximum chunk size and r. The size of chunks used affects both deduplication efficiency and system performance. On the one hand, smaller chunks lead to better deduplication as identical sequences of data are detected at a finer granularity. On the other, with smaller chunks, there are more hashes to be processed, which can reduce performance. At a minimum, more chunks mean more comparisons to determine duplicates, but they are also likely to mean more on-disk index lookups, if the index of already stored chunks cannot fit in memory. In addition, when it comes to metadata maintenance, more chunks mean more metadata to be stored and kept up-to-date. Finally, if metadata have to be sent through the network during the deduplication process, as in the case of backup systems, network traffic can also become an issue.

To this end, much effort has been invested in determining the right chunk size for a given workload. More specifically, the goal is to detect the maximum chunk size that will give good deduplication results while minimizing the size of the metadata and the computation involved in the deduplication process.

In [TPAK10], the authors propose the use of a compact, multi-resolution fingerprinting methodology that tries to estimate similarity between data items at different chunk sizes. According to this, a file is chunked based on CBC using different chunk sizes (resolutions) and the final signature of the file is composed of a mix of samples of signatures taken from these different resolutions. To determine which chunk signatures to include in the final one content-based sampling is used, i.e. sampling is based on bit patterns of the hashes. The main contribution of this study is the computation of bounds on the number of samples that have to be included in the fingerprint to achieve a required level of accuracy in the estimation.

In [KUD10], the authors study the same problem in the context of deduplicating streams of data. The solution they propose is based on the observation that during repeated backups, long regions of data may be duplicates, and even when changed, the changes may be localized to a relatively small edit region. Based on this assumption, they focus on detecting "change regions", which stand for non-duplicate series of bytes between big duplicate chunks. To do this, they propose to initially chunk files using a big chunk size and when one of them is detected as non-duplicate while there are duplicates before and after that, then this chunk is further split in smaller chunks and re-deduplicated. In addition to this split-apart approach, the authors also propose a chunk amalgamation algorithm where files are initially split in small chunks, and series of consecutive duplicate small chunks are combined into bigger ones. The goal of this algorithm is to reduce the size of the metadata that have to be indexed by the deduplication server and the

number of hashes to be processed during deduplication.

As mentioned earlier, many studies compared the different chunking strategies presented above and the main conclusion is that content-based chunking outperforms its counterparts in terms of achieved deduplication. This is the reason that, as we will see in the paragraphs that follow, most of the systems that use deduplication as a means to reduce data volume, use CBC. The same holds for Produck. As far as the choice of the chunk size is concerned, the techniques to detect the best chunk size for a given workload are complementary to our work and they can be easily integrated.

2.2.2 Single-node Deduplication Systems

As illustrated in Figure 2.1, after the chunking stage comes the duplicate detection one, where the signatures of the new chunks are compared against the ones of the chunks already encountered by the system. The goal of this process is to detect duplicates, i.e. chunks with identical content. As explained in the previous paragraph, signature comparison can be considered sufficient and there is no need to compare chunk contents byte by byte. To make duplicate detection efficient, the signatures of already stored chunks are kept in an index, that, from now on, we will call chunk index.

In this paragraph we are interested in single-node backup systems that use deduplication to reduce the storage space required to back up a given workload. Cluster-based storage systems, which also constitute the main focus of the chapter, are presented in the next paragraph of the section. The goals of a single-node system can be summarized in the following three:

1. good deduplication,
2. good throughput, and
3. support of large disk capacities.

Good deduplication stands for the requirement that the duplicates have to be detected and eliminated as much as possible. In the ideal case, there should be no duplicate chunks among the data written to disk after the duplicate detection process. The reason why this is not always possible can be found in the remaining two requirements, good throughput and support of big disk capacities.

Focusing on throughput, we are interested in the speed at which the system is able to receive a given workload, replace duplicate chunks with references to already stored ones, and store the unique ones on disk. In this process, duplicate detection can easily become the bottleneck if the chunk index is not efficiently structured and organized, as we will see in the paragraphs that follow.

In small, single-node deployments, to make the duplicate detection process fast and accurate, it is sufficient to keep the chunk index in main memory. This can be feasible due to the small disk capacity supported by the system, that implies a relatively small number of unique chunks. Having the whole chunk index in main memory allows the system to accurately detect duplicates without touching the disk during the duplicate detection process, thus providing good performance.

Unfortunately, in larger deployments, where big volumes of unique chunks are expected to be stored, it is a matter of time till the chunk index grows beyond the size of the available main memory, and starts spilling to disk. In this context, the main challenge is to organize the index in a way that minimizes disk accesses during the duplicate detection process. The reason behind this is that, given the relatively small chunk size, which in the state-of-the-art systems ranges from 4 to 64 KBs, the chunk index can easily account for gigabytes of data and a given workload that needs to be stored can have chunks in the order of millions. If for each newly arriving chunk the system had to search the whole index that is partially stored on disk to decide if it is a duplicate, this would incur a severe performance penalty that could make the whole system impractical. This problem is often termed in the bibliography as the "disk bottleneck problem" and constitutes one of the main design issues in single-node deduplication systems.

To minimize the impact of the "disk bottleneck problem", almost all proposed solution leverage a property inherent to backup workloads that in the literature is termed "data locality" or "chunk locality". This property stands for the fact that in consecutive backups of a given workload, if chunks A, B and C appear close to each other, then the next time chunk A is encountered, there is a high probability that chunks B and C will also appear later on. In other words, "data locality" implies that the probability of a chunk being a duplicate increases if its preceding chunk is one. In addition, this property is also present in backups of different files. For example, when backing up all the machines in an institution, several copies of the same OS would be stored if no mechanism is in place to prevent it.

Leveraging "data locality", single-node deduplication systems [DSL10, BELL09, LEB+09, XJFH11, GE11] organize the chunk index in two tiers, with one residing in main memory and the other being stored on disk. Series of consecutive chunks are organized by the storage nodes into segments (or blocks or containers) and only a compact representation of the chunks in a given segment is kept in memory. The full chunk index of a given segment is stored on disk. Often, the part of the segment kept in memory consists of a sample of the signatures of the chunks belonging to the segment. When a new group of chunks has to be stored, the in-memory index is searched and for each fingerprint match, the full index of the corresponding segment is fetched from disk and cached for further dedu-

plication. Given the "data locality" property, it is expected that a small number of segments will have to be fetched to deduplicate large groups of chunks, thus limiting the number of disk accesses required. The information in the in-memory tier serves as a hook to decide which parts of the on-disk index will be cached in memory. Another thing that we have to note, is that now, a segment becomes the basic I/O unit for reading and writing to disk. This is a design choice made to further improve performance, as magnetic disks are more efficient at reading and writing relatively big blocks of data.

In Extreme Binning [BELL09] the authors focus on workloads that consist of individual files with no locality among consecutive files in a given window of time. In these settings, each segment corresponds to a file and for each one, a representative fingerprint, the hash of the contents of the file and a pointer to the disk location where its full chunk index is stored, is kept in memory. As a representative fingerprint, the authors use the minimum hash value (minHash) or minimum fingerprint of the chunks belonging to the file. When a new file arrives, its minHash is compared against the minHashes of the files stored in the system. If no match is found, an entry is added to the in-memory index and the full index of the chunks in the new segment is stored on disk. If the minHash matches the one of an already existing file, then the file is deduplicated against the existing one. This methodology guarantees that the disk is accessed only once per file stored in the system.

In Sparse Indexing [LEB+09], the authors focus on traditional workloads where chunk locality is relatively strong. In this work, segments are created in a content-based way, as in CBC, by observing bit patterns at the level of chunk fingerprints. This means that a chunk is declared to be a segment boundary if the lower order bits of its fingerprint match a certain pattern. This results in segments of variable size. After segmentation the chunks of each segment are sampled, again, in a content-based way, and the resulting subset is kept in memory as the signature of the segment while a full index of the chunks in the segment is kept on disk. To further reduce disk accesses, each incoming group of chunks is deduplicated against a limited number of the most similar already stored ones. Similarity between the new group of chunks to be stored and a given segment is measured by the number of fingerprints they share in their signatures. Only the full indices of the "winning" segments are fetched from disk. This technique is shown by the authors to result in small deduplication losses.

SiLo [XJFH11] operates in the same way as [LEB+09], with the difference that segments are further grouped into in blocks, to further mine data locality.

In ChunkStash [DSL10], the authors propose the use of flash memory instead of magnetic disk to store the full chunk index. To minimize false flash accesses, ChunkStash keeps in main memory a compact index of the location of each chunk on the flash disk. When a new chunk arrives, its is checked against this compact

index and if there is a match, then the pointer to the flash memory is followed for further investigation. When the pointer to flash is followed, the full segment that this chunk belongs to is fetched and cached. The above structure along with careful engineering allows ChunkStash to report speed-ups of up to 60x compared to traditional disk-based solutions.

Finally, in [GE11] the authors argue that the costs related to acquisition, maintenance and energy consumption of cluster-based deduplication solutions, make them unattractive to small and medium-sized companies. As a result, they focus on increasing the capacity that a single-node deduplication system can sustain. To reduce the memory needs of the chunk index, they use sampling and they propose a progressive sampling rate technique that uses the whole memory space and increases the sampling rate (decreases the number of chunks included) as more data are added to the system. Index sampling is performed in a content-based way–a given fingerprint is included in the the index if its r least significant bits are 0 (e.g. for a sampling rate of 1/8, r is set to 3). This allows them to downsample the index without having to recheck the full index. In addition, they introduce the problem of reference management. This refers to the problem of efficiently and safely updating the indexing structures (both in-memory and on-disk) in the face of data deletions. This problem is more pronounced in deduplication systems than in general storage systems as, here, a chunk may be referenced by multiple files. This means that even if a file is deleted, its chunks cannot be directly deleted, as this could corrupt other files as well. To tackle this, they introduce a technique called grouped mark-and-sweep. In grouped mark-and-sweep, additional data structures are maintained that group files together and permit to monitor changes happening on a groups-of-files granularity thus resulting in touching only the metadata of chunks that were actually affected by the deletions, and not having to scan the whole index during the mark-and-sweep process.

As shown in this paragraph, much effort has been invested during recent years to improve single-node deduplication systems in order to provide good and fast deduplication while supporting large disk capacities. Most of the recent solutions leverage "data locality" to minimize disk accesses during the duplicate detection process but they differ on how they structure their index. Some of them assume the use of new hardware, as in the case of ChunkStash, while others do not. In addition, in all solutions, apart from ChunkStash, in order to provide good throughput they sacrifice some of the deduplication efficiency of the system. In these solutions, chunks are grouped in segments and each new segment is deduplicated against other segments that contain chunks that are already stored in the system. The decision of which segments to fetch from disk to deduplicate a new one that arrives to be stored, is based on a compact segment descriptor, which is often a sample of the signatures of the chunks in the segment. This organization

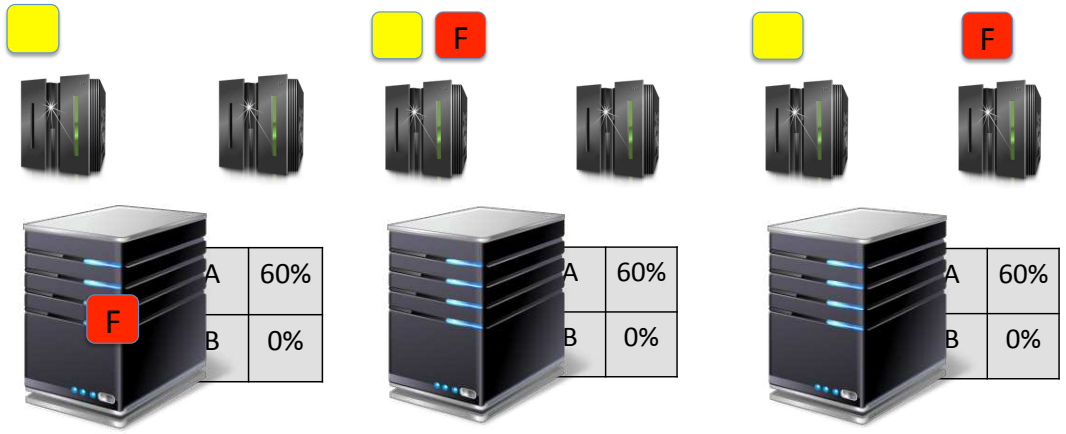


Figure 2.4: Tension between Load Balancing and Deduplication : at the beginning a file wants to be stored, who shares 60% of its content with the chunks stored on node A. If it were to optimize for deduplication, the new file should go on node `A`, while optimizing for load balancing would imply sending the chunk to node `B`.

may lead to some duplicate chunks not being discovered thus leading to reduced deduplication efficiency. All the above work on single-node deduplication systems is complementary to our work as Produck is agnostic to the architecture of the single-node systems used as storage nodes.

2.2.3 Cluster-based Deduplication Systems

Although single-node systems perform well and manage to efficiently decrease the storage needs for backing up a workload by up to 30 times [DDL+11], the explosive increase in the data created and stored has led researchers and companies to turn to multi-node solutions that are able to scale by adding more hardware instead of having to replace the whole system with a more powerful one.

For the system to scale it has to be able to make maximum use of the resources that more nodes have to offer. To achieve this, the system has to be able to equally spread the load among the available nodes. In other case, some nodes will end up overloaded and unable to perform at the expected level, while others would stay idle, thus resulting in lost investment. From the above discussion, it becomes clear that a multi-node system should be able to provide the same properties as single-node systems, while keeping the system load balanced.

Figure 2.4 presents the tension between data deduplication and load balancing with an example. In the figure, we assume a system with two storage nodes, `A` and `B`, and a central coordinator node that decides on which storage node to store each group of chunks. Consider a newly arriving group of chunks `F` (red) arrives

to be stored. As we can see from the first figure, `F` shares 60% of its content with node `A` and 0% with node `B` which is empty. In this scenario, if it were to optimize for deduplication, the new chunks (`F`) should end up on node `A` while if it were to keep the system load balanced, `F` should be sent to node `B` which is empty. In general, one can see that for data deduplication the ideal case would be to store all data on the same node, while load balancing tries to equally spread them among the nodes available to the system.

Solutions proposed so far in multi-node or cluster-based deduplication systems can be broadly classified in two categories. On the one hand, there are stateless solutions where assignment decisions are taken based on the contents of the chunk itself and no information about previous assignment decisions is stored. On the other, there are stateful solutions that keep information about previous chunk assignment decisions (i.e. which node stores which chunk), and new assignment decisions take into account this knowledge. A comparison between the two would reveal that stateless solutions are a lot more resource friendly than stateful ones, as there is no need for storing metadata about previous assignment decisions, but they provide worse deduplication than stateful ones who sacrifice resources to achieve more accurate chunk placement.

We start our description of the state-of-the-art on cluster-based deduplication solution by presenting HYDRAstor [DGH+09] and HydraFS [UAA+10], a filesystem based on HYDRAstor. HYDRAstor, is a deduplication storage system built on top of a Distributed Hash Table (DHT). Here, the fingerprint space is partitioned evenly across storage nodes using consistent hashing and each chunk is routed to a node based on the hash of its contents. This results in duplicate chunks being routed to the same node. Chunk assignment decisions in HYDRAstor are made on a per chunk granularity. This led the authors to use relatively big chunks (64KB) in order to provide good throughput. Although big chunks allow HYDRAstor to provide good throughput, this comes at the cost of reduced deduplication efficiency as the bigger the chunk, the lower the granularity at which duplicates are detected. For load balancing, HYDRAstor relies on the uniform distribution of the size of the chunks along with the fact that due to its routing scheme (the same as in DHTs), nodes are expected to receive approximately the same number of chunks. HYDRAstor assigns chunks to nodes based solely on the content of the chunk itself thus lending itself to the stateless deduplication solutions.

Although HYDRAstor is completely distributed, as it is built on top of a DHT and there is no central coordinator, other solutions, including Produck, prefer a more centralized approach where a node is used as a coordinator and the remaining nodes in the cluster constitute the storage nodes.

Two system architectures that prefer this more centralized approach are the ones presented in [DDL+11]. There the authors show the routing method pro-

posed by HYDRAstor is suboptimal when it comes to backing up big collections of files. Instead, and to further mine data locality, they organize chunks into superchunks, which are groups of consecutive chunks, and they route chunks belonging to the same superchunk on the same node. This makes it possible to improve performance, as assignment decisions are taken on a per-superchunk instead of a per-chunk basis, while using small chunks (4KB), thus having the benefit of being able to detect duplicates at a finer granularity than HYDRAstor. The two strategies proposed in this work differ on the way assignment decisions are taken. One of them routes superchunk according to the stateful model while the other according to the stateless. In the remainder of the section we present these two strategies in detail as these are the two strategies that we compare Produck against.

We start by presenting the stateful strategy presented in [DDL+11], which from now on will be refered to as BloomFilter. According to the BloomFilter strategy, the Coordinator maintains a bloom filter for each storage node as an index of the node's contents. It then uses these bloom filters to assign each superchunk to the node that already stores most of the chunks it contains. To compute the overlap between a node's contents and the chunks in a new superchunk, the Coordinator checks the fingerprints of the chunks in the superchunk against the node's bloom filter. To improve performance during the bootstrap phase, the Coordinator ignores overlap values that are below a dynamically computed threshold. Moreover, to balance the load in the system, it considers a node eligible to store a superchunk only if its load does not exceed the average load of the system by more than 5%. In [DDL+11] the authors observe that keeping in the bloom filter all chunks stored on a given node and chicking all the chunks in a given superchunk against it (i) uses a lot of memory and (ii) makes the duplicate detection process too slow. To this end, they propose a variation where chunks in a superchunk are sampled in a content-based way as described earlier and only 1/8th of them is kept in the bloom filter. This allows them to reduce the memory needs for the bloom filter and speeds up duplicate detection. We compare against both variations of BloomFilter in Section 2.6 of this chapter. This strategy is stateful since the Coordinator keeps track of what each StorageNode stores and when deciding where to store a new superchunk, it takes into account the location of existing ones.

The second strategy presented in [DDL+11] is MinHash. MinHash selects the minimum hash (minhash) of the chunks in a superchunk as the superchunk's signature. This strategy uses the additional abstraction of bins and instead of assigning superchunks to nodes, it allocates superchunks to bins, which it then assigns to nodes. This abstraction is used by the authors to keep the system load balanced. Typically $\#bins \gg \#nodes$. When assigning a superchunk to a bin, MinHash applies the $modulo(number\ of\ bins)$ operator to the superchunk's

minhash value in order to get the index of the destination bin. Although at first this strategy seems to be load balanced as bins are expected to be assigned the same number of superchunks on average, this turns out to be false as some nodes end up being overloaded. To keep the system load balanced, when a node becomes overloaded, i.e. when its load exceeds the average by more than 5%, MinHash reassigns bins to nodes (reshuffling) so that no node is overloaded. This implies that although MinHash is a stateless strategy, this is not completely true as the Coordinator has to keep track of which chunks are assigned to which bin in order to do the reassignment. If not, then the Coordinator is in no position to judge which combination of bins leads to a balanced system. Although this is true, reassigning bins to nodes rarely happens, as pointed out by the authors, so we still consider this strategy as stateless.

A comparison between the two categories of architectures, stateful and stateless, reveals that stateful strategies manage to have better deduplication at the cost of using a lot more resources in terms of memory and computation, while the stateless ones sacrifice deduplication to be more resource friendly . This is also the reason why in their production system, the authors of [DDL+11] use the stateless strategy. These strategies are good examples of cluster-based deduplication solutions and we compare against them in Section 2.6.

2.3 PRODUCK Principles

After presenting the state-of-the-art in cluster-based deduplication solutions, here we focus on the contributions of this chapter, and we present Produck, the system we designed for cluster-based deduplication solutions. We start by analyzing the main features in the design of Produck. We present its architecture, we provide the required background on Content-Based Chunking in detail, and finally, we introduce Produck's two main contributions: its chunk-assignment protocol, and its load-balancing strategy.

2.3.1 Architecture

Produck's architecture, depicted in Figure 1, is the same as most cluster-based deduplication systems and consists of three main entities: a Client application that may run on the machine of the user that wants to backup her data or on a machine under the control of the backup service provider, a Coordinator node, and a set of StorageNodes. The last two entities, are under the control of the service provider.

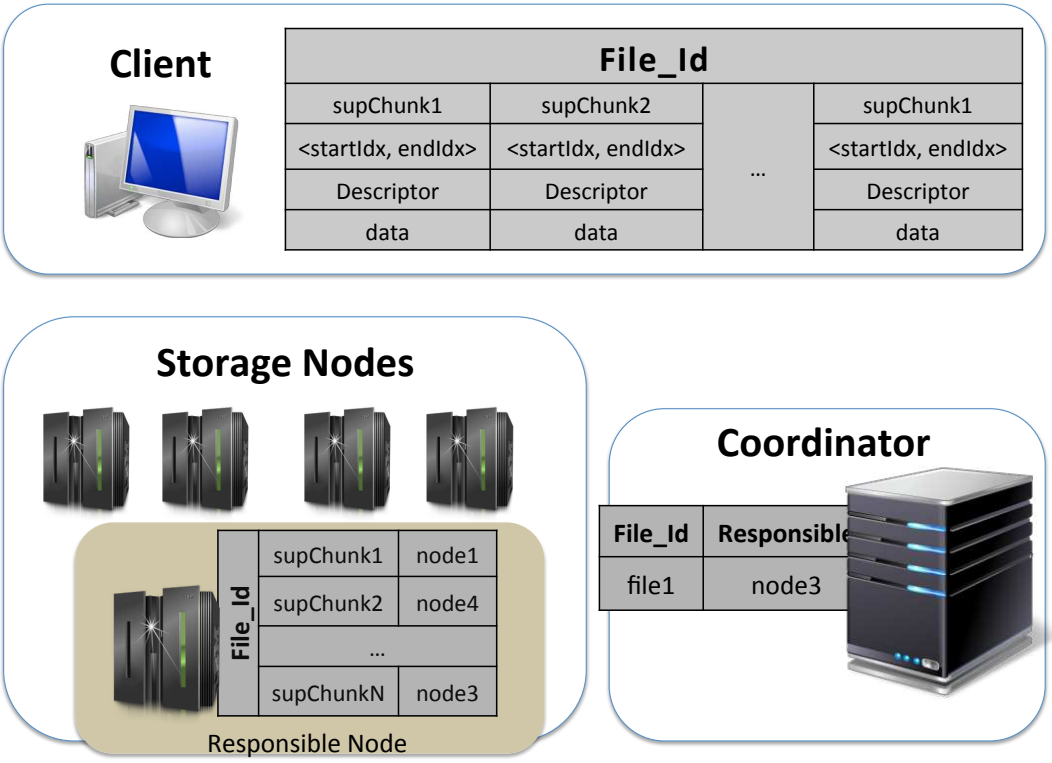


Figure 2.5: System Architecture

Client

The Client constitutes Produck's frontend, and thus provides the entry point through which Produck's users can interact with the backup system. The Client offers facilities to manage stored data, such as listing the contents of a backup. In the following, however, we concentrate on its two fundamental tasks: storing and retrieving a file. When storing a file, the Client is responsible for translating it into a set of chunks that it will then deliver to the StorageNodes. When retrieving a file, the Client is responsible for i) obtaining the chunks that belong to the file from the StorageNodes and ii) recomposing them into the original file. We describe the details of the chunking process in Section 2.3.2.

Coordinator

The Coordinator node is responsible for managing the requests of Clients. When storing a file, the Coordinator is responsible for (i) determining which StorageNode should store each chunk, and (ii) putting Clients in contact with the ap-

propriate StorageNode which will store the data. When it comes to retrieving a file, the Coordinator is the first node to be contacted by a Client and will then redirect the request to the appropriate StorageNode, as we will see in the paragraphs that follow. After this step, the Clients communicate directly with the StorageNodes for both operations, i.e. storing and retrieving content. From the above description, one can understand that the role of the Coordinator is crucial for the overall performance of the system, as it is the node responsible for achieving both deduplication and keeping the system load balanced. In Sections 2.3.3 and 2.3.4, we present how the Coordinator implements our novel chunk assignment and load balancing strategies. In the current version of Produck, there is only one Coordinator for the entire system. Federating multiple Coordinator nodes for higher throughput or fault-tolerance can be achieved through standard techniques.

StorageNode

StorageNodes perform a key role in our backup system by providing storage space according to the assignment decisions made by the Coordinator. As shown in Figure 1, however, StorageNodes also carry out an additional task, that of acting as responsible nodes, i.e. providing directory services for Clients that need to retrieve the chunks of a file that are stored on multiple StorageNodes. The Coordinator randomly selects a responsible node for each file among the StorageNodes. This offloads the Coordinator from the management of read requests and file metadata. We describe the details of the interactions between Clients, the Coordinator, and StorageNodes in Section 2.4.

2.3.2 Chunking

As outlined in Section 2.1, a deduplication system operates by splitting files into chunks and identifying identical chunks within one and across multiple backup operations. A key performance tradeoff is associated with the size of a chunk (Section 2.2). As in a single-node system, smaller chunks make duplicate detection more efficient as they search at a finer granularity, but result in a greater overhead for storing the associated indexing and metadata information. In addition, smaller chunks can also limit the achievable throughput in the case that all system operations are performed at a chunk granularity. Disks are more efficient when accessing continuous data, while the use of very small chunks results in less efficient random-access patterns.

Similar to [DDL+11], Produck addresses this tradeoff by using a two-level chunking algorithm. Specifically, it uses relatively small chunks as deduplication

units ($1KB$). However, it groups contiguous chunks into large superchunks when storing data onto storage nodes. This makes it possible to optimize data transfer by dealing with large amounts of contiguous data, while retaining the deduplication advantage associated with small data chunks. The chunking algorithm is executed at the Client as soon as the user requests the backup of a file. In the following paragraph, we describe its details by explaining how it creates chunks and superchunks.

2.3.2.1 Content-based chunking

The most intuitive way to split a file into chunks is to use fixed-size chunks starting from the beginning of the file. However, as described in Section 2.2, this approach suffers from a fundamental drawback when new data is added or removed from the beginning or in the middle of a file. The contents of the chunks that follow the location of the change are shifted, and the boundaries of the corresponding chunks change. This results in all chunks following the "change region" to be considered new, even if they are essentially the same. This is a major issue in a deduplication backup system that aims to identify duplicate chunks across multiple versions of the same files or across different files.

To address this issue, Produck uses, like most production deduplication systems, content-based chunking (CBC). Instead of using fixed chunk sizes, Produck determines chunk boundaries based on the data contained in the chunks, as shown in Figure 2.2. Specifically, it defines a minimum and a maximum chunk size, along with a step size, $l_{\min}$, $l_{\max}$ and l_{stp}, and defines chunk boundaries as sequences of w bytes that (i) cause the chunk size to remain within $l_{\min}$ and $l_{\max}$, and (ii) satisfy a condition on their hash values. Let W be a window of w bytes, let f be its fingerprint consisting of the Rabin hash value of its contents, and let D and r be two integer values. Then, W is selected as a chunk boundary if its fingerprint satisfies Equation (2.1).

$$f \mod D = r \tag{2.1}$$

If a chunk has no window that satisfies Equation(2.1) while maintaining the size within the limits, then it is truncated at the maximum allowed size. After a chunk boundary is declared, its signature is computed. This signature consists of the output of a cryptographically secure (collision resistant) hash function, such as SHA-1, and serves at comparing chunks and detecting if two chunks are identical, as we will see below. The values of $l_{\min}$, and $l_{\max}$, in combination with D, r, and w control the average chunk size, which in our case is 1KB.

From a practical perspective, the chunk-creation process is run by the Produck Client. When it needs to store a new file, the Client iterates through its contents as follows. First, it initializes a new chunk and inserts the first $l_{\min}$ bytes of the file in it. Then it computes the fingerprint f of the last w bytes it added. If the

fingerprint satisfies Equation (2.1), then it closes the chunk and it iterates the process by initializing a new chunk with the next(non-overlapping) $l_{\min}$ bytes from the file. If, instead, the equation is not satisfied, i.e. $f \mod D \neq r$, the Client continues filling the current chunk by adding l_{stp} bytes at a time, recomputing the fingerprint of its last w bytes, and reevaluating the equation. Here we have to note that the new last w bytes of the chunk may have an overlap with the previous chunk suffix, depending on the relation of the l_{stp} and w. In our case, we set l_{stp} to 1. The Client continues doing so until either the equation is satisfied, or the chunk has reached the maximum prescribed size, $l_{\max}$, at which point it stores the chunk and repeats the process by creating a new one. For each one of the created chunks, its signature is computes and stored.

Using this content-based process, the Client splits a file in chunks in a way that allows it to absorb changes in their content, without rendering duplicate detection impossible, as explained in Section 2.2 and shown in Figure 2.3.

2.3.2.2 Super-chunk formation

After splitting the file in chunks and computing their fingerprints, the Client is also responsible for grouping chunks into superchunks, which constitute the basic routing/storage unit, as in [DDL+11]. Similar to chunks, superchunks are constructed using a content-based approach. However, instead of checking the fingerprint of a window of w bytes, as in the case of chunk formation, the Client identifies the boundary of a superchunk at the level of chunks. To this end, a minimum and a maximum superchunk size is given as system parameters and to determine if a chunk is a superchunk boundary, its fingerprint is checked against a criterion similar to Equation (2.1). Specifically, when the Client detects the end of a new chunk, c, as part of the chunking process described above, it first adds the newly created chunk c to the current superchunk. Then it computes c's fingerprint, f, by hashing c's content. If the number of chunks in the current superchunk is within the minimum and maximum superchunk sizes and f satisfies Equation (2.1), the Client sets the end to the current superchunk and starts a new one, otherwise it continues adding chunks to the current one, until the maximum allowed size is reached or a content-based boundary is found. In our case, we use an average superchunk size of 15MB or $15 * 1024$ chunks. We examine the impact of varying the superchunk size in Section 2.6.2.

2.3.3 Chunk Assignment

The key operation in a cluster-based storage system is the assignment of data to StorageNodes so as to maximize deduplication. In Produck, data are assigned to StorageNodes at the granularity of superchunks, as superchunk is out basic

routing unit. This assignment is carried out by the Coordinator through a novel chunk-assignment protocol that employs a stateful approach to aggregate, on the same node, superchunks that share a large number of chunks.

Our protocol relies on the following key observation to reduce the computational and memory requirements of stateful deduplication. To choose the StorageNode offering the highest deduplication rate for a given superchunk, the Coordinator does not need to know exactly which chunks are stored on each StorageNode, but only the size of the overlap between the chunks on each StorageNode and those in the superchunk to be stored. Our novel protocol computes this overlap by relying on PCSA [FM85, DF03], a probabilistic method for computing the cardinality of a multiset, i.e. the number of distinct items (chunks in our case) it contains. To the best of our knowledge, Produck is the first application of PCSA in the context of backup systems.

In the following, we present the basic principles of PCSA in Section 2.3.3.1 and present its application to set intersection in Section 2.3.3.2. Finally we describe how we exploit it for chunk assignment in the context of Produck.

2.3.3.1 Probabilistic counting

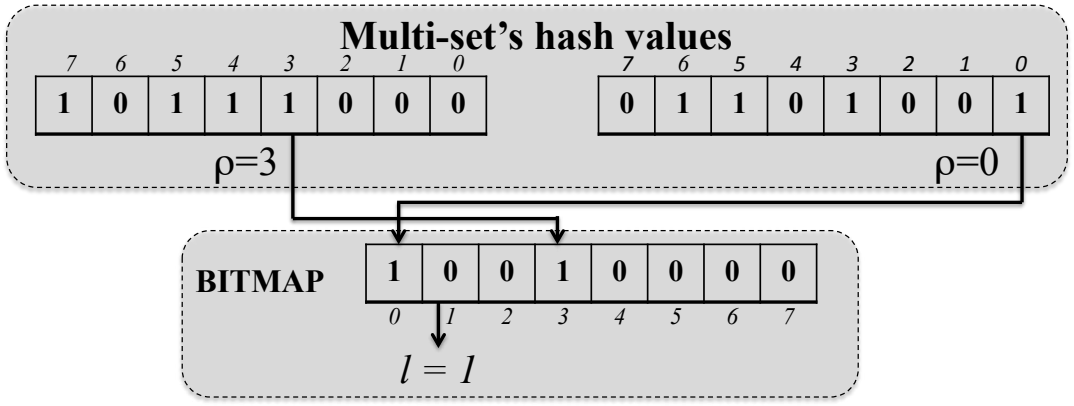


Figure 2.6: PCSA multi-set cardinality estimation.

Let S be a multiset of chunks, for example a superchunk or a set of superchunks stored by a StorageNode. Let $h : S \to [0, 2^L)$ be a hash function that outputs values that are uniformly spread over its target set. Also, let $bit(y, k)$ denote the k-th bit in y's binary representation, with bit 0 being the least significant. PCSA defines a function $\rho : [0, 2^L) \to [0, L)$ identifying the position of the least significant 1-bit in y, with position 0 corresponding to the least significant bit.

$$\rho(y) = \begin{cases} \min\limits_{k \geq 0} bit(y,k) \neq 0 & \text{for } y > 0 \\ L & \text{for } y = 0 \end{cases} \tag{2.2}$$

Given a multiset S, PCSA associates it with a bitmap, a vector of L bits initialized as 0's. Then, it iterates through all the elements in the multiset, $d \in S$ and computes $p = \rho(h(d))$, i.e. the position of the least significant 1-bit in the hash value of d, and records this position by setting the corresponding bit in the bitmap, bitmap[p], to 1. Since the hash function distributes its values uniformly in $[0, 2^L)$, bitmap[0] will be set to 1 approximately half of the times, bitmap[1] will be set 1/4 of the times, bitmap[2] will be set 1/8 of the times, and so on. This leads to the probability of setting the bit at position k shown in Equation (1).

$$P(p(h(d)) = k) = 2^{-k-1} \tag{2.3}$$

Because of this, PCSA uses the position, l, of the leftmost zero in the bitmap vector counting from the left as an estimation of $\log_2(\phi|S|)$, where $\phi = 0.77351$ [FM85]. This makes it possible to estimate the cardinality of S as $2^l/\phi$. Figure 2.6 exemplifies the process. The least significant 1-bit for the left item is at position 3 and causes bitmap[3] to be set. The one for the right item is instead at position 0 and it causes bitmap[0] to be set. Given these values, the position of the leftmost zero in the bitmap vector is then $l = 1$, which leads to a cardinality estimation of $2.58 = 2^1/0.77351$.

Clearly the probabilistic nature of PCSA implies the presence of an estimation error. To reduce it, the authors of [FM85] propose the use of m bitmap vectors, instead of a single one. Specifically, let $r = (h(d) \mod m)$; the improvement consists in using the value of $h(d)/m$ to update the rth vector as described above. Averaging the positions of the leftmost 0s (counting from the left) from all the m bitmap vectors provides an estimate of $\log_2(\phi|S|)$ with less than a 2% error when using $m = 8192$. We therefore use $m = 8192$ in our implementation. Even with this improvement, it is important to observe that the estimation error tends to be larger for very small multisets. This fact must be taken into account when selecting the size of Produck's superchunks, as discussed in Section 2.6.2.

2.3.3.2 Probabilistic multiset intersection

Although PCSA was designed to estimate the cardinality of a multiset, it is easy to see that the bitmap vector of the union of two multisets, A and B, can be computed by simply applying the bitwise OR operator on the two corresponding bitmap vectors, bitmap[A] and bitmap[B].

$$\text{bitmap}[A \cup B] = \text{bitmap}[A] \mid \text{bitmap}[B] \tag{2.4}$$

The above algorithm gives us an estimation of the cardinality of the union of the two sets. Leveraging this observation, the authors of [MBN+06] propose the evaluation of the cardinality of the intersection of two multisets by expressing it as follows.

$$|A \cap B| = |A| + |B| - |A \cup B| \tag{2.5}$$

Equation(3) makes it possible to approximate $|A \cap B|$ simply by estimating the three cardinalities on its the right-hand side, which can be estimated with PCSA and Equation(2).

2.3.3.3 Maintaining chunk information

In the context of Produck, we have two types of multisets. The first are the superchunks created by Clients as described in Section 2.3.2. These are multisets because each superchunk can contain repeated chunks of data. The second consists instead of the sets of chunks stored by each StorageNode. Again, these are multisets because each StorageNode stores multiple superchunks, which in turn may contain multiple identical chunks (although stored once). In fact, this is the goal of the data assignment process – to assign identical chunks to the same node.

The Client is responsible for creating bitmap vectors for the superchunks it creates. Specifically, when a Client wishes to store a file on the backup system, it first splits it into chunks and superchunks as described in Section 2.3.2. For each created superchunk, it computes an array of 8192 bitmap vectors, as described above, and sends it to the Coordinator as a request to store the corresponding superchunk. The total size of these vectors is 64KB. The value of 8192 for the number of vectors was chosen as it provides a good tradeoff between the estimation error and the time complexity of PCSA, as we will show in Section 2.6. In addition, this allows the Coordinator to efficiently assign superchunks to nodes by only storing 64KB per storage node, thus leading to minimal space requirements.

The Coordinator instead is responsible for creating, and maintaining bitmap vectors for the second type of multisets, as a result of its assignment decisions. Specifically, the Coordinator maintains a storage bitmap vector, bitmap^S, for each storage node in the system. Each such vector is initialized to a sequence of 0's and is updated by combining it with the bitmap vectors of stored superchunks according to Equation (2). After each assignment decision, the Coordinator updates the bitmap^S of the storage node that is to store the new superchunk by applying the bitwise OR operator on the node's bitmap^S and the superchunk's bitmap vectors.

In the case of data deletion, the bitmap^S vectors describing node content have also to be updated to reflect the new state of the node. Our method for computing the bitmap vector of the union of a set does not allow us to undo it on-the-fly, thus we rely on offline mechanisms to update the vectors of the nodes and reclaim the freed space. We expect this process to run twice per day as this is also the case for production systems, according to [GE11]. In our experiments, recomputing the bitmap vectors of all the superchunks in the Wikipedia dataset (520GB) takes less than 7 minutes, thus making it fast enough to avoid imposing performance problems.

2.3.3.4 Choosing the best StorageNode

When a Client wishes to store a superchunk, *sc*, on the backup system, it computes its corresponding bitmap vector and sends it to the Coordinator, along with the associated superchunk identifier. The Coordinator then uses the above technique to identify the StorageNode that is most suited to storing the received superchunk. Once it has selected the StorageNode that will store superchunk *sc*, the Coordinator informs the Client and updates the StorageNode's bitmap^S vector, by combining it with the one corresponding to *sc* and using Equation (2).

In order to select the best StorageNode for each superchunk, *sc*, the Coordinator estimates, for each StorageNode n, the cardinality of the intersection between the chunks stored by n and the content of *sc*. This is easily achieved by applying Equations (2) and (3) to the corresponding bitmap and bitmap^S vectors and yields a measure of the overlap between the received superchunk and each StorageNode.

In addition to the overlap between a node's content and a superchunk, and to minimize the effect of estimation errors, we leverage data locality properties observed in backup workloads. Specifically, instead of selecting the StorageNode with the highest overlap value, the Coordinator ranks the top k StorageNodes according to their overlaps. If the node that stored the previous superchunk in the file is among these k and his difference from the first node is less than 10%, then the Coordinator selects this node for the current superchunk, otherwise it selects the top node. We tested several values of k, and $k = 3$ provided a good compromise between locality and overlap. Finally, if no StorageNode is found to have an overlap with a given superchunk or all nodes with an overlap are overloaded, the Coordinator selects a StorageNode at random among the non-overloaded ones.

2.3.4 Load Balancing

As mentioned in Section 2.1, the main challenge when designing cluster-based deduplication systems is the reconciliation of the conflicting requirements for good deduplication while keeping the system load-balanced. In the previous paragraphs, we described how Produck manages to detect the node which shares the most chunks with a new-coming superchunk. In this paragraph we focus on the second goal of the chunk assignment process, which is to keep the system load-balanced.

Produck achieves load balancing through a novel bucket-based mechanism that is specifically designed to operate with our similarity-based chunk-assignment strategy. A simple approach to load balancing could, for example, constrain the selection of the most suitable StorageNode to the nodes that have a storage load that does not exceed the average load of the system by more than a fixed percentage, e.g. 5%. This mechanism, which is used for example in [DDL+11], however, is too aggressive when used with our chunk assignment strategy, as shown in Section 2.6.3.

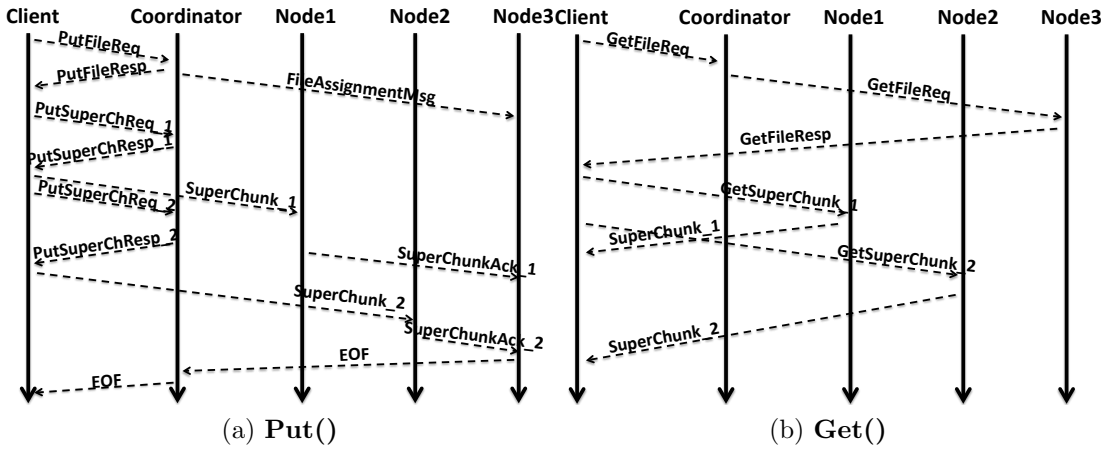
Our novel bucket-based load-balancing strategy, instead, operates by splitting the storage space of each node into fixed-size buckets. At initialization time, the Coordinator grants each StorageNode permission to use one of its buckets. When a StorageNode fills the latest bucket it was granted, the Coordinator grants it a new bucket only if after doing so, the node's used storage space would not exceed the number of buckets allocated to the least loaded node by more than a given threshold. The impact of the maximum number of buckets that the most loaded node is allowed to differ from the least loaded one is further studied in Section 2.6. In the same section, we show that this strategy gives better results in terms of deduplication while guaranteeing an equal distribution of storage load among the nodes in the system.

2.4 PRODUCK Operation

To provide a clearer picture of the behavior of Produck, we now detail the operations carried out when storing and when retrieving a file.

2.4.1 Backing Up a File

The storage of a file starts with a request that a user or an application issues to the Produck Client. The Client first splits the file into chunks and superchunks as described above. Then, it starts the actual backup operation by interacting with the Coordinator and StorageNodes. The details of the process are depicted in Figure 2.7a.



(a) **Put()** (b) **Get()**

Figure 2.7: Messages exchanged during storing and retrieving a file in PRODUCK.

When the Client has created the superchunks, it sends a PutFileReq message to the Coordinator. The PutFileReq contains the file's identifier (fileId), the number of superchunks in the file, their checksums, and the size of the file in bytes. The fileId is the SHA1 hash of the file content and is used in the retrieval process as a file-integrity check. Upon reception of the PutFileReq, the Coordinator chooses a StorageNode uniformly at random and delegates the responsibility for the file to it. This guarantees that all nodes will be responsible for approximately the same number of files. After selecting a responsible StorageNode, the Coordinator replies to the Client with the identifier of this node. In addition, it informs the responsible node about its selection and sends it all the information received by the Client in a FileAssignmentMsg.

The responsible node keeps track of which StorageNode stores each of the superchunks in the file. This allows the Coordinator to maintain state only on a per-file, or even on a per-backup instead of on a per-superchunk granularity. Furthermore, it contributes to responsiveness and scalability of the system, as the small size of this information allows the Coordinator to keep most or even all of the responsible node index in memory.

After the PutFileReq, the Client sends a PutSuperChReq request to the Coordinator for each of the superchunks in the file. This message contains the identifier of the superchunk and its bitmap. This is sufficient for the Coordinator to pick the best candidate among the StorageNodes as described in Section 2.3.3. After picking the right StorageNode, the Coordinator replies to the Client with a PutSuperChResp informing it of which node will store the superchunk. The client reacts by sending the actual data to the selected StorageNode in a SuperChunk message. When the transmission of the superchunk is over, the StorageNode that received it sends a SuperChunkAck message to the node responsible for the file.

The SuperChunkAck contains the checksum of the data in the superchunk and the corresponding identifier. This enables the responsible node to record which StorageNode stored which superchunk, and allows it to check the integrity of the data received by the StorageNode. If something went wrong, the responsible node requests the Client to resend the data to the StorageNode.

The above process is repeated for all the superchunks in the file. When the responsible node has received acknowledgements for all the superchunks, it sends an EOF message to the Coordinator, which then forwards it to the Client.

Here we have to note that, although, in the current version of Produck the operations of chunking and storing a file are executed the one after the other, the two operations could be pipelined and the Client could send a storage request as soon as a superchunk is ready to be stored.

2.4.2 Recovering a File

The process of reading a file is also initiated by a user request to the client. As depicted in Figure 2.7b, the client reacts to this request by contacting the Coordinator with a GetFileReq message, specifying the identifier of the file to retrieve. The Coordinator looks up the responsible node for the file and forwards the GetFileReq message to it. The responsible node replies to the client by providing the identifier, checksum, and StorageNode for each superchunk in the file in a GetFileResp message. The client downloads each superchunk from the corresponding StorageNode and uses the checksums received from the responsible node to verify their integrity.

2.5 Experimental Methodology

In this section we present our experimental set-up: the datasets we used, the competitors we compared Produck against, and the metrics we used. Since we focus on evaluating deduplication efficiency and load balancing with respect to storage, we evaluate Produck through simulations to avoid any networking side-effects. Yet, we built our simulator so that the transition to a real implementation only consists in changing its communication primitives from in-memory transactions to network messages.

2.5.1 Datasets

We evaluate Produck and its competitors using two real-world workloads. The first is publicly available for download[1] and consists of 16 full snapshots of the

[1] http://dumps.wikimedia.org/enwiki/

English version of Wikipedia. The oldest dates back to March 2011, while the newest was taken in May 2012. This dataset contains full versions of all articles in Wikipedia and accounts for more than 520GB of data. The second dataset contains images of the environments available for deployment on the servers of the Grid5000 experimental platform[2] and accounts for 142GB. These workloads contain both text(Wikipedia) and binary data(Images), thus covering many common cases. Table 2.1 presents more details concerning the above described datasets. In particular, the Deduplication Factor is the ratio of the original size of each dataset divided by its size after being deduplicated based on our chunking mechanism.

Dataset	**Size (GB)**	**Deduplication Factor**	**Data Format**
Wikipedia	522	1.96	HTML
Images	142	4.27	OS images

Table 2.1: Dataset Description

2.5.2 Competitors

Before evaluating the specific aspects of Produck, we compare it against the two state-of-the-art cluster-based deduplication storage systems presented in [DDL+11] and described in Section 2.2 of this document. These are the BloomFilter stateful strategy and the MinHash stateless one. To remind the reader of the basic principles of each one of these two strategies, we recall that in BloomFilter the Coordinator keeps a bloom filter as the chunk index of each StorageNode in order to assign superchunks to nodes, while in MinHash, the minimum hash value, i.e. the minimum signature, of the chunks in a given superchunk is selected as the superchunk's signature and the $modulo(number\ of\ bins)$ operator is applied on it to select the bin to store it. Bins are an abstraction used for load balancing and are assigned to nodes initially at random, and when a node becomes overloaded, they are reshuffled and reassigned to nodes. In both strategies, a node is considered overloaded if its load exceeds the average load in the system by 5%.

In all our experiments, we set the superchunk size for the two strategies mentioned above to 1MB. This value is quoted in [DDL+11] as the one giving the best results. We confirmed this by running MinHash on our datasets. In addition, we set the number of bins for MinHash to 100.

[2]https://www.grid5000.fr/

2.5.3 Evaluation Metrics

We evaluate Produck along the same metrics as the ones used in [DDL+11], namely Total Deduplication (TD) representing deduplication efficiency, Data Skew (DS) capturing load imbalance and Effective Deduplication (ED) accounting for both the deduplication factor and load balance. In addition, we also measure the Assignment Time (AT) to capture the computational cost incurred by each of the assignment strategies: Produck, BloomFilter and MinHash. We now detail each of these metrics.

- Total Deduplication (TD): TD is computed as the ratio between the original size of the dataset and its size after being deduplicated, as shown in Equaltion 2.6. The result is then normalized by the total deduplication achieved on a single-node cluster. This metric measures how efficiently a system is at detecting and, thus, eliminating redundancy (i.e. duplicate chunks) present in the workload. The normalization step makes it easier to compare a cluster-based system with the performance of the optimal case of a single-node system with no load-balancing constraints.

$$Total\ Deduplication\ (TD) = \frac{Original\ Size}{Deduplicated\ Size} \tag{2.6}$$

- Data Skew (DS): DS is the ratio of the occupied storage space on the most loaded node to the average load in the system. This is shown in Equation 2.7. DS captures the efficiency of a system in spreading the load equally among the available nodes. A high value of DS means that a node is overloaded and can result in duplicate data being sent to sub-optimal (in terms of deduplication) nodes. To understand this, one has to remember that the Coordinator will not consider an overloaded node eligible for storing a new superchunk, even though it may already store most of the superchunk's contents. Instead of the overloaded node, the Coordinator will select another node, resulting in sub-optimal deduplication.

$$Data\ Skew\ (DS) = \frac{Max\ Node\ Load}{Avg.\ Node\ Load} \tag{2.7}$$

- Effective Deduplication (ED): ED is defined as Total Deduplication (TD) divided by Data Skew (DS) and normalized by the TD achieved by a single

node system. This metric encompasses both deduplication effectiveness and storage (im)balance. The intuition behind it, is that the performance of the whole cluster degrades if one node is overloaded. The reason for the normalization is the same as in the case of Total Deduplication. Equation 2.8 presents the formula to compute the metric.

$$Effective\ Deduplication\ (ED) = \frac{Total\ Deduplication\ (TD)}{Data\ Skew\ (DS)} \tag{2.8}$$

- Assignment Time (AT): Finally, the Assignment time, measured in seconds, measures the time it takes for each strategy to assign all the dataset to the available nodes. This metric provides an estimation of the computational cost of each strategy and can be viewed as an upper bound on the system's throughput as data cannot be stored before being assigned to a node.

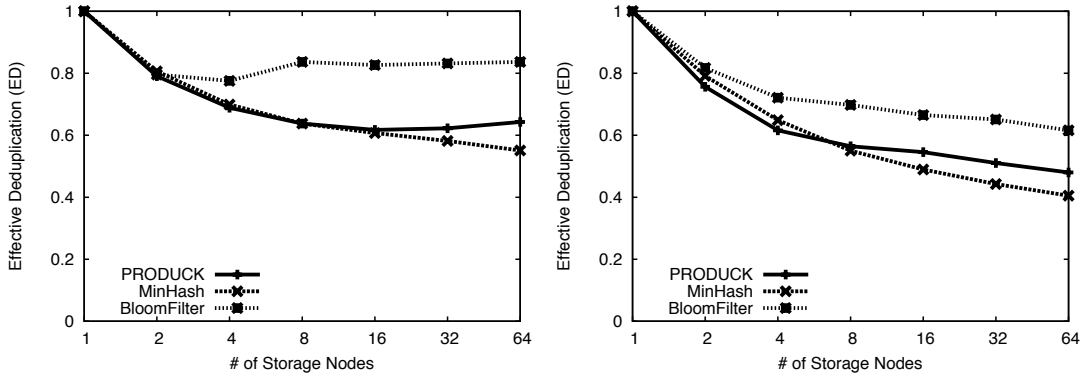
$$Assignment\ Time\ (AT) = seconds\ needed\ to\ assign\ all\ data\ to\ nodes \tag{2.9}$$

2.6 Experimental Results

In this section, we provide a thorough evaluation of Produck. In Section 2.6.1 we present the results of the comparison of Produck against the two alternative cluster-based deduplication strategies presented in Section 2.5.2. We then dive into the specifics of Produck in Section 2.6.2 and study how different values of its configuration parameters affect its performance. Finally, Section 2.6.3 shows the contribution of our bucket-based load-balancing mechanism to the overall performance of Produck and presents how load balancing is preserved as more data is added to the system.

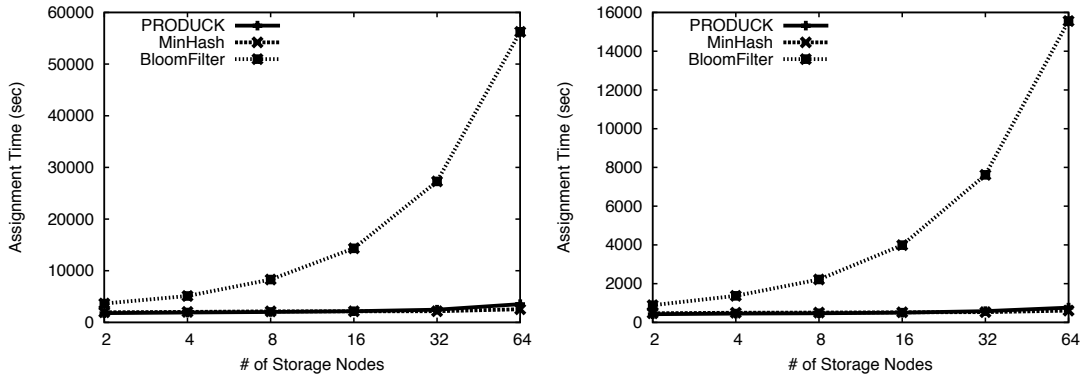
2.6.1 Produck against Competitors

We first compare Produck against the BloomFilter (stateful) and the MinHash (stateless) approaches across different cluster sizes. For all approaches, we use a chunk size of 1KB. However, for the sake of fairness, we set the other parameters to values that have been identified as optimal for each approach. The optimal superchunk size for MinHash and BloomFilter was identified by their



(a) *Wikipedia* (b) *Images*

Figure 2.8: Effective Deduplication (ED) for all datasets for PRODUCK, BLOOMFILTER and MINHASH as a function of the cluster size.

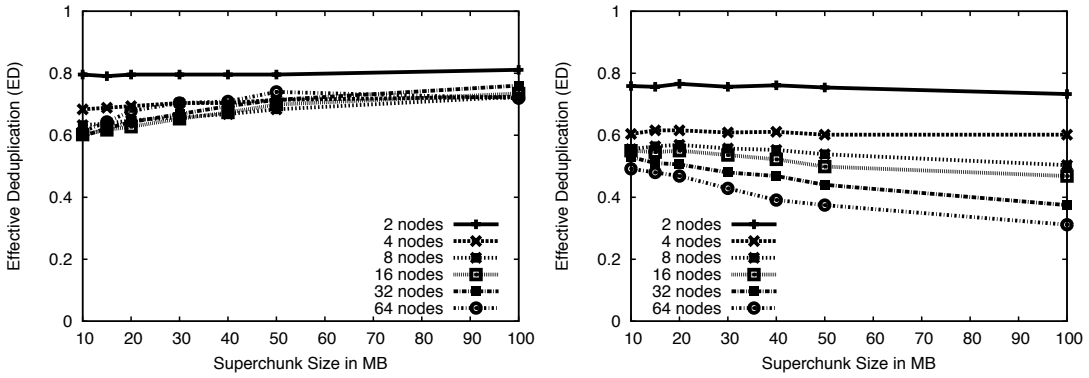


(a) *Wikipedia* (b) *Images*

Figure 2.9: Assignement Time (AT) in seconds for all datasets for PRODUCK, BLOOMFILTER and MINHASH as a function of the cluster size.

authors [DDL+11] as 1MB. This is confirmed by our own experiments. With 64 nodes, MinHash yields ED values of 0.49 and 0.26 for Wikipedia, and 0.27 and 0.13 for Images for superchunk sizes of 10MB and 100MB respectively, against values of 0.55 and 0.40 with a superchunk size of 1MB. For Produck, we set the superchunk size to 15MB, the size of each bucket used for load balancing to 5 superchunks and the maximum allowed difference between the nodes to 1 bukcet. The reasoning behind these default values along with a sensitivity analysis of the impact of different superchunk sizes on Produck's performance are presented in Section 2.6.2.

We are firstly interested in studying how each strategy manages the tension between the level of deduplication it achieves and the requirement for a load-



(a) *Wikipedia* (b) *Images*

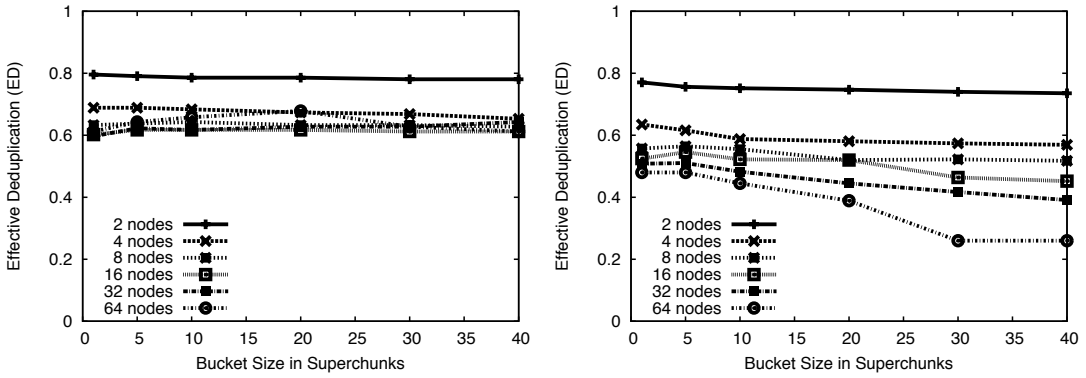
Figure 2.10: Effective Deduplication (ED) for both datasets for different superchunk and cluster sizes.

balanced system. For this purpose, we choose to present in Figure 2.8 the Effective Deduplication (ED) of each system, as this metric combines deduplication effectiveness and load balancing. We observe from Figure 2.8 that Produck outperforms MinHash across both datasets for big clusters and that the difference increases as the cluster-size grows. In fact, focusing on 32- and 64-node clusters, which constitute the most difficult cases, as the more the nodes the more pronounced the deduplication/load balancing tension, we can see that Produck improves ED by 7% and 16% for the Wikipedia dataset and 16% and 21% for the Images one when compared to MinHash. In addition, for the Wikipedia dataset, which has the largest size after deduplication, we see that Produck scales better than MinHash as the cluster grows while for Images the two curves present similar behavior.

Although Produck is outperformed by the BloomFilter strategy, its ED is at most 23% and 22% lower, respectively in the Wikpedia and Images datasets. The better ED of BloomFilter is expected since this strategy sacrifices memory and computational resources to achieve almost accurate knowledge of a node's content at all times. This allows BloomFilter to leverage deduplication while ensuring that the load remains balanced. Yet, this also causes its cost to be significantly higher than that of Produck as shown in Figure 2.9. The plot presents the Assignment Time (AT) metric, i.e. the time needed by each strategy to assign all the data in a given dataset to the available storage nodes. This metric, apart from being an illustration of computational cost, also serves as an upper bound on the throughput of each storage system. Figure 2.9 clearly indicates that the computational cost of BloomFilter is much higher than those of both Produck and MinHash. In addition, and more importantly, the computational

nodes	Produck				MinHash				BloomFilter					
	ED	**DS**	**TD**	**AT**	**ED**	**DS**	**TD**	**AT**	**ED**	**DS**	**TD**	**AT**	**ED (1/8)**	**AT (1/8)**
Wikipedia														
1	1.00	1.00	1.00	-	1.00	1.00	1.00	-	1.00	1.00	1.00	-	1.00	-
2	0.79	1.00	0.79	1795	0.80	1.00	0.80	1959	0.80	1.00	0.80	3655	0.79	2177
4	0.69	1.00	0.69	1916	0.70	1.00	0.70	2035	0.78	1.00	0.79	5108	0.66	2484
8	0.64	1.00	0.64	2015	0.63	1.01	0.64	2135	0.84	1.01	0.86	8294	0.59	2962
16	0.62	1.00	0.62	2170	0.60	1.01	0.61	2146	0.83	1.02	0.87	14344	0.56	3819
32	0.62	1.00	0.62	2466	0.58	1.03	0.59	2171	0.83	1.05	0.87	27292	0.64	5304
64	0.64	1.01	0.65	3541	0.55	1.05	0.58	2548	0.84	1.05	0.88	56237	0.79	9302
Images														
1	1.00	1.00	1.00	-	1.00	1.00	1.00	-	1.00	1.00	1.00	-	1.00	-
2	0.76	1.00	0.76	426	0.79	1.00	0.79	472	0.82	1.00	0.82	889	0.74	529
4	0.62	1.00	0.62	457	0.65	1.00	0.65	500	0.72	1.05	0.76	1370	0.56	642
8	0.56	1.01	0.57	470	0.55	1.02	0.56	506	0.70	1.05	0.73	2216	0.44	799
16	0.55	1.01	0.55	503	0.49	1.03	0.50	521	0.67	1.05	0.70	3989	0.47	1049
32	0.51	1.02	0.52	582	0.44	1.05	0.46	531	0.65	1.05	0.68	7616	0.54	1530
64	0.48	1.01	0.48	762	0.40	1.08	0.44	610	0.62	1.05	0.65	15555	0.57	2763

Table 2.2: Effective Deduplication (ED), Data Skiew (DS) , Total Deduplication (TD) and Assignment Time (AT) as functions of the cluster size. The last two columns of the table are for the case where sampling (1 over 8) is applied to the superchunks to reduce the number of comparisons and the size of the bloom filter.



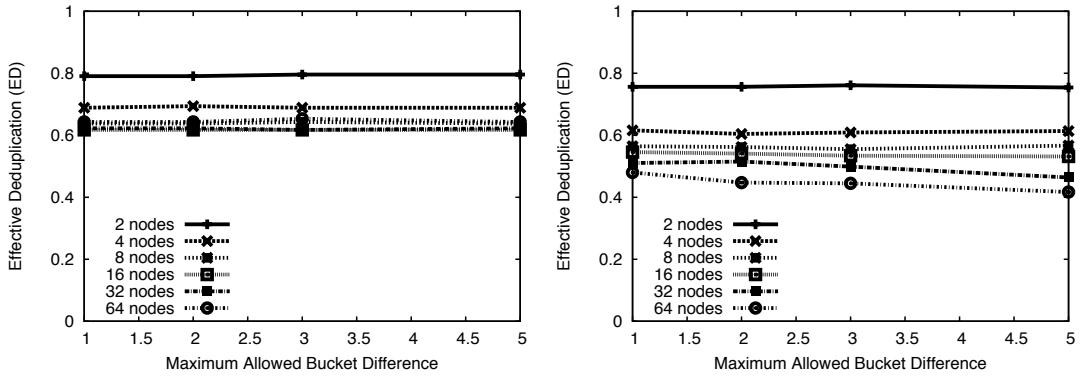
(a) *Wikipedia* (b) *Images*

Figure 2.11: Effective Deduplication (ED) for both datasets for different *bucket* and cluster sizes.

cost of BloomFilter increases too fast as the cluster grows, something that can easily result in scalability problems. For a 32-node cluster, BloomFilter is more than 11 times slower than Produck for Wikipedia and more than 13 times for Images while for a 64-node one, these values become 16 and 21 respectively. More importantly, the memory overhead induced by the maintenance of the Bloom filters remains far from negligible. To achieve a 1% rate of false positives, a Bloom filter requires 9.6 bits per element. Given a chunk size of 1KB and for a storage node with a capacity of 140TB of unique data, as the ones mentioned in the commercial version of [DDL+11], the Bloom filter can reach the size of 168GB. [3] And this is just for 1 storage node. The amount of memory required by the Coordinator should be multiplied by the number of storage nodes in the system. These drawbacks were also observed by the authors of [DDL+11] who addressed this issue by sampling the chunks in a superchunk at a frequency of 1 over 8. We applied the same strategy and present the corresponding results for ED and AT in the last two columns of Table 2.2, along with those for all the other metrics and strategies.

Table 2.2 shows the results of all the metrics, including Data Skew (DS) and Total Deduplication (TD) for all the considered approaches. From the table, it is clear that Produck consistently improves TD with respect to the MinHash approach for large clusters, while ensuring that the load remains equally spread among nodes. In fact, DS does not surpass 1.02, which corresponds to the most loaded node being only 2% more loaded than the average. More importantly, the fact that BloomFilter provides a better deduplication factor comes at the price

[3] This is due to the fact that 140GB can store up to $\frac{140 \cdot 2^{10} \cdot 2^{10} \cdot 2^{10}}{1} \approx 150 \cdot 10^9$ unique chunks which should be multiplied by the $9,6$bits to find the size of the BloomFilter.



(a) *Wikipedia* (b) *Images*

Figure 2.12: Effective Deduplication (ED) for both datasets for different values of maximum allowed *bucket* difference between the most and the least loaded nodes and across different cluster sizes.

of high computational and memory cost.

In addition, in Table 2.2 we present the results of BloomFilter, after applying the sampling strategy proposed in [DDL+11]. After the sampling, we see that its ED drops significantly in both datasets and across most cluster sizes, becoming lower than that of Produck in most of the cases. At the same time, its Assignment Time remains 3 to 4 times larger than that of Produck. In addition, although sampling reduces the sizes of the Bloom filters by 8 times, the total amount of memory required for storing the Bloom filters on the Coordinator remains large: 64×21GB for a system of 64 storage nodes of 140TB each.

2.6.2 Produck Sensitivity Analysis

In this section, we study the impact of various configuration parameters on the performance of Produck. This contributes to (i) illustrating the tradeoffs in cluster-based deduplication, and (ii) revealing the reasoning behind the default values we selected for our system. The parameters we are interested in are (i) the size of the superchunk, (ii) the size of the buckets used for load balancing, and (iii) the maximum allowed difference in the number of used storage buckets between the most and the least loaded node.

The results of this analysis are presented in Figures 2.10, 2.11, and 2.12 across different cluster sizes. This provides an idea of the evolution of the system as the cluster grows. In the plots, the size of the superchunk is measured in groups of 1024 consecutive chunks, which correspond to MBs as the average chunk size is of 1KB. The default superchunk size is 15MB. The bucket size is instead expressed in number of superchunks, with a default value of 5. Finally, the maximum allowed

bucket difference is expressed in number of buckets, and has a default value of 1. In each experiment, we vary one of the above three parameters while keeping the other two constant at their respective default values.

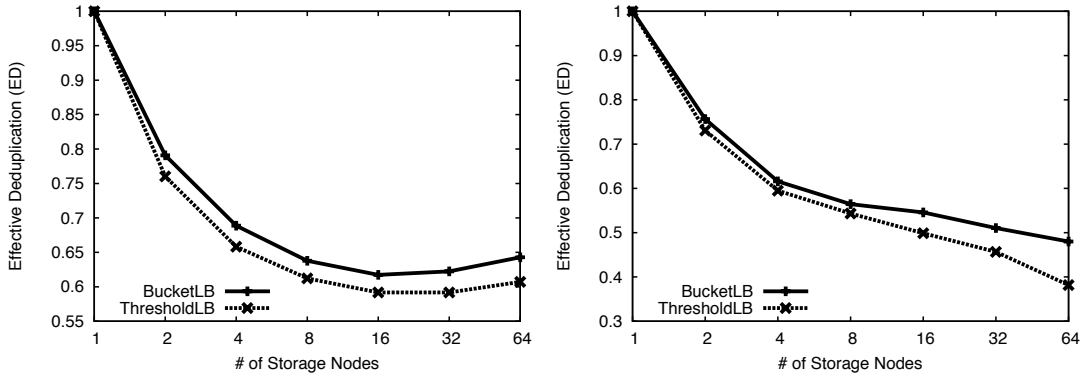
2.6.2.1 Superchunk size

Figure 2.10 shows how the values of ED are affected by the size of superchunks for various cluster sizes. In each of the datasets, performance is best for a specific superchunk size. In the case of Wikipedia the best ED is achieved with a size of around 50MB, while for Images the maximum is around 10MB. The difference between the optimal values of the two datasets reflects their different properties. As a general rule, smaller superchunks are better for workloads characterized by larger deduplication factors. This is because smaller superchunks improve the ability to detect redundancy and thus provide a greater advantage for workloads characterized by the presence of a large number of duplicates. Nonetheless, even in the case of Images, too small superchunk sizes prove to be disadvantageous. The reason lies in the estimation error inherent in PCSA. As observed in Section 2.3.3, the estimation error is larger when the multisets (superchunks) being considered are small. The choice of the best superchunks size therefore requires a balance between these two aspects. In the current version of Produck, we use a default value of 15MB. This value not only provides good results on the considered datasets, but it is also close to the optimal value for Images, which, according to the study in [WDQ+12], is a good example of real workloads. We are investigating the possibility of dynamically determining the best superchunk size for the workload being considered.

Figure 2.10 also shows another interesting fact. In the case of Images, ED tends to drop as the cluster size grows. This negative impact of the cluster size on ED is expected. The bigger the cluster the more the load-balancing mechanism will spread the data among the cluster nodes. Yet, this is at odds with the effort of the deduplication mechanism that wants data to be concentrated on as few nodes as possible so that more duplicates end up on the same server. The figure also shows that the impact of the cluster size is much less significant for the Wikipedia dataset. Again, this is because the larger number of duplicates in Images make it more important to keep data clustered.

2.6.2.2 Bucket size

Figures 2.11 analyze the values of ED for increasing bucket sizes. A first conclusion we can draw from the graphs is that ED either stabilizes or drops for bucket sizes greater than 10 superchunks. In the case of the Wikipedia dataset, where duplicates are scarce, all bucket sizes perform almost the same and, once again, the cluster size does not have a big impact on ED beyond a size of 2 nodes (ED



(a) *Wikipedia ED* (b) *Images ED*

Figure 2.13: Effective Deduplication (ED) for both datasets for two load balancing strategies (i) our *bucket*-based one, here called *BucketLB*, and (ii) the one from [DDL+11], namely *ThresholdLB*, where the storage load of a node is not allowed to deviate by more than 5% from the average load in the system.

varies from 0.69 to 0.61). For the Images dataset, where deduplication has more potential for space reduction, smaller bucket sizes have better performance. More precisely, a bucket size of 5 seems to be the optimal value for most cluster sizes. This may seem counter intuitive as bigger bucket sizes mean fewer interventions of the load-balancing mechanism, and thus more deduplication. However, a closer look at the results of the experiments reveals that, in most cases, although the achieved deduplication drops slightly with increasing bucket sizes, the drop in ED is mostly due to an increase in the load imbalance (DS) among the nodes. This, in turn, leads to more non-optimal node assignments because of the effort to spread the load equally, ultimately causing a small drop in deduplication.

Based on this analysis, we set Produck's default bucket size to 5 superchunks. This keeps the system load balanced even for small datasets like Images on big clusters (for Images on a 64 node cluster, the value of DS is 1.01) while giving good deduplication results. For larger datasets, equal load distribution is also guaranteed as the maximum deviation of a node from the average load in the system is bounded. We further examine the efficiency of Produck's load-balancing strategy in Section 2.6.3.

2.6.2.3 Maximum allowed bucket difference

Finally, Figure 2.12 presents the performance of Produck for different levels of allowed load imbalance among StorageNodes. The plot shows that increasing the maximum allowed load imbalance does not have a significant impact for the Wikipedia dataset. The same also holds for Images, although here, the impact

is a bit more pronounced. This is rather expected as in Images there are more duplicates. For very small cluster sizes, the maximum allowed bucket difference does not significantly affect the achieved ED. When the cluster grows, a maximum allowed difference of 1 bucket gives the best results.

2.6.3 Load Balancing and Evolution with Time

The whole design of Produck was driven by the tension between the requirement for efficient deduplication and the one for a load-balanced system. We now study the impact of our novel load-balancing strategy on the performance of Produck. To this end, we compare our bucket-based mechanism (BucketLB) to the one used by the BloomFilter strategy in [DDL+11], here termed ThresholdLB. In ThresholdLB, the storage load of a node is not allowed to deviate by more than 5% from the average load in the system. We ran Produck using both load balancing strategies for different cluster sizes. Figure 2.13 presents the Effective Deduplication (ED) and Table 2.3 presents the Total Deduplication (TD) and Data Skew (DS) achieved in both datasets. In terms of ED, BucketLB outperforms ThresholdLB across all cluster sizes and datasets. In fact, on a 64-node cluster, BucketLB achieves 4.2% and 10.3% better ED, respectively, on Wikipedia and Images. From Table 2.3, we can see that for small cluster sizes, both strategies achieve the same total deduplication, but ThresholdLB pays a higher price in terms of load imbalance, as its DS is constantly higher than that of BucketLB. As the cluster grows and the intervention of the load-balancing mechanism is more pronounced, we see that BucketLB also enables Produck to achieve better deduplication especially in workloads with more duplicates. Ultimately, this allows Produck to achieve the same or better deduplication while keeping the system more load balanced. In addition, its benefits are more pronounced as the cluster grows. This leads to the conclusion that BucketLB provides Produck with better scalability, a highly desired property as deduplication clusters are expected to grow constantly due to the fast pace at which digital data is produced [GCM+08b].

We also examine how our load-balancing mechanism behaves as more data is added to the system. We focus on the Wikipedia dataset, which is the largest one after deduplication, and we store the snapshots in the order they were taken while recording system statistics after each snapshot is stored. We do so in order to make our results as close to a real deployment as possible. The same conclusions hold for the other dataset. The results for Data Skew in 32- and 64-node clusters are presented in Figure 2.14. We focus on big clusters for two reasons. First, backup clusters are expected to grow constantly, as mentioned earlier. Second, the effect of the load-balancing mechanism on deduplication is more pronounced on large clusters as data is more scattered across nodes. Figure 2.14 shows that our bucket-based load-balancing policy manages to guarantee an equally distributed

nodes	Wikipedia				Images			
	Bucket		Threshold		Bucket		Threshold	
	TD	DS	TD	DS	TD	DS	TD	DS
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	0.79	1.00	0.8	1.05	0.76	1.00	0.77	1.05
4	0.69	1.00	0.69	1.05	0.62	1.00	0.62	1.05
8	0.64	1.00	0.64	1.05	0.56	1.01	0.56	1.05
16	0.62	1.00	0.62	1.05	0.55	1.01	0.52	1.05
32	0.62	1.00	0.62	1.05	0.51	1.02	0.48	1.06
64	0.65	1.01	0.64	1.05	0.48	1.01	0.41	1.06

Table 2.3: Total Deduplication (TD), and Data Skiew (DS) for both datasets and for both load balancing strategies, *BucketLB* and *ThresholdLB*, as a function of the cluster size.

storage load even from the first few GBs stored in the system. In a 64-node cluster, after only the first snapshot has been stored (29GBs), the most loaded node does not store more than 10% more data than the average load in the system (DS=1.1). In addition, the load imbalance drops fast and after 6 snapshots, DS does not surpass the value of 1.03. The drop in the load imbalance is expected as nodes are allowed to deviate by one bucket at most and buckets are of fixed size. As more data is added to the system, and the average load increases, the size of a bucket becomes smaller as a percentage of the increasing average load, thus causing a continuous decrease in DS. The good performance of Produck in terms of load balancing allows it to avoid periodic rebalancing operations as is instead done in MinHash [DDL+11].

Finally, although we considered nodes equipped with disks of equal size, this does not limit the applicability of Produck and its load-balancing strategy. In the case of nodes with different disk sizes, each node can be assigned a weight equal to the ratio of its disk size to the maximum disk size in the system. This allows our load-balancing mechanism to load each node proportionally to its capacity.

2.7 Conclusion

In this work, we presented Produck, a stateful yet lightweight deduplication solution for cluster-based storage systems. We compared our solution to state-of-the-art stateful and stateless techniques over two real-world workloads of close to 700GB in total and we showed that Produck achieves an 18% average improvement in deduplication with respect to existing stateless solutions while reducing the superchunk-assignment time by up to 18 times, when compared to stateful

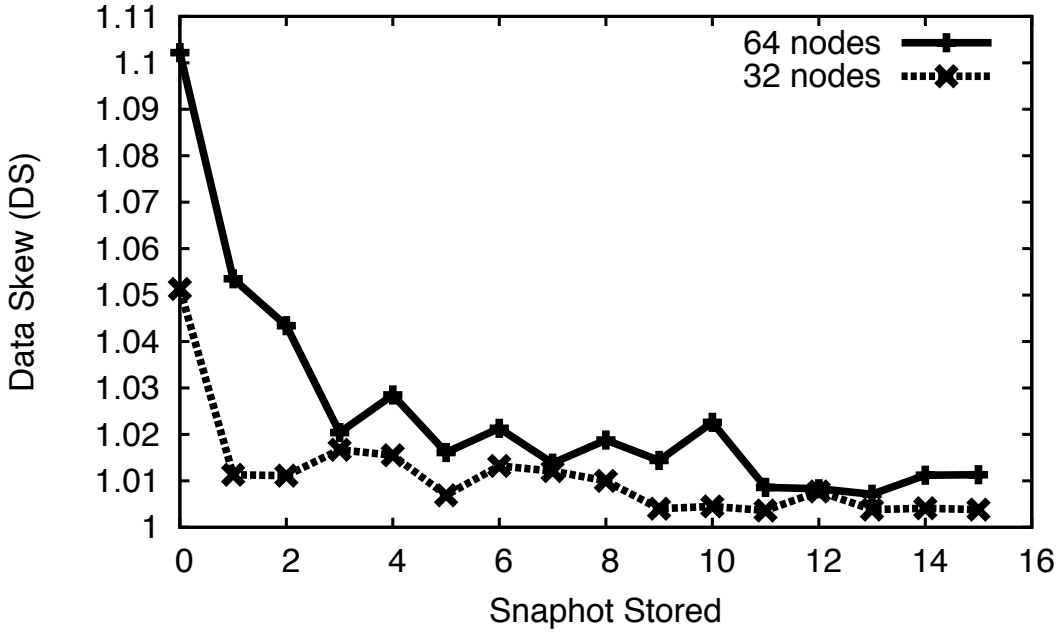


Figure 2.14: Evolution of the Data Skew (DS) for the *Wikipedia* dataset as data is stored at their chronological order.

ones. In addition, this is achieved with minimal computational and memory costs. Specifically, the memory overhead is limited to an indexing structure of 64KB per storage node. Our findings demonstrate that stateful solutions can actually be used in practice.

Apart from the above conclusions, our study has two main contributions. The first is the introduction and evaluation of a compact indexing data structure in the context of data deduplication. This structure allows the Coordinator to accurately detect the overlap between the contents stored by a node and those of a superchunk, with minimal memory requirements ($64KB$) and minimal computational cost. The second contribution is a novel, bucket-based load-balancing mechanism. This mechanism manages to maintain the system load balanced while not sacrificing duplicate detection and without requiring any additional rearrangement of data to nodes, which could result in huge amounts of traffic.

Chapter 3

Content and Geographical Locality in User-Generated Content Sharing Systems

User Generated Content (UGC), such as YouTube videos, accounts for a substantial fraction of the Internet traffic and according to analysts this is bound to increase. To optimize their performance and minimize their operational costs, UGC sharing sites invest a lot of effort in placing content close to their users that request it. To do this, they usually rely on both proactive and reactive approaches that exploit spatial and temporal locality in access patterns. Alternative types of locality are also relevant and hardly ever considered together. In this chapter, we show on a large (more than 650,000 videos) YouTube dataset that content locality (induced by the related videos feature) and geographic locality (i.e. the fact that two videos have most of their views originating from the same countries), are in fact correlated. More specifically, we show how the geographic view distribution of a video can be inferred to a large extent from that of its related videos. We leverage these findings to propose a UGC storage system that proactively places videos close to their expected, future requests.

3.1 Introduction

Over the last few years, users have become the most prolific content generators on the Web, prompting the phenomenal success of user-generated content (UGC) sharing sites such as YouTube [CKR+07, GALM07]. This rapid growth combined with the never fulfilled user demand for better quality, makes serving UGC to an increasing number of users all around the world a daily engineering feat [SSF08]. To this end, UGC sharing sites rely, in most cases, on Content Delivery Networks (CDNs) to place content close to consumers, in order to be able to guarantee good

quality of service while minimizing their operational costs. To achieve this goal, CDNs employ both proactive approaches that rely on a priori knowledge (e.g., it is likely that a French-speaking video will be accessed mostly from French-speaking countries) and reactive ones where aggressive replication is performed whenever a new request arrives for a given video. In addition, in the reactive approaches we could also add the ones where the recent history of a given video is analysed to predict its future requests [HWLR08, CLYL08, YLZ+10].

Interestingly, beyond traditional forms of locality that account for each content item independently, recent studies have shown that UGC viewing patterns are significantly influenced by the fact that content in these sites is no longer independent but is organized in a content graph. In YouTube, this content graph is embodied by the lists of "related videos" present on each video's page, and its influence on the viewing behavior of users has been clearly documented [KZGZ11, ZKG10].

In this chapter we study the geographic viewing patterns of UGC and how they are affected by the content graph. Our analysis on a novel YouTube dataset shows that (i) related videos tend to have correlated geographic viewing patterns, with most of their views coming from the same countries, and (ii) popular videos tend to have their views more uniformly spread across more countries than less popular ones. The latter category accounts for the vast majority of YouTube's content and have their views coming from a small number of countries. Although this category contains less popular videos, in our analysis we show that its view share is far from negligible, thus accounting for an important percentage of the consumed bandwidth.

Building on these insights, we propose DTube, a system that accurately predicts the origins of a video's future views by looking at its position in the content graph and proactively places its replicas close to its expected consumers. Although the impact of content locality, i.e. proximity in the content graph, on a video's views and geographically concentrated viewing patterns have been studied independently [SMMC11, ZKG10, BSW12], to the best of our knowledge, this is the first work that considers both aspects to optimize the placement of UGC.

3.2 Related work

To explain the challenges faced by User-Generated Content (UGC) sharing sites when it comes to serving the content they host, we start by presenting the advances done in the architecture of Content Delivery Networks, whose goal is to serve content from a server close to the origin of the request, before passing to content placement strategies that leverage content characteristics to make efficient use of the available resources.

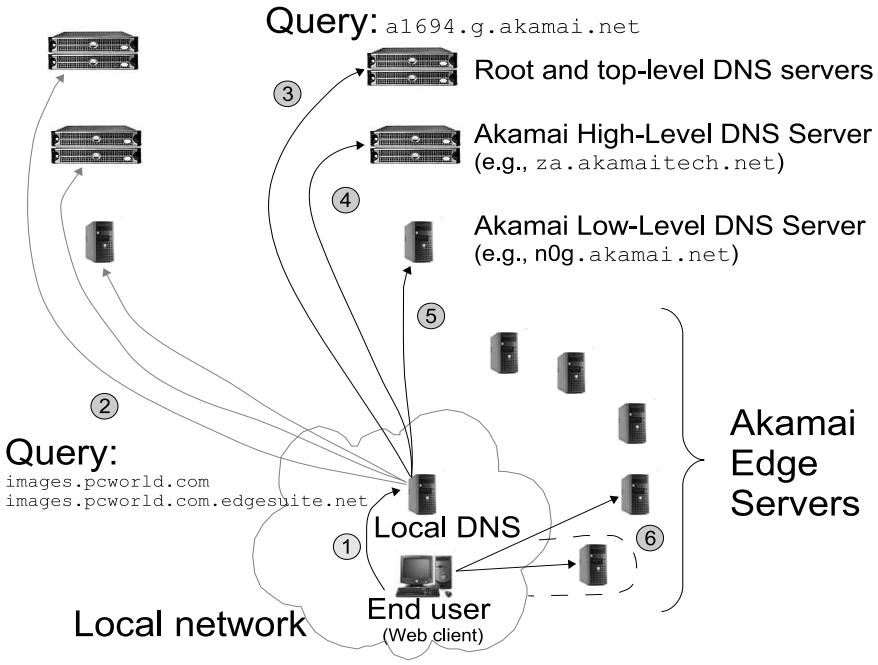


Figure 3.1: Illustration of Akamai DNS translation.

3.2.1 Content Delivery

The traffic directed to sites like YouTube is already huge, as presented in the introduction of this document, and is bound to increase, according to analysts [cis12]. This implies that even if an organization manages to satisfy its current needs, there is no guarantee that it will be able to keep up with the increasing request volume in the future, thus making the danger of "death by success" a constant menace. To face these constantly increasing needs and offload some of the related costs, organizations employ Content Delivery Networks (CDNs) to help them satisfy this increasing demand.

A content delivery network (CDN) is a large distributed system that consists of servers deployed in multiple data centers all around the world. The goal of a CDN is to serve content to end-users with high availability and high performance and alleviate some of the burden that the servers of the CDN's clients have. A CDN operator gets paid by content providers for delivering their content to their audience. In turn, a CDN pays ISPs, carriers, and network operators for hosting its servers in their data centers.

When a site hires a CDN to serve its content (partially or entirely), the network of servers owned by the CDN operator becomes responsible for redirecting incoming requests to its servers that hold the requested content based on load balancing and network path criteria. To do this, a CDN uses a hierarchy of DNS

servers and by performing DNS redirection, it manages to translate a web client's request for content in a CDN customer's domain into the IP address of a nearby CDN server, called edge server. To gain an intuition on how DNS translation works, we present a simple scenario where a client launches a request for a site that happens to be a client of a CDN. First, the end user requests a domain name translation to fetch content from the original site. The customers DNS server uses a canonical name (`CNAME`) for the items that are to be served by the CDN. This `CNAME` serves as an alias that contains a domain name in the network owned by the CDN and enables a DNS server to redirect lookups to the CDN domain [SCKB06]. After this initial translation, it is the responsibility of the hierarchy of DNS servers owned by the CDN to respond to the DNS name-translation request. The goal of this redirection process is to locate the n (in the case of Akamai $n = 2$) edge servers that are not overloaded and are connected to the request issuing client through a good quality network path, so that a good quality of service is guaranteed for the end-user.

Figure 3.1 presents the above process using the `PCWorld.com` site as an example. We assume that the images of the site are served by Akamai. After the client issues a request for an object in the `images.pcworld.com`, the local DNS server (LDNS) is contacted for the IP address of the domain (1). The LDNS then returns the address for the domain (2) and when the `pcworld.com` name server is contacted for a name translation, it begins the DNS redirection by returning a `CNAME` entry for `images.pcworld.com`, that is served by Akamai. In the figure, the value of the `CNAME` entry in this case is `images.pcworld.com.edgesuite.net`; `edgesuite.net` is a domain owned by Akamai. The LDNS performs another a name translation, this time on the `edgesuite.net` domain. Following this, two more DNS redirections are subsequently performed. At first the LDNS is directed to `akamai.net` domain (3), which begins the process of finding a nearby edge server by forwarding the request to a high-level Akamai DNS server (4). After this, the high-level Akamai DNS server forwards the request to a low-level one (5) that, generally, is closer to the LDNS than the high-level one. Then, the low-level Akamai DNS server returns the IP addresses of two edge servers that it expects to offer high performance to the web client (6).

3.2.1.1 Peer-assisted CDNs

Although CDNs perform well, not all organizations can afford hiring them. In addition, CDNs themselves have to face the same scalability challenges, as more and more traffic is redirected to their servers. A natural way to face these challenges is to take advantage of the user resources residing at the end of the network. This can permit services to scale as these resources are close to the users and their number reflects the number of users enjoying the service at any given time. To

do this, researchers and organizations started investigating the potential benefits from a pass from the hierarchical tree-like structure proposed by the original CDN design to the P2P paradigm.

To this end, peer-assisted CDNs have received a great deal of attention in the last years [HWLR08, CLYL08, YLZ+10] with both hybrid and entirely P2P solutions being proposed. In this chapter, we focus on video-on-demand services and in the remaining of this paragraph we present the work related to these services.

In [HWLR08], the authors study the potential benefits of leveraging the user resources on Internet video-on-demand (VoD) and software update distribution. To this end, they envision a hybrid architecture, where CDN owned servers cooperate with user provided resources, i.e. end-user machines, during the content serving process to guarantee better quality of service. The authors initially study the infrastructure of two major CDNs, Akamai and Limelight, and they use trace-driven simulations to evaluate the impact of augmenting their infrastructure with P2P facilities. Their evaluation shows that for a VoD service, even if users contribute to the serving of a video only during the period that they are watching it, can cut by more than 2/3 the bandwidth needs for the CDN and when it comes to software updates, the reduction reaches up to 95%.

In LiveSky [YLZ+10] the authors focus on IPTV and they propose a hybrid architecture to support this type of services. IPTV is a more synchronous service than video-on-demand, as users are watching (almost) the same part of the stream at any given time. This imposes stricter time constraints but guarantees that some peers are available throughout the broadcast. The architecture that the authors propose can be seen as an augmented tree-like structure where the servers owned by the service provider are organized in a tree and the users that are connected to the same edge-server can communicate among them to exchange parts of the stream, thus offloading some of the burden of the edge server. In Livesky, clients switch between CDN and P2P delivery depending on the number of the available peers, the CDN servers' available bandwidth and peer churn. i.e. peers changing their state from offline to online and vice versa.

In [HFC+08] the authors propose an architecture for video-on-demand services that is based entirely on the P2P paradigm. Users participating in the service are contributing a limited volume of storage (1GB) and part of their available upload bandwidth and the system takes care of the putting in contact nodes that request a given film with users that store it. To accomplish this, their system leverages all three types of protocols available in the P2P literature, i.e. trackers (also known as super nodes), DHTs [SMK+01, RD01, ZHS+04, RFH+01] and gossip protocols [EGH+03, EGKM04, GKM03]. These methods provide different levels of availability, freshness and robustness while each one of them requires different amount of metadata to be stored by each participating peer. Trackers

are used to keep track of which peers hold (parts of) a given movie. A peer stores locally a copy of a given video as soon as it watches it. As soon as a peer selects a movie, it contacts the tracker that it is assigned to, in order to locate a node that stores the movie. After this, the new peer becomes member of the swarm of peers watching the requested video and the remaining peers become its neighbors. Discovering which one of the neighbors stores a given part of the film, is done through a gossip protocol. This limits the amount of data the tracker would have to store in the case that it was responsible for saying which peer stores which part. In addition, and from a fault tolerance point of view, this cuts down on the reliance on the tracker, and makes the system more robust. Finally, DHT is implemented by the trackers to assign videos to them and by peers, as well, to provide a non-deterministic paths to the trackers.

In [CLYL08], Chen et al. propose a staged approach for placing and populating servers in hybrid CDN/P2P networks to minimize the operational costs related to streaming media in local P2P communities. In other words, the main problem that the authors try to solve is how to move CDN owned resources from one user swarm to another so that the quality of the stream arriving to the end users is guaranteed to be of good quality throughout the duration of the stream.

In all the above solutions, the main goal is to leverage the stability of the CDN owned servers and the fact that user machines, although unstable, reside close to the users, in order to provide good quality of service. This model was disrupted with the coming of the home gateways as a replacement for the classic modem. Home gateways are devices with computational and storage capabilities, that reside at the edge of the network and are responsible for connecting a collocated set of machines to the Internet. Measurements on the availability of these machines like [VLM+09] have shown that they are almost constantly online, with an availability close to 98%. This change inspired researchers and organizations to study the potentials of using home gateways as edge servers for multiple services, including video-on-demand.

In [VLM+09] the authors propose a distributed computing platform composed of home gateways. In this study, the authors focus on the energy savings incurred by not using servers located in remote (from users) data centers but devices at the edge of the network for providing video-on-demand services. Given this, [HCD09] proposes a decentralized video storage and delivery service that uses home gateways for both storing and serving content. Finally, in [MVCG11], the authors focus on UGC sharing sites and show the feasibility of building a system like YouTube, using residential gateways. The motivation the authors provide for decentralizing such services can be found in the fact that this would permit users to have better control over the content they want to share with the world. Focusing on gateways, they argue that given their high availability and their current processing capabilities, they are good candidates to be used as home servers where

users can upload their content and make it available to their "friends".

Although the above systems leverage advances in technology to optimize serving the content to end-users, none of them has explored what properties of the content itself can be leveraged to further optimize content distribution. In fact, all of them assume that content moves according to user requests (reactively) while none of them investigates how different placement strategies could contribute to the goal of good quality of service while making efficient resource utilization.

3.2.1.2 Content Agnostic Placement

Focusing on the content placement problem on a geographically distributed system, Kangasharju et al. [KRT07] investigate optimal placement strategies in P2P content networks to maximize availability in content communities. To do this, they develop an analytical optimization theory for benchmarking the performance of replication/replacement algorithms and they propose a content management algorithm, the Top-K Most Frequently Requested algorithm. Through their framework, they show that in most cases this algorithm converges to an optimal replica profile. Tan et al. [TM10] investigate the same problem for video-on-demand, but optimize for the total upload bandwidth needed.

The above solutions consider requests as entities by themselves and the channels through which an item can become popular are not taken into account. In fact, most of the above solutions consider that there is a central catalog that the user visits and can select the movie that she wants to watch. In [SMMC11], the authors make the case that nowadays, the diffusion of multimedia content offered by sites like YouTube is more and more done through channels offered by other online social networks such as Twitter and Facebook. Based on this assumption, they investigate the possibility of monitoring these alternative channels to enhance the quality of the service provided by CDNs. To this end, they track social cascades happening on Twitter, i.e. consecutive "retweets" among "followers" of a message, and concerning YouTube videos and they show that they tend to be confined to a geographically limited region. Thus, a CDN could improve its cache hit rates by tracking the propagation of these cascades and caching content accordingly.

3.2.1.3 Content Aware Placement

Although the above strategies take into account past user behavior to predict the possible origins of future video requests, an aspect that they tend to neglect even though it is crucial in user behavior in these sites, is the fact that in social-networks content is no longer independent. Users "follow"(Twitter) or "befriend"(Facebook) certain other users while videos are considered as "related"(YouTube) to other videos. This creates inter-dependencies between con-

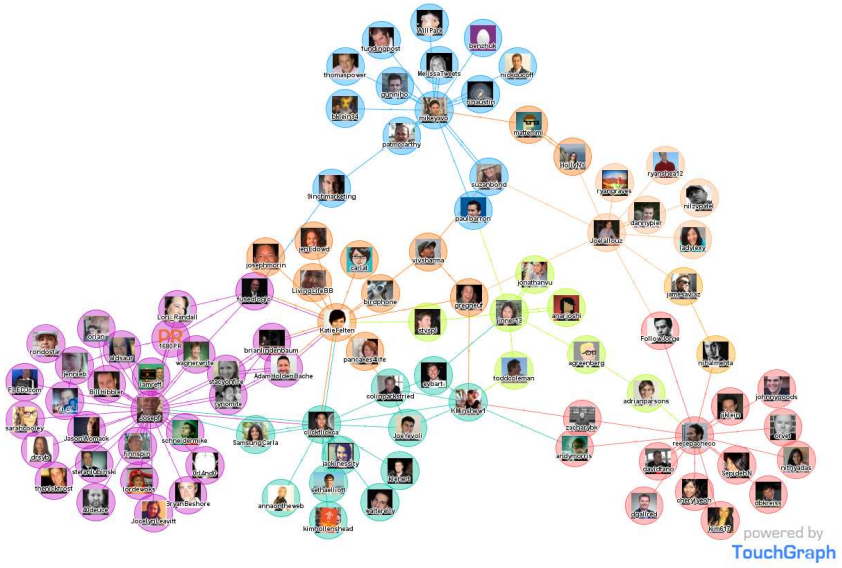


Figure 3.2: Content Graph in UGC networks.

tent items that, from now on, become part of a content graph where adjacent items are considered as "related". Figure 3.2[1] shows a part of the content graph of Twitter. The fact that these relations affect how users interact with content and how this can be leveraged by distributed systems to efficiently place content is the main focus of this chapter and in the remainder of this paragraph we present some of the work already done towards this direction.

Volley [ADJ+10] focused on sites like Live Mesh and Live Messenger and aims at exploiting content interconnections to place the data on data centers that are close to users that are like to request them. The target is to decrease the perceived latency. In this work the authors assume that only one copy is stored for each content item as data durability is not considered.

NetTube [CL09] is a peer-assisted video-on-demand system that leverages the correlated viewing patterns of users of services like YouTube and aims at reducing the start-up delays observed in previous systems that organized viewer of a given video in "swarms" through "social"-aware pre-fetching. In more detail, swarms consist of users watching the same video and a user joins a swarm when watching a video for the first time. To locate each of its consecutive choices, the user explores the Friend-of-a-Friend network to look it up and in the case that there is no hit, then the request is directed to the service provider's servers. In addition, and to smooth the playback, the user pre-fetches the prefixes (10 first seconds) of the 3 top related videos of the video she is currently watching. As one can see

[1]Source: http://www.touchgraph.com/news

from the above description, in NetTube the content graph is exploited implicitly during the placement process as content is replicated on nodes that request it. The only explicit exloitation of the content graph is during the pre-fetching phase, where related videos are pre-fetched to smooth the playback.

Finally, in SPAR [PES+10] the authors propose a novel online graph partitioning and replication algorithm that aims to provide better scalability properties to existing social networks. According to their scheme, is a social partitioning and replication system that achieves one-hop replication of user profiles in social networks: data of all the friends of a user are replicated on the same server as her own data thus enableing efficient decoupling and scaling of online social networks. At the core of SPAR lies an on-line greedy data placement algorithm coping with dynamic additions and removals of nodes, edges and servers.

3.2.2 Youtube and User Behavior

In this paragraph we focus on user behavior in UGC sharing sites and we focus on YouTube. The popularity of YouTube attracted the interest of many researcher, that have generated numerous studies on user behavior and video characteristics.

Cha et al. propose to use a video's history to predict its future demand [CKR+07]. In their work, among other useful results, the authors conclude that a video's early days can reveal a lot about its future popularity.

Zhou et al. also study a video's popularity evolution but this time from the standpoint of its position in the content graph. In this work, the authors show that there is a strong correlation between the view count of a video and the average view count of its top referrer videos. This implies that a video has a higher chance to become popular when it is placed on the related video recommendation lists of popular videos. In addition, they also find that the click through rate from a video to its related videos is high and the position of a video in a related video list plays a critical role in the click through rate. This is the work on which we based our user behavior model when evaluating DTube's performance.

Finally, in [BSW12], Brodersen et al. analyze the geographical distribution of videos' views in YouTube. Their main results are i) that most of the videos exhibit a high concentration of their views in a small number of regions aorund the world, ii) the impact of viral spreading on a video's popularity is not trivial, in fact they show that there is a threshold of 20% in how many views a video receives through "social" sources, below which these views help the video widen geographically its audience and above which, these views lead to further view concentration, and iii) a video's popularity expands geographically and then withdraws back to the main region of focus.

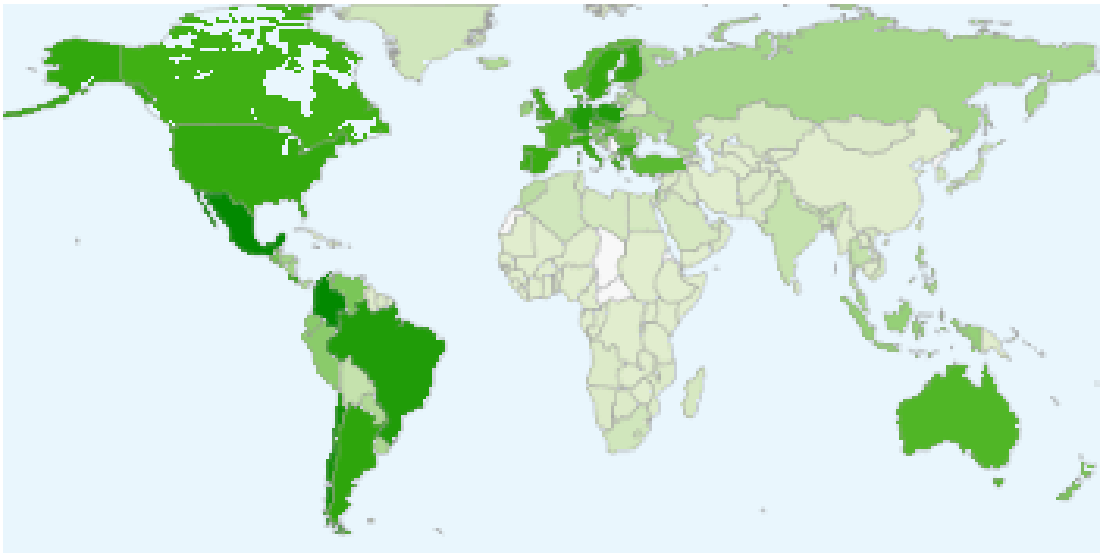


Figure 3.3: Geographic distribution of the origin of views for a sample YouTube video.

3.3 Locality in UGC

Given the above discussion, in this paragraph we move on to investigate the relation between the position of a video in YouTube's content graph and the distribution of the origins of its views around the world. Using a YouTube dataset we crawled in March 2011, in Section 3.3.2 we start by showing that for the vast majority of videos in YouTube, a small number of countries is responsible for most of their views. Trying to leverage this for the efficient content placement, in Section 3.3.3 we show that there is a strong correlation between the geographic distribution of a video's views and that of its related videos, and we leverage this finding to propose a simple yet efficient mechanism to predict the geographic distribution of a video's future views.

3.3.1 Dataset Description

For our study we crawled our own dataset from YouTube, during the first three weeks of March 2011, using snowball (BFS) approach. As a seed for our crawl, we used an initial set consisting of the 10 most popular videos for 25 different countries. This information is publicly available through the API provided by YouTube. For each video, we collected three attributes:

i its list of related videos as provided by YouTube,

ii its total number of views, and

iii its View Source Vector (VSV).

The VSV of a video represents how many views this video received from each country in the world. For most of the videos, the VSV is available, on the statistics page, as a colour map (Figure 3.3) generated by a specific URL (`charts.apis.google.com`) containing (country, percentage_of_views) couples encoded according to Google's Simple Encoding Format. In our experiments, we extracted the actual VSVs from these URLs. Table 3.1 shows the distribution of the views (top 8 countries) at the granularity of a country, over the whole dataset. The original dataset contained $1,063,844$ videos in total. From these, we removed the videos with no VSV, and filtered out non-crawled videos from the related video lists. This left us with $689,265$ videos, each having 8 related videos on average, for a maximum of 25 related videos allowed in YouTube.[2]

Country	US	CA	GB	BR	JP	DE	PL	AU
Prop. of views (%)	6.6	3.1	3.0	3.0	2.6	2.5	2.2	2.2

Table 3.1: Geographic distribution of views at the granularity of a country (top 8 countries).

In the following, we use the view-per-country information contained in the VSVs to analyze the geographic distribution of views in our dataset. The goal of our analysis is three-fold. At first we want to study for each video, the geographic distribution of its views. After that we investigate the relation between the way content is organized in UGC hosting sites and their viewing patterns and, finally, we explore the feasibility of a proactive placement mechanism that places videos in countries where they are most likely to be viewed.

Category	**# views**	**% of videos**	**% of total views**
C_1	$[0, 10^4]$	42.0	0.49
C_2	$(10^4, 10^5]$	33.5	5.10
C_3	$(10^5, 10^6]$	19.7	25.18
C_4	$(10^6, 10^7]$	4.4	45.02
C_5	$(10^7, 10^8]$	0.3	21.10
C_6	$(10^8, \infty)$	0.04	3.10

Table 3.2: Popularity categories and statistics

[2]Because the geographic information in our dataset is given at the granularity of a country, we conduct our analysis at the same granularity.

3.3.2 Popularity vs. Geographic View Distribution

We first investigate the link between a video's overall popularity and the geographic distribution of its views. To this end, we partition our dataset into six categories based on a video's number of views. The distribution of videos shown on Table 3.2 confirms earlier analysis [BSW12, CKR+07], highlighting a long-tail distribution of views. Very popular videos (categories C_5 and C_6) represent less than 1% of videos while accounting for almost 25% of all views. Because of the long tail, the bandwidth cost of "unpopular" videos is however far from being negligible: The 3 least popular categories (C_1, C_2, C_3, or 95.2% of all videos) still represent more than 30% of the total incoming requests.

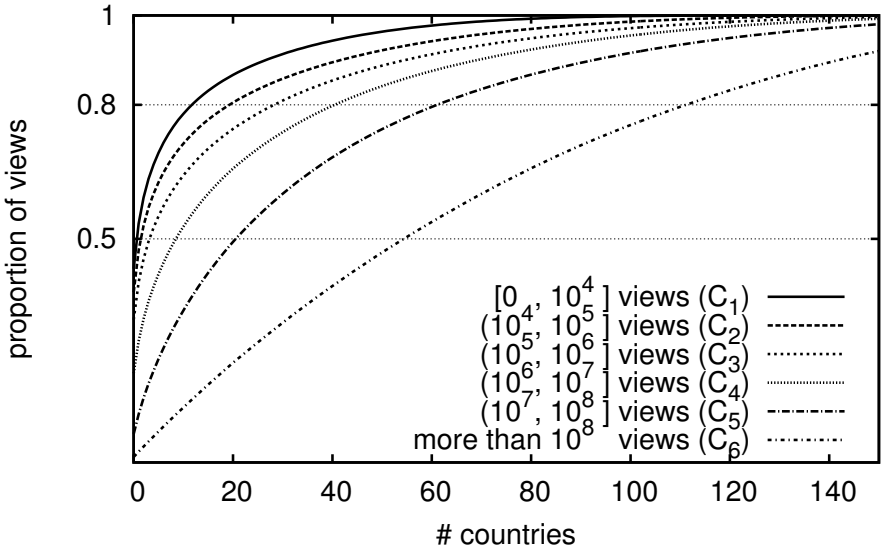


Figure 3.4: Geographic cumulative distribution of views for various popularity categories

To analyse the geographic distribution of views for each category, we compute the average geographic spread of video views as follows. For a given video, we sort the countries in its VSV in decreasing order according to the proportion of views originating from each country. We then compute the cumulative distribution of views of each video and plot the average over each popularity category. Figure 3.4 presents the results of the above study. A point (n, m) on the graph means that the n top countries for videos in this category account for m% of the total number of views of the category.

Figure 3.4 shows that the views of niche videos (category C_1, less than $10,000$ views) are geographically highly concentrated: 80% of all views for videos in C_1

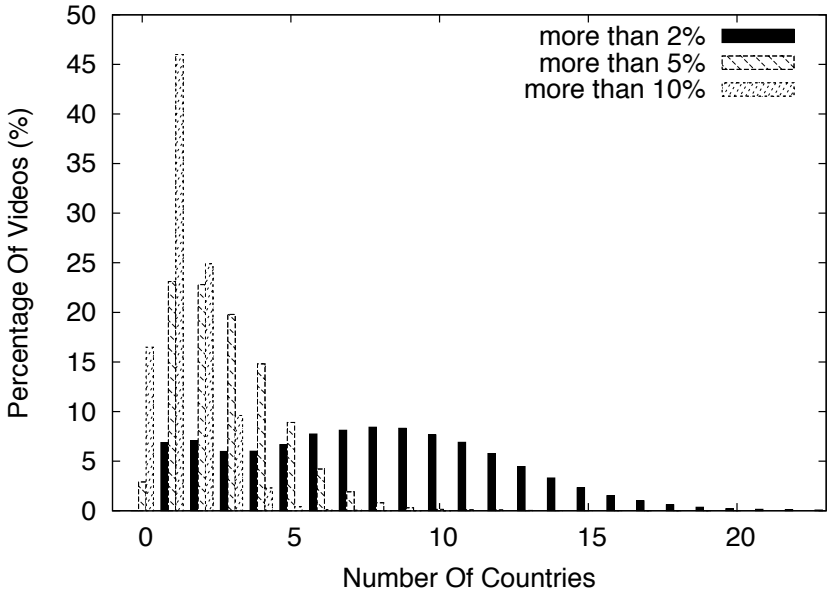


Figure 3.5: Geographic cumulative distribution of views for various popularity categories

come from less than 15 countries. This phenomenon fades out as the popularity of videos increases, to reach an almost uniform global distribution for extremely popular videos (category C_6).

Yet, this concentration effect remains relatively strong for all videos up to 100M views (categories C_1–C_5, 96.9% of all views). This implies that a content distribution system could benefit significantly from a proactive placement mechanism that accurately predicts a small number of countries in the first ranks of a video's VSV, and places its replicas accordingly. For instance, for videos in C_3 (between $100,000$ and 1M views, 25.18% of all views), proactively placing video replicas in (or close to) the top 25 countries would cover up to 80% of all views.

To further illustrate the potential gains a system may have by placing correctly a video in a small number of countries, in Figure 3.5 we rank the countries in a video's VSV in decreasing order, and we plot the percentage of videos (y axis) that has up to x countries responsible for more than 2%, 5% and 10% of its worldwide views. In this experiment we do not take into account the video's popularity, as we want to show the importance of a prediction mechanism from a video's perspective. From this graph, we can clearly see that most of the videos have a small number of countries responsible for the vast majority of their views.

We further study the geographic spread of the origins of the views by taking into account the distances between the main sources of views. The motivation behind this experiment is that it is easier to serve a video with low latency, with

a single replica, for users in France and in Switzerland than for users in the UK and in India. For each video, we compute the average of the pairwise distance between the main sources of views (i.e., the top country covering 80% of the views), weighted by the proportion of views each country is responsible for. For instance, for a video viewed 1,000 times, whose three main sources are US (500 views), UK (200 views), and Japan (100 views), our metric is:

$$\frac{(0.5+0.2)d(\mathsf{us},\mathsf{uk})+(0.2+0.1)d(\mathsf{uk},\mathsf{jp})+(0.1+0.5)d(\mathsf{jp},\mathsf{us})}{(0.5+0.2)+(0.2+0.1)+(0.1+0.5)},$$

In our dataset, we observed this average distance to be 26% less for unpopular videos ($\sim$5,200 km) than for popular ones ($\sim$7,000 km).

3.3.3 Geographic vs. Content Locality

To explore how the top n countries of a video might be predicted, we now turn to the relation between content locality and geographic locality. For each video, we compute the Spearman correlation coefficient between its sorted VSV, i.e., the list of all countries sorted by decreasing number of views, and that of each of its related videos. The Spearman coefficient is defined in Equation 4 and captures the correlation between the rank of countries in two sorted VSVs, taking into account the permutations of ranks. The closer the absolute value of the coefficient to 1, the more correlated the lists. The results of this experiment are plotted in Figure 2 as a function of the rank in the list of related videos. In our dataset, this correlation ranges from 0.64 to 0.68 which is relatively high. This means that a video's VSV can be inferred with high accuracy from that of its related videos. In addition, from Figure 2 we can see that the smaller the rank of a video in the related videos list, the higher the correlation of its VSV with the one of the video pointing to it (for the video at rank 1 we have a correlation of 0.68). This implies that the first related videos are the best candidates for VSV predictors.

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \tag{3.1}$$

In order to see if the above finding can be translated into an efficient mechanism for proactively placing video replicas close to where their future requests are most likely to come from, we conduct the following experiment. For a given video V and its first related $Rel(V)[1]$, we compute for a given number m of replicas,

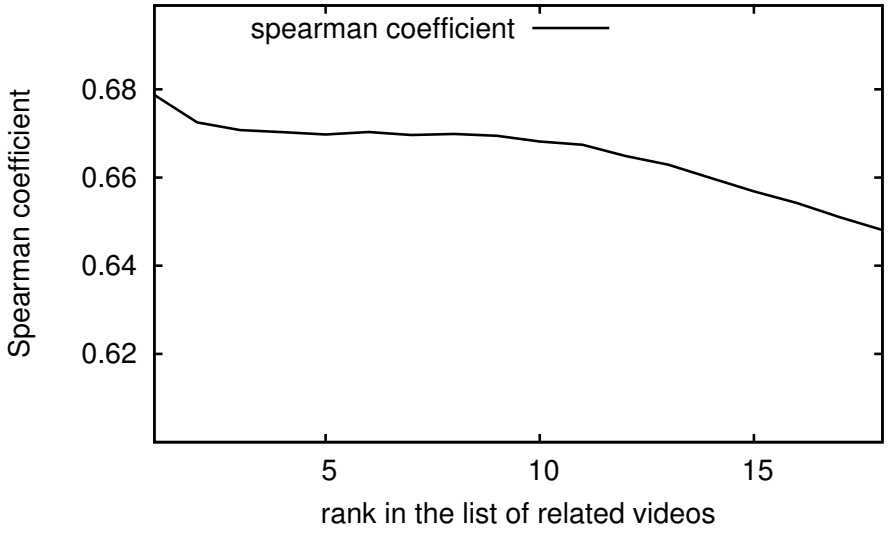


Figure 3.6: Spearman correlation coefficient between a video's geographic distribution of views and that of its related videos, as a function of their rank in the list of related videos

the percentage of views covered by placing them on the first m countries of the VSV of $Rel(V)[1]$, normalized by the percentage of views covered by placing the replicas on the first m countries of the actual VSV of V. The later corresponds to an ideal case where the placement mechanism knows in advance where the views will come from. The results are presented in Figure 3.7, and, as we can see, even for a small number of replicas, this simple prediction mechanism can accurately follow the actual geographic distribution of the views of a given video. For instance, 85% of the views covered by the first 5 countries of V's VSV are covered by the first 5 countries of $Rel(V)[1]$'s VSV.

In summary, unpopular videos, which represent a large proportion of the YouTube video collection, have (i) most of their views originating from a few countries which (ii) spread in a limited region, thus foreseeing a great potential for geographic locality-aware data placement. Furthermore, the geographic distribution of views of a video is strongly correlated with that of its related videos. This implies that the geographic distribution of views of a video can be predicted, but most importantly, it makes the case for a placement mechanism in which videos close in the content graph are stored geographically close to one another.

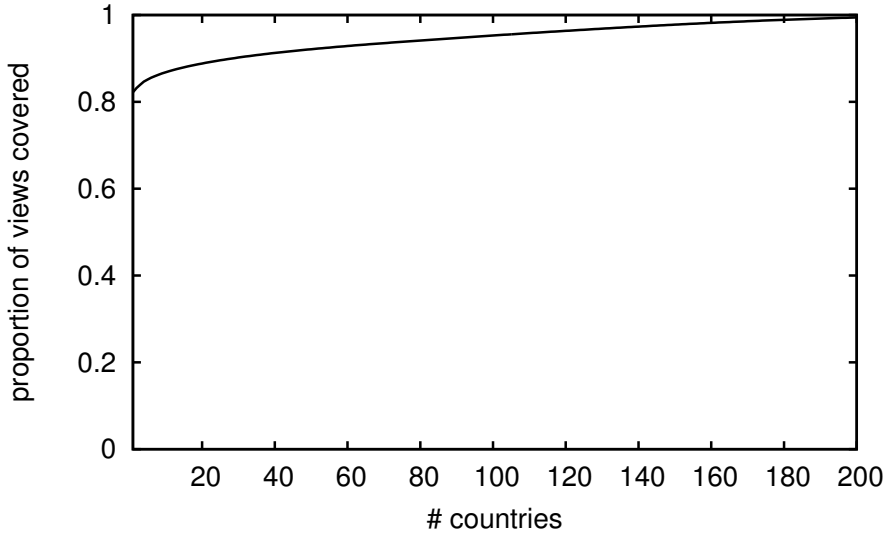


Figure 3.7: Views covered by placing replicas of a video V in top-countries of the VSV of V's first related video, normalized by the number of views covered when using V's actual VSV.

3.4 DTube

Building on the insight from the previous section, we propose DTube, a proactive content placement mechanism that places videos close to their future requests, extracting a video's geographical viewing patterns from the content graph. The purpose of the system is to accurately predict where a video's views will come from, just by looking at its position in the content-graph. In addition, and to further improve the accuracy of our placement mechanism, DTube places videos based on a package-based replication scheme that reflects locality in the content graph, as we will see in the following sections. Reactive strategies that create video replicas upon incoming requests, could be deployed on-top of DTube to further enhance its performance.

3.4.1 System model

We consider a User Generated Content sharing system that uses geographically distributed nodes as its storage/serving infrastructure. Our findings hold for both residential gateways [HCD09, MVCG11], peer-assisted CDN systems [HWLR08] and data centers distributed around the world.

DTube assumes the existence of a catalogue service that holds, for each video, the location of its replicas, its current View Source Vector and its meta-data including the list of its related videos. This service is used to retrieve up-to-

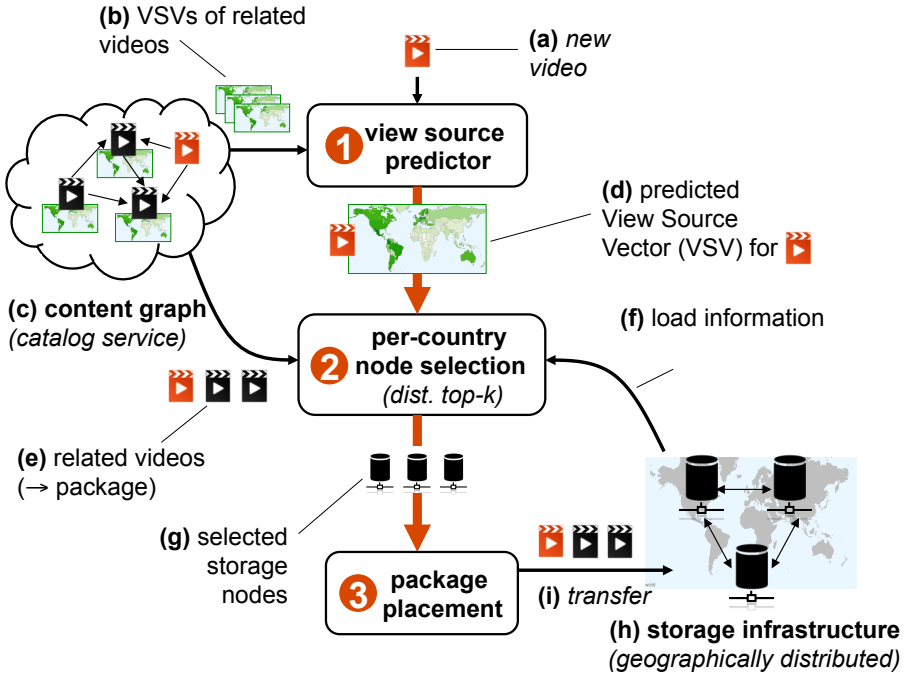


Figure 3.8: Overview of DTube's placement strategy.

date meta-data and statistics about videos. We further assume the existence of a recommendation algorithm [RRSB] that dynamically computes videos' lists of related videos and updates the catalogue accordingly.

3.4.2 Video placement

At a high level, DTube's video placement mechanism seeks to solve a relaxed partitioning problem that aims at placing related videos close to each other and close to where they will be consumed. Its design is guided by the observation that related videos are likely to share similar access patterns (Section 3.3). Exploiting this observation allows us to proactively place new videos accurately, while ensuring both geographic and content locality.

The key steps of DTube's placement mechanism are depicted in Figure 3.8, and we will present each one of them in detail in the remainder of this section. When a new video V is uploaded to the system (a), YouTube's recommendation mechanism computes its "Related Videos" list, thus making it part of the content

graph (c). DTube then estimates V's main future view sources, $\widehat{\mathsf{VSV}}(V)$ (d) which is later used to place replicas of V on the storage infrastructure. This step takes as input V's "Related Videos" list and outputs its predicted View Source Vector, $\widehat{\mathsf{VSV}}(V)$ (Step 1). After this step, DTube constructs V's packages (e), which contain (i) the video itself, and (ii) copies of its related videos (Step 2). Details on how $\widehat{\mathsf{VSV}}(V)$ is computed and why packages are used, are presented in the paragraphs that follow. Finally, in Step 3, DTube locates optimal storage nodes for V's packages in the top countries of the predicted VSV (d), according to criteria that we detail below (f).

3.4.2.1 Step 1: View source prediction

When a new video V is uploaded on DTube, the recommendation mechanism of the catalog service computes its list of Related Videos. This results in V becoming a node in the (directed) content graph of the system (c in Figure 3.8), with edges pointing from V to each one of its related videos. We denote by $Rel(V)$ the list of related videos of V and by $Rel(V)[i]$, the video V' in the i-th position in $Rel(V)$.

After this step, DTube leverages the findings of Section 3.3, and tries to predict the origins of V's future requests, by only looking at its position on the graph (d). To compute $\widehat{\mathsf{VSV}}(V)$, DTube obtains from the catalog service (b in the figure) the VSV of its top-related video, i.e. $Rel(V)[1]$, as shown by Equation 3.2. The reason for this is illustrated by Figure 2 and is that the higher the rank of a video in V's "Related Videos" list, the higher the correlation between its VSV and the one of V. This is a simple strategy but as we show in Section 3.5 performs well. Other more complex strategies could also be applied.

$$\widehat{\mathsf{VSV}}(V) = \mathsf{VSV}(Rel(V)[1]) \tag{3.2}$$

3.4.2.2 Step 2: Package Creation

In Step 2, DTube uses the previously computed $\widehat{\mathsf{VSV}}(V)$ to place $\Re$ replicas of V on nodes in the system. $\Re$ is a system parameter called replication factor and corresponds to the the minimum number of replicas a video must have to guarantee its durability, i.e. not being lost due in case of storage node failure. The $\Re$ replicas of V are not stored alone however: each of them attracts a replica of each video related to V ($Rel(V)$) (e on the figure). We call this bundle of $|Rel(V)| + 1$ videos, containing V and its related videos, a package. V is the primary replica of the package while the other videos are the package's secondary replicas.

This package mechanism creates a first coupling between graph locality and

storage locality, leading related videos to be stored on the same node. Thanks to this package mechanism, most videos also receive more than $\Re$ replicas: they get their $\Re$ primary replicas, plus a number of secondary replicas each time they appear in another video's related list. The net result is a coupling between a video's number of replicas (primary and secondary) and its in-degree in the content graph. The above is a desirable property as Zhou et al. in [ZKG10] show that there is a strong correlation between the view count of a video and the view counts of its top referrer videos (i.e. its in-degree). This ensures that videos that are more likely to be watched, also get more replicas. Given this, we can see that creating videos for a replica proportionally to its in-degree could further help at balancing the request load that each node has to serve. A popular video will have more replicas thus leading to more nodes being able to serve requests for it.

3.4.2.3 Step 3: Storage node selection

Having the $\widehat{\mathsf{VSV}}(V)$ and the packages of V, the final decision to be made is on which node to place these packages. Each package of a video V is placed on a node in each of the $\Re$ first countries of $\widehat{\mathsf{VSV}}(V)$ (d) as most views are expected to come from these countries. To minimize transfer and storage costs, only replicas of the videos that do not exist in the country are transferred. In addition, to evenly balance the storage load among the nodes in the system, for a node to be eligible to store a new package, the number of videos it already stores must be lower than the average storage load over all nodes. The average load in the system can be computed with a standard averaging gossip protocol. Finally, copies of the package are transferred (i) to the selected nodes (g).

3.5 Evaluation

In this section, we evaluate through simulations the performance of DTube with respect to the geographic distance between users that request a video and the storage nodes serving it. In addition, we compare it against a system that employs reactive caching on top of persistent storage.

3.5.1 Evaluation Setup

We distribute the storage nodes in countries according to the proportion of views originating from this country, as observed in our dataset (see Table 3.1). We set the number of storage nodes to $10,000$ and we consider the videos from our dataset, with the corresponding popularity and the content-graph induced by the related video feature.

We generate synthetic view traffic based on individual users' behavior, using the model proposed in [ZKG10], with the popularity values from our dataset: We consider a number of users ($50,000$ in our experiments), distributed across all countries for which we have information in our dataset. Users are distrbuted in the same way as storage nodes, according to the geographic distribution of views observed in our dataset (see Table 3.1). The number of videos a user watches during a session is picked at random, with an average value of 10. The first video V a user watches is selected from the whole set of videos according to the probability of this video being watched in her country, i.e. the number of views for V originating from her country divided by the total number of views originating from her country (for all videos). Each subsequent video she watches is selected among the related videos of the previously watched video, excluding already viewed ones. The probability of a video being picked is set to be inversely proportional to the video's rank in V's list of related videos, following a Zipf distribution.

3.5.2 DTube and Alternatives

We compare DTube against standard caching. For DTube, we use the placement algorithm as described in Section 3.4. In addition, we implement and evaluate several variations of DTube to identify the performance gains conveyed by the different mechanisms involved, namely without the use of packages (Partial DTube) and using the actual VSV of the video (Ideal DTube) instead of that of its first related video. Ideal DTube can be thought of as an upper bound on DTube's performance and reflects how the efficiency of the VSV prediction mechanism (evaluated in Section 3.3) translates in practice with respect to the viewer experience. In our experiments, videos are served from the node closest to the user.

As for caching, we consider a storage infrastructure composed of persistent storage nodes (e.g., YouTube servers) and CDN caching nodes (e.g., Akamai servers). The persistent storage nodes hold a complete copy of the YouTube dataset, thus ensuring durability. Videos are only served by CDN nodes, the caches of which are populated in a reactive fashion, based on the users' view traffic: Consider a user located in a given country who requests a given video. If the video is stored on a CDN node in this country, it is served from this node to the user. If not, the video is first fetched from a persistent node to a random CDN node (with free storage space) in the country and then served. If none of the CDN nodes in the country has sufficient free storage space to store the video, we apply LRU cache replacement: the least recently used video is replaced by the new entry.

3.5.3 Evaluation Results

We evaluate and compare the performance of all placement strategies with respect to the geographic distance between the user and the node serving the video. More specifically, we look at (i) the out-country hit-rate, that is the proportion of requests that are served from a storage node (i.e., a gateway or a CDN node) located in a different country than the user, and (ii) the distance between the user and the storage node when the video is served from a different country. We assume that networking infrastructure is usually well integrated in each country, thus in-country hits are likely to encounter better network quality and that geographic distance is a good indicator for transfer latency.

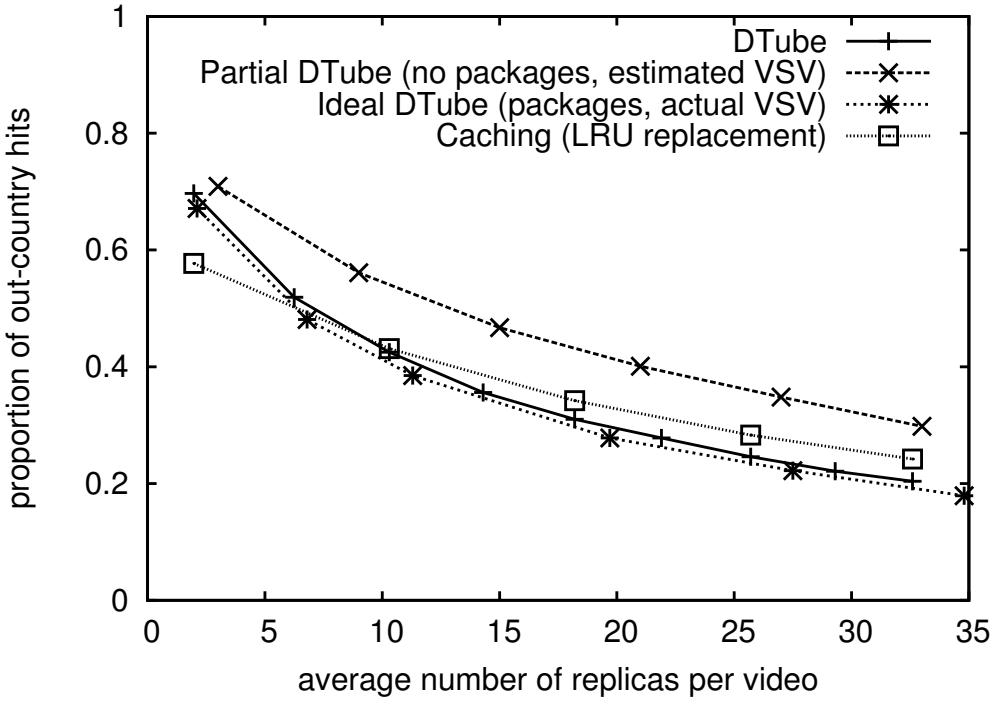


Figure 3.9: Proportion of out-country requests with DTube and caching.

In order for DTube and caching to be comparable, we use the same storage space in both. More specifically, for a given replication factor $\Re$, we first run simulations with DTube. Because it makes use of packages, the average number of replicas R per video is larger than $\Re$. We therefore run simulations with caching, for a system composed of $\Re$ persistent storage nodes and CDN nodes with a storage space of $(R - \Re) \times$ (total number of videos)/(number of CDN nodes), which corresponds to the same total storage space as for DTube. We evaluate

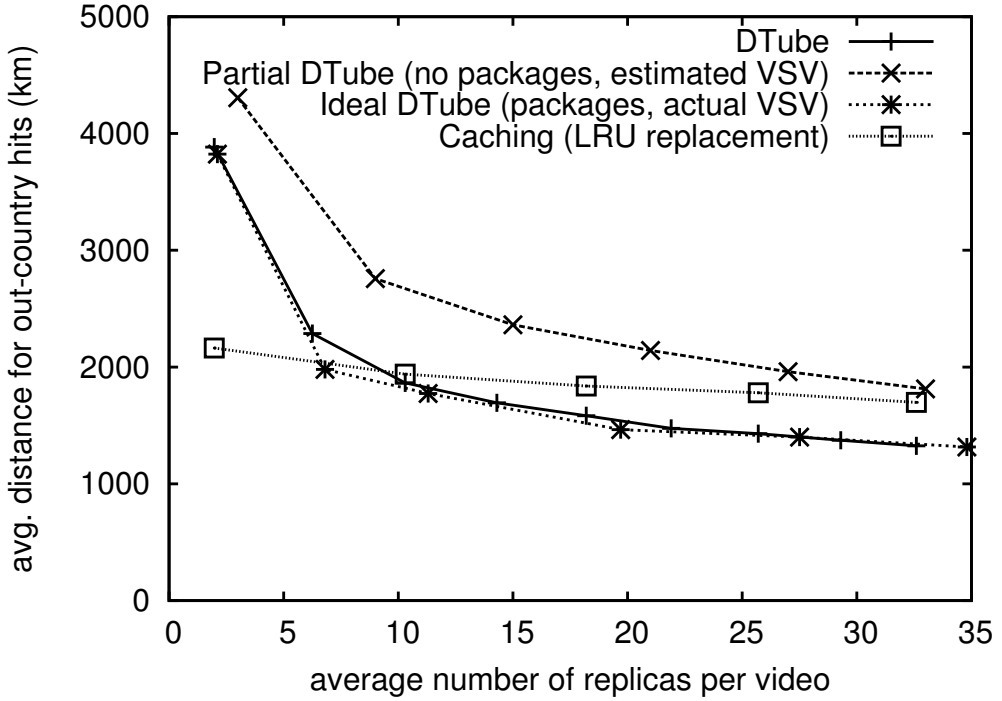


Figure 3.10: Average distance to storage node for out-country requests with DTube and caching.

our metrics at steady state, i.e., when all the caches of all CDN nodes are full.

Fig. 3.9 depicts the out-country hit-rate for the different versions of DTube and caching. It can be observed that for larger values of the average number of replicas per videos, DTube outperforms the caching-based solution. For instance, for an average number of 30 replicas per video, DTube decreases the proportion of out-country request from 0.25 to 0.21, that is a 16% improvement. By comparing DTube to Ideal DTube and Partial DTube, we observe that (i) the use of packages accounts for a significant part of DTube's performance, (ii) the estimation of a video's VSV from that of its related videos (i.e., based on the content graph) incurs only a little decrease in performance compared to an omniscient solution in which the actual VSV is known in advance. This illustrates the synergy between our geographic prediction and the use of packages.Similar results can be observed in Fig. 3.10, which depicts the average distance between the user and the storage node serving the video for out-country requests. For instance, with an average of 30 replicas per videos, DTube reduces the average distance by 29%. Note that the distance remains relatively high as the distance to the closest country can be large, e.g., $\sim$ 2000km between the US and Canada which are the two main sources of views.

3.6 Conclusion

In this chapter we have highlighted the correlation between content locality and geographic locality in User Generated Content (UGC). More precisely, we have shown using a large YouTube dataset that related videos present similar geographic viewing patterns and that, except for extremely popular videos, video views are concentrated in a limited number of countries.

This coupling between content and geographic locality in UGC system has led us to propose DTube, a decentralized storage infrastructure which proactively places content close to their future requests leveraging on the videos' positions in the content graph.

Chapter 4

Conclusion and Perspectives

This chapter concludes this document by giving a summary of the previous chapters and potential directions for future work.

4.1 Summary

In this thesis, we tried to tackle some of the challenges that come with the possibly unlimited opportunities of the "Big Data" era. Our main objective was to optimize the storage infrastructure so that it can safely store the constantly increasing volume of data while guaranteeing timely access to them. To this end, the second chapter focuses on scalable archive storage systems that are able to accommodate huge volumes of data while making efficient utilization of the available resources, while in the third chapter we deal with content placement strategies on geographically distributed systems, so that user perceived latency, when fetching the content, is minimized.

Data Deduplication

To protect their data from hardware failures and natural disasters, organizations but also individuals rely on frequent data backups. Trying to keep up with the rapid growth in the volume of produced data, new generation backup systems face huge scalability challenges. Trying to answer these challenges by simply buying more hardware would imply huge operational costs. To this end, we focus on cluster-based backup systems that apply data deduplication to reduce the storage and network resources needed to back up a given workload. Data deduplication stands for the idea of replacing duplicate regions of data with references to their already stored "identical twins". This technique, although simple, is reported to reduce dramatically the storage needs in production backup systems (up to 30x

when combined with classic compression) but integrating it efficiently in storage systems presents a number of design challenges. To answer these challenges, we propose Produck, a cluster-based deduplication system that combines state-of-the-art techniques with novel contributions in order to provide good deduplication at high throughput and minimal resource requirements while keeping the storage load equally spread among the nodes in the system. Our two contributions are (i) a novel set-intersection mechanism that is for the first time used in the context of storage systems and (ii) a novel bucket-based storage quota management algorithm that manages to keep the system load balanced with minimal intervention in the deduplication process. Our experiments show that, Produck provides better deduplication than other resource-friendly solutions (stateless strategies) while using a lot less resources and being a lot faster than solutions that choose to sacrifice resources to achieve better deduplication (stateful strategies). This work highlights the benefits of deduplication for backup systems but also the questions that a system designer has to answer when designing such systems.

Content Placement

For a geographically distributed content delivery system to scale while providing good quality of service, one of the main questions it has to answer is how to efficiently place content on its servers so that the user perceived latency is minimized. The goal in most cases is to place content on servers close to where requests are expected to come from. In the third chapter we focus on User-Generated Content (UGC) sharing sites and we leverage content organization properties that are inherent in these sites, to propose a novel content placement strategy that manages to achieve the above objective. The main feature that we leverage is that items in the collections of these sites are no longer independent, as in the case of previous sites, but are organized in a graph that in the literature is termed as content graph. The content graph of each site reflects the relations between the items for the site and although there may be semantic differences from one site to another, they all share many common characteristics. This allows us to believe that although we use YouTube as our usecase, many of our findings can be generalized for other sites as well. For our study, we use a large ($650,000$ videos) dataset that we crawled from YouTube to study properties of its content graph. We initially verify previous results that the organization of content in the content graph has an important impact on how users interact with it and we take this finding one step further by showing that the position of a video in the content graph influences the geographic distribution of its views. In more detail, we show (i) that less popular videos tend to have most of their views coming from a small number of countries while more popular ones tend to have their

views more uniformly spread all around the world and (ii) that "related videos" tend to have highly correlated viewing patterns, i.e. they tend to have most of their views coming from the same countries. Leveraging the later, we propose a simple yet efficient prediction mechanism that allows us to know in advance the main sources (geographic regions) of views for a given video, and based on this knowledge we select on the servers of which country to place the video's replicas. Our experimental evaluation shows that leveraging the content graph for content placement allows a system to place replicas of videos closer to the origins of their requests, compared to classic reactive content placement techniques that tend to aggressively replicate content on servers located in a given region as soon as a request for it comes.

4.2 Open Problems

In this thesis we focus on improving the performance of storage systems in the "Big Data" era and we propose two systems that leverage content properties to do so. Although these systems manage to face some of the challenges, there is a number of open questions that have to be addressed. In this section, we present some of these questions that we consider crucial for both backup and content delivery systems.

Fault Tolerance in Archival Storage Systems

In Chapter 2 we show that data deduplication provides a way to drastically reduce the storage space needed to back up a given workload. This enables the reduction of the costs related to the construction and maintenance of disk-based backup systems, thus making them financially viable even for small to medium sized organizations that can now change from tape-based solutions with low throughput, to disk-based ones.

Although deduplication reduces the storage needs of a given workload, another requirement that a backup system has to satisfy is fault-tolerance. Fault-tolerance means that data integrity must be guaranteed against any danger that threatens it. Given that with deduplication, magnetic disk is used as the main means of storage, disk-based backup systems face the same fault-tolerance challenges as the users whose data the system stores. In addition, given that with deduplication a chunk of data may belong to multiple files, this implies that a single disk failure may corrupt multiple files. Backing up the backup system on tape, would eliminate the benefits of disk-based storage, as far as throughput is concerned, as the final system throughput would be dictated by its bottleneck, thus the tape.

To guarantee fault tolerance and high throughput, redundancy has to be added in the system. Redundancy stands for the fact that multiple copies of the data are kept, so that if one is lost, data can be recovered from the remaining ones. Adding redundancy to a system that tries to eliminate it in order to reduce a workload's storage needs, seems contradicting. But, as explained, the roles of the two mechanisms, i.e. deduplication and replication, are complementary. Reconciling these two goals in order to provide a backup system with the same or even better fault tolerance properties as a tape-based one but with better throughput and better storage utilization, implies the study and comparison of many different design options that may or may not fit in such a system.

Consistency in Geographically Distributed Storage Systems

In Chapter 3 we focused on content placement in User Generated Content (UGC) sharing sites, so that the geographic distance between the user that requested the content and the server that actually serves it, is minimized. Guaranteeing timely access to the content is of critical importance for the success of UGC sharing sites. But, this is not the only requirement.

One of the open questions that is especially interesting in the context of interconnected content, is consistency. As explained in Chapter 3, content in UGC sharing sites is organized in a content graph. This implies that items on these sites are no longer independent and, in the context of consistency, modifications on one can have an impact on its adjacent ones in the graph. Given that the content of these sites is stored in data centers all around the world, maintaining replicas consistent is a challenging problem that becomes even more challenging as modifications on one item (e.g. a user's timeline in Twitter), imply modifications on other items (his "followers"' timeline in the above example). As we can see, given the content graph structure, updates now trigger a chain reaction that affects part of the graph.

In these settings, innovative content placement techniques that collocate related content can facilitate the process and novel data-types and consistency protocols can be applied to guarantee the convergence of the state of the replicas while maintaining the required invariants.

Bibliography

[ABC+02] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In OSDI, 2002.

[ADJ+10] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In NSDI, 2010.

[BDGM95] S. Brin, J. Davis, and H. Garcia-Molina. Copy Detection Mechanisms for Digital Documents. In SIGMOD, 1995.

[BELL09] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge. Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup. In MASCOTS, 2009.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In WWW, 1998.

[BSW12] A. Brodersen, S. Scellato, and M. Wattenhofer. YouTube Around the World: Geographic Popularity of Videos. In WWW, 2012.

[cis12] Cisco Visual Networking Index: Forecast and Methodology, 20112016. Technical report, Cisco, May 30, 2012.

[CKR+07] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In IMC, 2007.

[CL09] X. Cheng and J. Liu. NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing. In INFOCOM, 2009.

[CLYL08] Z. Chen, C. Lin, H. Yin, and B. Li. On the Server Placement Problem of P2P Live Media Streaming System. In PCM, 2008.

[DDL+11] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in Scalable Data Routing for Deduplication Clusters. In FAST, 2011.

[DF03] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In ESA, 2003.

[DGH+09] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. HYDRAstor: a Scalable Secondary Storage. In FAST, 2009.

[DSL10] B. Debnath, S. Senguptaz, and J. Li. Chunkstash: Speeding up Inline Storage Deduplication using Flash Memory. In USENIX ATC, 2010.

[eco] http://www.economist.com/node/21553445.

[EGH+03] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. In ACM Transactions on Computer Systems (TOCS), 2003.

[EGKM04] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. In Computer, 2004.

[FM85] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences, 31, 1985.

[GALM07] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From The Edge. In IMC, 2007.

[GCM+08a] J. F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, and A. Toncheva. The diverse and exploding digital universe: an updated forecast of worldwide information growth through 2011. Technical report, IDC, 2008.

[GCM+08b] J. F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, and A. Toncheva. The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011. Technical report, An IDC White Paper - sponsored by EMC, March 2008.

[GE11] F. Guo and P. Efstathopoulos. Building a High-performance Deduplication Systems. In USENIX ATC, 2011.

[GKM03] A. J. Ganesh, A. M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. In Computers, 2003.

[HCD09] J. He, A. Chaintreau, and C. Diot. A Performance Evaluation of Scalable Live Video Streaming with Nano Data Centers. Computer Networks, 53:153–167, 2009.

[HFC+08] Y. Huang, T. Z. J. Fu, D.-M. Chiu, J. C. S. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P-VoD Systems. In SIGCOMM, 2008.

[HWLR08] C. Huang, A. Wang, J. Li, and K. W. Ross. Understanding Hybrid CDN-P2P: Why Limelight Needs its Own Red Swoosh. In NOSSDAV, 2008.

[KDLT04] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey. Redundancy elimination within large collections of files. In USENIX ATC, 2004.

[KRT07] J. Kangasharju, K. W. Ross, and D. A. Turner. Optimizing File Availability in Peer-to-Peer Content Distribution. In INFOCOM, 2007.

[KUD10] E. Kruus, C. Ungureanu, and C. Dubnicki. Bimodal Content Defined Chunking for Backup Streams. In FAST, 2010.

[KZGZ11] S. Khemmarat, R. Zhou, L. Gao, and M. Zink. Watching User Generated Videos with Prefetching. In MMSys, 2011.

[LEB+09] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. In FAST, 2009.

[LV03] P. Lyman and H. R. Varian. How Much Information? Technical report, University of California at Berkeley, 2003.

[MB11] D. T. Meyer and W. J. Bolosky. A Study of Practical Deduplication. In FAST, 2011.

[MBN+06] S. Michel, M. Bender, N. Ntarmos, P. Triantafillou, G. Weikum, and C. Zimmer. Discovering and Exploiting Keyword and Attribute-Value Co-occurrences to Improve P2P Routing Indices. In CIKM, 2006.

[MCM01] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In SOSP, 2001.

[MVCG11] M. Marcon, B. Viswanath, M. Cha, and K. P. Gummadi. Sharing Social Content from Home: A Measurement-driven Feasibility Study. In NOSSDAV, 2011.

[PAK07] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting Similarity for Multi-Source Downloads Using File Handprints. In NSDI, 2007.

[pbb] http://www.emc.com/collateral/analyst-reports/idc-worldwide-purpose-built-backup-appliance.pdf.

[PES+10] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The Little Engine(s) That Could: Scaling Online Social Networks. In SIGCOMM, 2010.

[PP04] C. Policroniades and I. Pratt. Alternatives for Detecting Redundancy in Storage Systems Data. In USENIX ATC, 2004.

[PWB07] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure Trends in a Large Disk Drive Population. In FAST, 2007.

[QD02] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In FAST, 2002.

[RCP08] S. Rhea, R. Cox, and A. Pesterev. Fast, inexpensive content-addressed storage in foundation. In USENIX ATC, 2008.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In Middleware, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In SIGCOMM, 2001.

[RRSB] F. Ricci, L. Rokach, B. Shapira, and P. B.Kantor. Recommender Systems Handbook. Springer.

[SCKB06] A. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting Behind Akamai. In SIGCOMM, 2006.

[SG07] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In FAST, 2007.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In SIGCOMM, 2001.

[SMMC11] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades. In WWW, 2011.

[SSF08] M. Saxena, U. Sharan, and S. Fahmy. Analyzing Video Services in Web 2.0: A Global Perspective. In NOSSDAV, 2008.

[TM10] B. R. Tan and L. Massoulié. Adaptive Content Placement for Peer-to-Peer Video-on-Demand Systems. CoRR, abs/1004.4709, 2010.

[TPAK10] K. Tangwongsan, H. Pucha, D. G. Andersen, and M. Kaminsky. Efficient Similarity Estimation for Systems Exploiting Data Redundancy. In INFOCOM, 2010.

[UAA+10] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Calkowski, C. Dubnicki, and A. Bohra. HydraFS: a High-Throughput File System for the HYDRAstor Content-Addressable Storage System. In FAST, 2010.

[VLM+09] V. Valancius, N. Laoutaris, L. Massouli, C. Diot, and P. Rodriguez. Greening the Internet with Nano Data Centers. In CoNEXT, 2009.

[VOE11] R. L. Villars, C. W. Olofson, and M. Eastwood. Big Data: What It Is and Why You Should Care. Technical report, IDC, 2011.

[WDQ+12] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu. Characteristics of Backup Workloads in Production Systems. In FAST, 2012.

[XJFH11] W. Xia, H. Jiang, D. Feng, and Y. Hua. SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput. In USENIX ATC, 2011.

[YLZ+10] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li. LiveSky: Enhancing CDN with P2P. ACM TOMCCAP, 6:16:1–16:19, 2010.

[ZHS+04] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. In IEEE Journal on Selected Areas in Communications, 2004.

[ZKG10] R. Zhou, S. Khemmarat, and L. Gao. The Impact of YouTube Recommendation System on Video Views. In IMC, 2010.

List of Figures

Résumé en français

Introduction

Les fournisseurs de services de cloud computing, réseaux sociaux et entreprises de gestion des données ont assisté à une augmentation considérable du volume de données qu'ils reçoivent chaque jour. Des études récentes [VOE11, GCM+08a] ont montré que la taille des données produites et stockées a augmenté de façon exponentielle ces 5 dernières années et qu'en extrapolant son taux de croissance pour l'année 2020, on peut s'attendre à ce que la taille de notre "monde numérique" atteigne 35 ZB ou 35 trillions GB. Pour comparaison, en 2009, l'ensemble de notre écosystème numérique contenait seulement $0,8$ ZB de données, ce qui correspond à peu près à 2.3% du volume prévu pour 2020.

Cette augmentation rapide de la production de données est principalement due à (i) l'extension des limites géographiques de l'Internet : les utilisateurs avec un smartphone peuvent par exemple s'y connecter de presque n'importe où, et (ii) à des changements majeurs dans la façon dont les utilisateurs interagissent avec l'Internet. Premièrement, l'utilisateur est aujourd'hui placé au centre du processus de création de contenu. Dans les premières années de l'Internet, la production et la publication de contenu étaient le privilège d'un petit nombre d'individus ou d'entreprises. En effet, dans l'article original de PageRank [BP98], S. Brin et L. Page, mentionnent qu' en 1994, l'un des premiers moteurs de recherche du Web, le World Wide Web Worm (WWWW), contenait seulement 110000 pages et documents Web. Cela était dû (i) aux coûts liés à l'accès Internet et l'hébergement de contenu et (ii) aux connaissances techniques nécessaires pour le faire. De nos jours, ces deux obstacles ont été supprimés. Le coût d'une connexion Internet haut-débit a été grandement diminué dans la plupart de pays développés. En outre, les services tels que Facebook, YouTube, Flickr, Twitter et bien d'autres permettent à l'utilisateur de publier tout, à tout moment, gratuitement et ce, sans aucun pré-requis technique. L'impact de cette révolution peut être comparé à l'impact que l'invention de la typographie a eu sur le processus de publication au milieu du 15e siècle. Le second changement ayant eu un important impact sur l'augmentation des données produites, provient de l'augmentation spectaculaire des appareils situées à la périphérie du réseau qui produisent des données et ont accès à l'Internet. Il s'agit notamment de capteurs embarqués, téléphones intelligents et les tablettes électroniques.

Toutes ces données créent des nouvelles opportunités pour étendre la connaissance humaine dans des domaines comme la génétique, la santé, l'urbanisme et le comportement humain et permettent d'améliorer les services offerts comme la recherche, la recommandation, et bien d'autres. Ce n'est pas par accident que plusieurs universitaires mais aussi les médias publics se réfèrent à notre

époque comme l' époque "Big Data". Mais ces énormes opportunités permises par cette abondance de données ne peuvent être éxploitées que grâce à de meilleurs systèmes de gestion de données. D'une part, ces derniers doivent accueillir en toute sécurité ce volume énorme et sans cesse croissant de données et, d'autre part, être capable de les restituer rapidement afin que les applications puissent bénéficier de leur traitement. L'incapacité à atteindre l'un des objectifs ci-dessus peut limiter drastiquement les possibilités offertes par les "Big Data". La protection des données contre les dangers tels que les pannes du matérielle et les catastrophes naturelles est essentielle, afin déviter à leur perte ou leur corruption. En outre, l'incapacité à suivre leur rythme d' augmentation, conduirait inévitablement à la suppression des données en raison du manque d'espace de stockage, ce qui, à son tour, peut conduire à une "perte de mémoire" avec des conséquences importantes. Par exemple, une simple recherche sur des événements comme les élections nationales qui ont eu lieu il y a cinq ans, révélerait que la plupart des liens qui pointaient vers des articles sur l'événement sont maintenant morts et l'article correspondant a été supprimé. Des phénomènes comme cela peuvent conduire à une perte de mémoire humaine ou, pire encore, l'écriture contrôlée de l'histoire, que seules les informations qui ont "survécu" seront disponible pour les historiens de l'avenir. Enfin, ne pas être capable de servir le contenu d'une manière rapide et utile, peut (i) décourager les utilisateurs d'utiliser les services, stoppant ainsi la fourniture de données menant à la détérioration de service et (ii) conduire à des dysfonctionnements graves du système, en cas d'applications à temps critique. À cette fin, beaucoup d'efforts à la fois du milieu universitaire et l'industrie sont investis afin d'"apprivoiser" ces "Big Data".

Ce document se concentre sur ces deux défis relatifs aux "Big Data". Dans notre étude, nous nous concentrons sur le stockage de sauvegarde (i) comme un moyen de protéger les données contre un certain nombre de facteurs qui peuvent les rendre indisponibles et (ii) sur le placement des données sur des systèmes de stockage répartis géographiquement, afin que les temps de latence perçue par l'utilisateur soient minimisés tout en utilisant les ressources de stockage et du réseau efficacement. Tout au long de notre étude, les données sont placées au centre de nos choix de conception dont nous essayons de tirer parti des propriétés de contenu à la fois pour le placement et le stockage efficace.

Probabilistic Deduplication for Cluster-Based Storage Systems

En se concentrant sur le premier objectif, qui est le système de sauvegarde pour l' époque de "Big Data" , nous proposons Produck, une solution de stockage à base de cluster qui applique la déduplication pour réduire les besoins en bande passante et en espace de stockage. La Déduplication peut être considérée comme un type

de compression, puisque son objectif est d'éliminer la redondance inter-fichiers, inhérente aux données stockées dans le but de réduire leur volume.

Dans les systèmes de déduplication, les données sont divisées en chunks de tailles variables et chaque chunk n'est stocké qu'une seule et unique fois. Quand un nouveau fichier arrive et contient un chunk qui se trouve être identique à un autre déjà stocké dans le système, ce chunk est remplacé par une référence vers celui déjà enregistré. Pour comparer le contenu de deux chunks, on calcule pour chacun son empreinte par application d'une fonction de cryptographie de hachage sur son contenu. Pour déclarer que deux chunks sont identiques, au lieu de comparer leur contenu octet par octet, leurs empreintes sont comparées et si elles sont égales, leur contenu est considéré comme identique.

Definition du problème Dans un système de déduplication à base de cluster, le challenge principal est de maintenir le taux de déduplication élevé tout en maintenant la charge de stockage répartie également entre les noeuds du système. La raison pour laquelle cela constitue un challenge est que, d'une part, la déduplication essaie de clusteriser les données du cluster autant que possible, de sorte que tous les chunks en double se retrouvent sur le même noeud, tandis que la repartition de la charge du stockage vise à diviser la charge également entre les noeuds disponibles. Les solutions proposées peuvent être divisées en deux catégories, les stateful et les stateless. À un extrême, les solutions stateful s'appuyent sur l'état stocké pour chaque noeud dans le système afin de maximiser la déduplication. Cependant, le coût de ces stratégies en termes de ressources de calcul et de mémoire, rend leur déploiement à grande échelle difficile ou même irrealisable. À l'autre extrême, les stratégies stateless stockent les données basées uniquement sur leur contenu, sans qu'il soit tenu compte des décisions de placement précédentes, ce qui permet de réduire le coût mais aussi l'efficacité de la déduplication.

Contributions Dans cette thèse, nous proposons Produck, une strategie de sauvegarde stateful et légère qui offre de meilleurs taux de déduplication comparé aux stratégies stateless de l'état de l'art, tout en restant proche des performance des solutions stateful. En outre, la performance de Produck est obtenue à un coût très faible en termes de calcul et de mémoire supplémentaire, par rapport aux stratégies stateful, conduisant au stockage de données plus efficace. Pour cela, nous proposons deux contributions principales: un mécanisme probabiliste et léger qui permet à Produck de choisir le meilleur noeud pour stocker un chunk et une strategie de repartition de la charge de stockage, innovante. La première permet à Produck d'identifier rapidement les noeuds qui peuvent fournir les plus forts taux de déduplication pour un chunk et la deuxième garantit la repartition de la charge de façon uniforme entre les noeuds du système.

Architecture L'architecture de Produck est identique à celle de l'état de l'art des systèmes de déduplication en cluster et se compose du Client, du Coordinator

et des StorageNodes. Les utilisateurs interagissent avec Produck via le Client qui est responsable de la division du fichier en chunks quand un fichier doit être stocké et reconstruit le fichier à partir de ses chunks quand le fichier doit être récupéré. En outre, et afin d'améliorer la performance du système, le Client organise les chunks en superchunks, qui sont des groupes de chunks consécutifs. Les chunks appartenant au même superchunk sont stockés ensemble sur le même StorageNode. Le Coordinator est l'entité qui décide qui parmi les StorageNodes va stocker chaque superchunk, selon des critères de déduplication et de répartition de charge. Enfin, les StorageNodes sont responsables du stockage des données. L'architecture est représentée sur la figure 1.

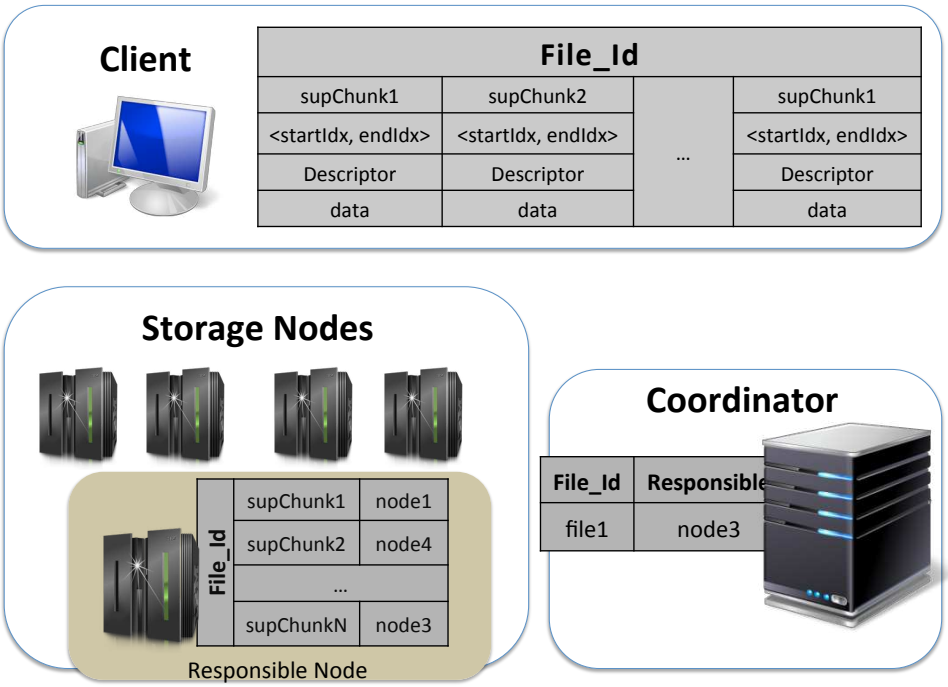


Figure 1: Architecture du Système

Dans cette thèse, nous nous concentrons sur le rôle du Coordinator, qui est chargé de gérer au mieux le compromis entre la déduplication et la repartition uniforme de la charge. À cette fin, nous proposons (i) un mécanisme resource-friendly pour la détection des duplicats qui parvient à détecter avec précision l'intersection entre les chunks inclus dans un superchunk et ceux stockés sur un StorageNode, et (ii) un nouveau mécanisme de repartition de la charge qui maintient la charge équitablement répartis entre les noeuds du système tout en conservant le taux de déduplication élevé.

Détection de Duplicats

Notre algorithme de détection de duplicats est basé sur l'observation que pour choisir le meilleur StorageNode pour un superchunk donné, le Coordinator n'a pas besoin de savoir exactement quels chunks sont stockés sur chaque StorageNode. Une estimation de l'intersection entre les chunks dans le superchunk et ceux stockés sur le StorageNode suffit pour sélectionner le meilleur (en termes d'efficacité de déduplication) StorageNode. Notre nouveau protocole calcule cette intersection en s'appuyant sur PCSA [FM85, DF03], une méthode probabiliste pour calculer la cardinalité d'un multiset, c'est à dire le nombre d'éléments distincts (chunks dans notre cas) qu'il contient.

Pour créer le descripteur d'un multiset en utilisant PCSA, nous devons d'abord d'appliquer une fonction de hachage sur les éléments que le multiset contient. Cette étape permet de rendre aléatoire les données d'entrée. Par la suite, pour chaque élément dans le multiset, on enregistre dans un vecteur bitmap la position du bit le moins significatif qui est mis à 1. L'intuition derrière PCSA est que, puisque la fonction de hachage distribue ses valeurs de manière uniforme, la position 0 du bitmap sera mis à 1 environ la moitié du temps, la position 1 sera mis en 1 1/4 de temps, et ainsi de suite. Cela conduit à la probabilité que la position k du bitmap est mis à 1 selon l' équation (1). Pour réduire l'erreur d'estimation, au lieu de garder un bitmap par multiset, nous en gardons plusieurs (8192 dans notre cas). D'après l'équation (1) nous pouvons voir que le bitmap nous donne une estimation du logarithme de la cardinalité du multiset initiale.

$$P(p(h(d)) = k) = 2^{-k-1} \tag{1}$$

Bien que PCSA ait été conçu pour estimer la cardinalité d'un multiset, il est facile de voir que le descripteur PCSA de l'union de deux multisets, A et B, peut être calculée en appliquant simplement l'opérateur OR sur les deux vecteurs bitmap correspondants, bitmap[A] et bitmap[B].

$$\text{bitmap}[A \cup B] = \text{bitmap}[A] \mid \text{bitmap}[B] \tag{2}$$

L'algorithme ci-dessus nous donne une estimation de la cardinalité de l'union des deux ensembles. Fort de cette observation, les auteurs de [MBN+06] proposent l'évaluation de la cardinalité de l'intersection de deux multi-ensembles en exprimant comme suit.

$$|A \cap B| = |A| + |B| - |A \cup B| \tag{3}$$

D'après l'équation(3), il est possible d'approximer $|A \cap B|$ simplement en estimant les trois cardinalités des trois termes à droite de l'équation, qui peuvent être estimés avec le PCSA et l'équation(2).

Répartition de la Charge

Pour maintenir la charge système équilibrée, nous introduisons une nouvelle bucket-based stratégie de repartition de la charge de stockage. Selon cette stratégie, l'espace de stockage de chaque noeud est divisé en buckets de taille fixe. Au moment de l'initialisation, le Coordinator accorde à chacun de StorageNodes la permission d'utiliser un seul de ses buckets. Quand un StorageNode remplit le dernier bucket qui lui a été accordé, le Coordinator lui en accorde un nouveau seulement dans le cas où le nombre de buckets qui lui ont déjà été accordés ne dépasse pas (de plus d'un seuil donné) le nombre de buckets alloués au noeud le moins chargé.

Résultats

Pour évaluer la performance de Produck par rapport à l'état de l'art des systèmes de deduplication en clusters, nous le comparons avec les deux solutions présentées dans [DDL+11]. Dans ce travail, les auteurs proposent une stratégie stateless qu'on appelle MinHash qui attribue les superchunks au StorageNodes en appliquant l'operateur `mod (no_de_noeuds)` sur l'empreinte minimum des chunks dans le superchunk et une stateful, qu'on appelle BloomFilter, où le Coordinator maintient une `BloomFilter` pour chaque StorageNode afin de détecter le noeud avec l'overlap le plus grand avec les chunks dans le superchunk. Les deux stratégies fixent un seuil de 5% de la charge moyenne dans le système, au-dessus de laquelle un noeud est considéré comme surchargé, et n'a donc pas le droit de stocker de nouvelles données.

Dans nos expériences, nous montrons que Produck fournit en moyenne (i) jusqu'à 18% de déduplication en plus par rapport à MinHash (stateless), et (ii) une réduction de 18 fois en coût de calcul par rapport à BloomFilter. En outre, Produck ne stocke pas plus que 64Ko pour l'état de chaque noeud de stockage tandis que l'espace de stockage nécessaire pour BloomFilter est proportionnel à la capacité des StorageNodes.

Content and Geographical Locality in User-Generated Content Sharing Systems

Passant au deuxième objectif, qui est de servir le contenu rapidement afin que les applications puissent les consomer efficacement, nous proposons DTube, une stratégie de placement de contenu qui vise à le placer près du lieu où il sera consommé. Dans cette étude, nous nous concentrons sur le contenu généré par l'utilisateur (UGC), tels que des vidéos YouTube, car il représente une fraction importante du trafic Internet et selon les analystes cette fraction va augmenter.

Pour optimiser leur performance et réduire leurs coûts d'exploitation, les sites de partage de UGC utilisent des systèmes distribués à grande échelle avec des serveurs déployés dans le monde entier dans le but de placer leur contenu proche des utilisateurs qui vont le consomer. Pour sélectionner les meilleurs noeuds, ils s'appuient généralement sur des approches proactives et réactives qui exploitent la localité spatiale et temporelle des modèles d'accès.

Au-delà des formes traditionnelles de localité qui compte pour chaque élément de contenu indépendamment, des études récentes ont montré que les habitudes de consommation de contenu sur ces sites sont nettement influencés par le fait que le contenu de ces sites n'est plus indépendant, mais est organisé dans un "content graph". Dans YouTube, le "content graph" est incarné par les listes des "Related Videos" présentes sur la page de chaque vidéo, et son influence sur le comportement des utilisateurs a été clairement documentée [KZGZ11, ZKG10].

En etudiant plus en profondeur la corrélation entre la localité de contenu (i.e. le fait que deux vidéos soient proches dans le "content graph") et localité géographique (le fait que les deux vidéos ont la plupart de leurs requêtes provenant des mêmes pays), nous montrons sur un grand ensemble de vidéos YouTube (plus de 650.000) qu'ils sont en fait liés. Pour le montrer, pour chaque vidéo, nous avons recueilli son VSV, qui est un vecteur avec les vues de la vidéo en provenance de chaque pays, et nous avons calculé le coefficient de corrélation de Spearman entre son VSV trié, classé par nombre décroissant de views, et celui de chacune de ses vidéos associées. Le coefficient de Spearman est défini dans l'équation 4 et capture la corrélation entre le rang du pays en deux VSV triés, en tenant compte des permutations de rangs. Plus la valeur absolue du coefficient se rapproche de 1, plus grande est la corrélation entre les deux listes. Les résultats de cette expérience sont présentés à la figure 2 en fonction du rang des vidéos associées dans la list de "Related Videos". Dans notre base de données, cette corrélation varie de $0,64$ à $0,68$ ce qui est relativement élevé. Cela signifie que le VSV d'une vidéo peut être déduit avec une grande précision de celui de ses vidéos associées.

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \tag{4}$$

En outre, partir de la figure 2, nous pouvons voir que plus le rang d'une vidéo dans la liste "Related Videos" est petit, plus la corrélation de son VSV avec celui de la vidéo pointant vers elle sera grande (pour la vidéo au rang 1, nous avons une corrélation de $0,68$). Cela implique que le VSV de la première vidéo reliée à chaque vidéo V peut être un bon indicateur des origines des vues de V pour l'avenir. Nous nous appuyons sur ces résultats pour proposer une stratégie de

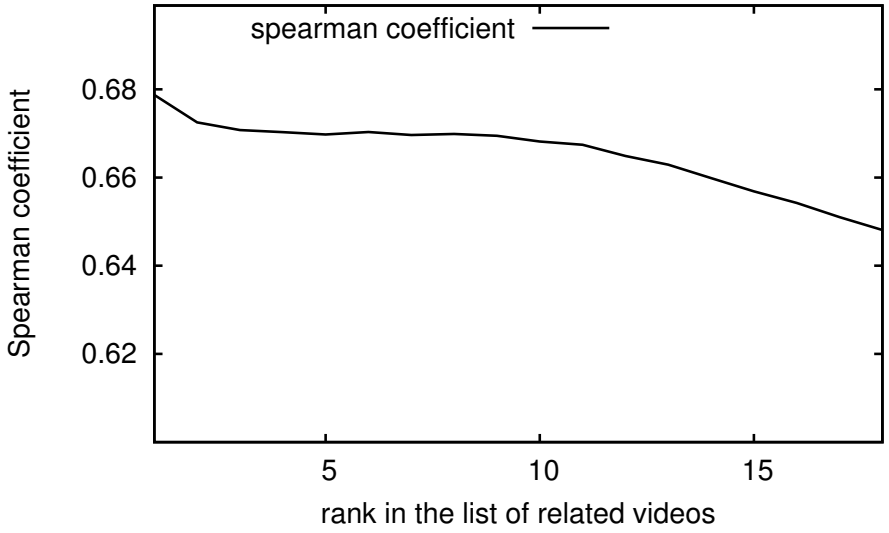


Figure 2: Spearman correlation coefficient between a video's geographic distribution of views and that of its related videos, as a function of their rank in the list of related videos

placement pour les sites de partage de contenu UGC, appelé DTube, qui place de façon proactive les vidéos proches de leurs vues futures.

Selon DTube, les répliques d'une vidéo V sont placées sur des noeuds dans les pays qui sont parmi les sources principales de vues de la première vidéo associée a V. En outre, afin de continuer à exploiter la localité du contenu, chaque réplique d'une vidéo V attire au même noeud une réplique de chacune des vidéos dans sa liste de "Related Videos".

Résultats

Nous avons évalué la performance de DTube par des simulations par rapport à la distance géographique entre les utilisateurs qui sollicitent une vidéo et les noeuds de stockage qui le servent. En outre, nous le comparons avec un système qui utilise la mise en cache réactive au-dessus du stockage persistant. Pour que les deux systèmes soient comparables, nous supposons que les deux utilisent la même quantité d'espace de stockage. Le comportement de l'utilisateur est simul selon le modè [ZKG10].

Nos résultats montrent que, pour un petit nombre de répliques par vidéo, la mise en cache est plus performante que DTube mais cela s'inverse dès que le nombre de répliques par vidéo dépasse 5. Par exemple, pour un nombre moyen de 30 répliques par vidéo, DTube diminue de 16% les demandes qui sont desservies par un noeud dans un autre pays que celui des demandes. Des résultats similaires

sont également valables pour la distance moyenne entre l'utilisateur et le noeud de stockage desservant la vidéo pour les demandes hors du pays. Avec une moyenne de 30 répliques par vidéo, DTube réduit la distance moyenne de 29%.

Perspectives

Dans cette thèse, nous nous concentrons sur l'amélioration de la performance des systèmes de stockage pour lépoque des "Big Data" et nous proposons deux systèmes qui exploitent des propriétés du contenu des donnes. Bien que ces systèmes réussissent à faire face à certains défis, il reste un certain nombre de questions ouvertes qui doivent être pris en compte. Dans cette section, nous présentons certaines de ces questions que nous considérons comme essentielles pour la sauvegarde et les systèmes de distribution de contenu.

Tolérance aux fautes dans les systèmes d'archives

Bien que la déduplication permette de réduire les besoins de stockage d'une charge de travail donnée, une autre exigence qu'un système de sauvegarde doit satisfaire est la tolérance aux pannes. La tolérance aux pannes signifie que l'intégrité des données doit être garantie contre tout danger qui la menace. Le fait que dans les systèmes de déduplication, le disque magnétique soit utilisé comme principal moyen de stockage, signifie qu'il faille faire face aux pannes que ces derniers peuvent encourir. En outre, à cause de la déduplication, un chunk de données peut appartenir à plusieurs fichiers, ce qui implique qu'une panne de disque peut corrompre plusieurs fichiers en même temps.

Cohérence dans les systèmes de stockage géographiquement distribués

Une des questions ouvertes qui est particulièrement intéressante dans le contexte du contenu interconnecté, est la cohérence des données. Le contenu des sites de partage de UGC est organisé dans un "content graph". Cela implique que les éléments de ces sites ne sont plus indépendants et, dans le contexte de la cohérence, les modifications sur un élement peuvent avoir un impact sur ses éléments adjacents dans le graphe. Étant donné que le contenu de ces sites est stocké dans des data-centers repartis partout dans le monde, conserver des réplicas cohérents devient un problème difficile et interesant.

Abstract

Cloud service providers, social networks and data-management companies are witnessing a tremendous increase in the amount of data they receive every day. All this data creates new opportunities to expand human knowledge in fields like healthcare and human behavior and improve offered services like search, recommendation, and many others. It is not by accident that many academics but also public media refer to our era as the "Big Data" era. But these huge opportunities come with the requirement for better data management systems that, on one hand, can safely accommodate this huge and constantly increasing volume of data and, on the other, serve them in a timely and useful manner so that applications can benefit from processing them. This document focuses on the above two challenges that come with "Big Data". In more detail, we study (i) backup storage systems as a means to safeguard data against a number of factors that may render them unavailable and (ii) data placement strategies on geographically distributed storage systems, with the goal to reduce the user perceived latencies and the network and storage resources are efficiently utilized. Throughout our study, data are placed in the centre of our design choices as we try to leverage content properties for both placement and efficient storage.

Résumé

Les fournisseurs de services de cloud computing, les réseaux sociaux et les entreprises de gestion des données ont assisté à une augmentation considérable du volume de données qu'ils reçoivent chaque jour. Toutes ces données créent des nouvelles opportunités pour étendre la connaissance humaine dans des domaines comme la santé, l'urbanisme et le comportement humain et permettent d'améliorer les services offerts comme la recherche, la recommandation, et bien d'autres. Ce n'est pas par accident que plusieurs universitaires mais aussi les médias publics se réfèrent à notre époque comme l' époque "Big Data". Mais ces énormes opportunités ne peuvent être éxploitées que grâce à de meilleurs systèmes de gestion de données. D'une part, ces derniers doivent accueillir en toute sécurité ce volume énorme de données et, d'autre part, être capable de les restituer rapidement afin que les applications puissent bénéficier de leur traitement. Ce document se concentre sur ces deux défis relatifs aux "Big Data". Dans notre étude, nous nous concentrons sur le stockage de sauvegarde (i) comme un moyen de protéger les données contre un certain nombre de facteurs qui peuvent les rendre indisponibles et (ii) sur le placement des données sur des systèmes de stockage répartis géographiquement, afin que les temps de latence perçue par l'utilisateur soient minimisés tout en utilisant les ressources de stockage et du réseau efficacement. Tout au long de notre étude, les données sont placées au centre de nos choix de conception dont nous essayons de tirer parti des propriétés de contenu à la fois pour le placement et le stockage efficace.