Caju: a content distribution system for edge networks

Guthemberg Silvestre, Sébastien Monnet, Ruby Krishnaswamy, Pierre Sens

Caju: a content distribution system for edge networks

Guthemberg Silvestre[1,2], Sébastien Monnet[1], Ruby Krishnaswamy[2], and Pierre Sens[1]

[1] LIP6/UPMC/CNRS/INRIA
4 place Jussieu - 75005 Paris - France
[2] Orange Labs,
38-40, rue du Général Leclerc - 92130 Issy-les-Moulineaux - France

Abstract. More and more, users store their data in the cloud. While the content is then retrieved, the retrieval has to respect quality of service (QoS) constraints. In order to reduce transfer latency, data is replicated. The idea is make data close to users and to take advantage of providers home storage. However to minimize the cost of their platform, cloud providers need to limit the amount of storage usage. This is still more crucial for big contents.
This problem is hard, the distribution of the popularity among the stored pieces of data is highly non-uniform: several pieces of data will never be accessed while others may be retrieved thousands of times. Thus, the trade-off between storage usage and QoS of data retrieval has to take into account the data popularity.
This paper presents our architecture gathering several storage domains composed of small-sized datacenters and edge devices; and it shows the importance of adapting the replication degree to data popularity.
Our simulations, using realistic workloads, show that a simple cache mechanism provides a eight-fold decrease in the number of SLA violations, requires up to 10 times less of storage capacity for replicas, and reduces aggregate bandwidth and number of flows by half.

1 Introduction

Content distribution over the internet has increased dramatically in the recent years. A recent study published by Cisco System, Inc [2] revealed that the global internet video traffic has surpassed peer-to-peer traffic since 2010, becoming the largest internet traffic type. Cisco Systems also forecasts that internet video traffic will reach 62% of the consumer internet traffic by 2015. The vast majority of this traffic consists of big popular content transport, including high-quality videos.
Nowadays, a large amount of data is stored "in the network". This allows users to ease data sharing and retrieval, anywhere in the world. In the cloud, customers and providers come with storage service guarantees, such as QoS metrics, drawn up in Service Level Agreement (SLA) contracts. The provider is therefore responsible to ensure data durability and availability. To enforce SLAs, providers rely on content replication. Yet, they need to do this carefully, it can have a huge impact on storage and bandwidth consumptions, even more for big contents. As data

popularity is highly non-uniform, it is important to avoid replicating unpopular data, that will never be accessed, and also to ensure enough number of replicas for a popular content which may be retrieved concurrently by hundreds of users. This work introduces and evaluates Caju, a content distribution system for edge networks. We analyse the performance of Caju as infrastructure for offering elastic storage cloud to users. We assume that cloud users may be eager to watch high-quality videos on-demand, on which strict SLA contracts have to be enforced. We study the impact of adding strict data transfer rates, as the main QoS metric, for SLA contracts in the cloud. We evaluate through simulations two replication schemes with synthetic traces that fairly reproduces big data requests, including popularity growth.
This work makes two main contributions:

- We describe the design, model, and implementation of Caju, a content distribution system for edge networks, that provides simple replication mechanisms, and allow us to manage edge resources properly.
- We evaluate the impact of big popular content on replication schemes in order to provide elastic storage to cloud users with regard to strict SLA contracts.

The rest of this work is organized as follows. Section 2 covers some background of the today's content distribution systems and related work. Section 3 presents our approach to tackle elastic storage provision on the edge of the network, and provides an in-depth description of Caju, our system for CDNs at edge-networks. Section 4 analyses and explains our evaluation scenario and performance results. Finally, Section 5 shows future work and concludes the paper.

2 Background and State of the art

We first describe the current scenario of content distribution networks and the role of edge networks in the content distribution. Then, we focus on replication systems used by cloud and P2P storage systems.
Content distribution networks and edge networks: Content distributions networks (CDN) are distributed systems that maintain content servers in many different locations in order to improve content dissemination efficiency, enhance QoS metrics for end-users, and reduce network load. There are two types of servers in CDN compositions: origin and replica servers (so-called surrogate servers) [6]. We can therefore differentiate CDNs on the basis of their surrogate servers placement, and classify them into core and edge architectures. Core CDN architectures rely on the deployment of private datacenters close to ISP points of presence (PoP), and it has been a successful approach used by most of the big DCN infrastructure providers, including Akamai [5]. Since this approach uses private resources deployment, and are not designed for cooperating with other CDNs, they require huge amounts of money for deployment and maintenance. Yet as core architectures are connected to PoPs, they do not have control of traffic throughout ISP until the end-customer that undermines QoS guarantees enforcement. Interoperable CDNs in edge network have emerged to tackle directly these issues. Network service providers look forward to (i) take advantage of their infrastructure, (ii) deploy their own datacenters, and (iii) deliver content as close as possible

to end-customer. The aim is to be able to offer differentiated QoS guarantees to regular customers [8]. In this work, we focus on challenges risen by edge CDN architectures. In particular, we have studied how to organize consumer-edge devices to cooperate with small-sized datacenters, and how to enforce different classes of strict SLA contracts at the edge of the network.

Replication schemes: Network providers can rely on replication schemes for enhancing disseminating content efficiency. Considering resource allocation strategies, we are mostly interested in two categories of replication schemes: uniform, and adaptive replication schemes. The Google File System (GFS) [4] and Ceph [9] adopt a pragmatic approach where the number of replicas is uniform, that mean a fixed number of replicas per stored object. This trivial and primitive approach has had a considerable success in the industry, particularly for datacenters deployment, because it is easy to adopt. However it relies on over-provision to provide resource allocation for popular content, and despite of using commodity servers, it is inefficient and quite expensive. Overall, these issues are addresses by adaptive replication schemes. For instance, the use of non-collaborative LRU caching allows us to easily adapt the replication degree of an content according to its demand. More sophisticated adaptive schemes, such as EAD [7] and Skute [1] tackle content replication by using a cost-benefit approach over decentralized and structured P2P systems. EAD creates and deletes replicas throughout the query path with regards to object hit rate using an exponential moving average technique. Skute provides a replication management scheme that evaluates replicas price and revenue across different geographic locations. Its evaluation technique relies on equilibrium analysis of data placement. Despite being highly scalable and providing an efficient framework for replication in distributed systems, these approaches result in inaccurate transfer rate allocations, hence they are inappropriate for high-quality content delivery.

3 Approach

3.1 Caju's design

We introduce a simple content distribution system, called Caju to study adaptive replication schemes at the edge of the network. Its design is depicted in Figure 1. We assume that the service provider infrastructure is organized in federated storage domains. A storage domain is a logical entity that aggregates a set of storage elements that are located close to each other, e.g. connected to a digital subscriber line access multiplexer (DSLAM). The storage elements are partitioned in two different classes: (i) operator-edge elements, furnished by storage operators, e.g. small-sized datacenters, and (ii) consumer-edge entities provided by consumers, such as set-top boxes. Consumer-edge devices contribute to storage and network resources according to their availability and load. Operator-edge nodes run a distributed storage system for local-area network over commodity servers. They provide cheap and high available resources dedicated to the storage service.

On top of each Storage Domain runs a couple of services that deals with serving clients' requests, and performs appropriate object data placement and replication.

The main functional blocks are depicted in Figure 2. Remote storage clients contact the coordinator when they need perform any request over an object (PUTs and GETS). The coordinator maintains a catalogue of all clients and available resources, it is also responsible for scheduling the requests. On behalf of the clients, it selects proper resources for fulfilling their SLA contracts based on replications schemes.

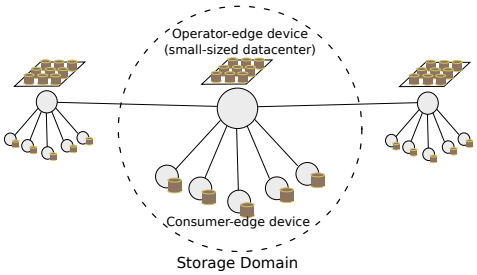

Fig. 1. Storage Elements (SEs) and Storage Domains (SDs)

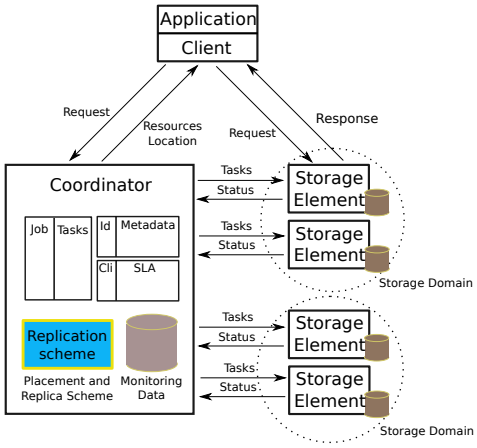

Fig. 2. The main functional blocks.

3.2 System model

Here, we formally describe the target storage systems' main components, interactions, and constraints. We also provide guidelines on our performance goals for the evaluation Section 4.

Object store service and client satisfaction Our target system provides a distributed object store service for clients. We denote the set of all possible client's objects as $\mathcal{O}$. We consider that objects comprise a set of data blocks of fixed size, K_C, called chunks. So that, an object $o \in \mathcal{O}$ of size z_o has $\frac{z_o}{K_C}$ chunks.
We consider that M clients are able to do any number of requests R_M to the system. There are three types of client request: *get* any object, *put* new objects it into the system, and *delete* their own objects. The storage system provides a fourth system request type in order to *replicate* an object. It also might perform *deletions*, a fifth request type, for maintenance or control purposes.
We assume that clients are eager for quality of storage service, and their wills are formally defined by SLA contracts. SLAs allow a client m to choose a suitable rate of chunks per request λ_s and minimum acceptable percentage of successful requests P_s. Assuming a period of request analysis T, we consider that a client m who did r'_m requests is satisfied with the store service if at least $r'_m x P_s$ requests were accomplished with λ_s rate. The object store service places and replicates objects throughout the system with regard to the client satisfaction.

Storage at the edge of network providers We consider a distributed storage system deployed at the edge of network providers, that is organized in storage domains. We assume that there exists I storage domains. A storage domain i, $i \in \{1, 2, \ldots, I\}$ has storage capacity of S_i and throughput T_i. Each storage domain has a set $\mathcal{J}_i$ of J storage elements, $j \in \{1, 2, \ldots, J\}$, partitioned in two distinct classes: $\mathcal{C}_o$ for operator-edge class, and $\mathcal{C}_c$ for consumer-edge class, where $|\mathcal{C}_c| \gg |\mathcal{C}_o|$. The storage capacity of storage element j is denoted by:

$$s_{ij} = \begin{cases} D_o \text{ if } j \in \mathcal{C}_o; \\ D_c \text{ if } j \in \mathcal{C}_c. \end{cases} \tag{1}$$

where D_o and D_c are maximum storage capacity parameters. Hence,

$$S_i = \sum_{j=1}^{J} s_{ij} \qquad i \in \{1, 2, \ldots, I\} \tag{2}$$

The storage element bandwidth capacity is denoted by:

$$b_{ij} = \begin{cases} W_o \text{ if } j \in \mathcal{C}_o; \\ W_c \text{ if } j \in \mathcal{C}_c. \end{cases} \tag{3}$$

where W_o and W_c are maximum bandwidth capacity parameters. We assume that any storage element has a full-duplex, symmetric connection links. Moreover, b^u_{ij} denotes the instantaneous bandwidth consumption of storage element j of i. In a same storage domain i, if j and j' are two storage element from different classes, and there is not active transfer between them, their respective b^u_{ij} do not interfere with each other. Despite that, we consider that network infrastructure imposes the following condition (4) on the maximum throughput of a set of consumer-end storage elements of i:

$$\sum_{j \in \mathcal{C}_c} b^u_{ij} \leq W_l \tag{4}$$

where W_l is a the maximum aggregated bandwidth consumption for a set of consumer-edge devices that the network provider infrastructure permits. Considering inequality (4), the maximum throughput of a storage domain i is denoted as follows:

$$T_i = \frac{1}{K_C}(\sum_j b_{ij}) \leq \frac{1}{K_c}(W_l + |\mathcal{C}_o| W_o) \tag{5}$$

where K_C is the chunk size parameter.

System interactions and performance goals Each client m is connected, through its own consumer-edge storage element j, with a single domain i, called home storage domain. Any m belongs to a SLA class. The system might have one or many SLA classes, such that different levels of quality of service might

be provided. As described here above, SLA's constraints allow clients to choose a suitable rate of chunks per request λ_s and minimum acceptable percentage of successful requests P_s, and that the client satisfaction depends on these parameters.

The system allows clients to do storage requests towards their own homes only. However, all requests might be served by storage elements from any federated storage domain, except for objects' insertions, that must be served by client's home storage domain. For mapping and monitoring resources, and interactions in our storage system, we assume that there exists logically centralized coordinator. The performance and design issues of coordinator are beyond the scope of our current work.

We denote the set of all R possible requests by $\mathcal{R}$. Requests are grouped in two distinct manners: by requester or by type of request. In terms of requester, there are two disjoint subsets: $\mathcal{R}_M$ for client's requests, and $\mathcal{R}_S$ for own storage system's requests. When our system receives a request r that requires to move objects between any node and storage element of i, it serves this request by creating data transfers from a source to a destination. As described above, a requester might be either a client or the own system. If $r \in \mathcal{R}_S$ the transfer is always made between two storage elements. For all $r \in \mathcal{R}$, let:

$$p_{j,r} = \begin{cases} 1, \text{ if } j \text{ serves } r; \\ 0, \text{ otherwise.} \end{cases} \tag{6}$$

be a 0-1 variable indicating if the storage element $j \in \mathcal{J}_i$ provides resources to serve request r. We assume there is a function $A^i_j(t)$ that yields the current available rate of chunks by j in t for serving a system incoming request. Therefore, if client m requests r^m over a storage domain i in time t, asking for a λ^s_m rate, the storage system fulfil m's expectations if and only if:

$$\textbf{Constraint 1: } \sum_{j \in \mathcal{J}_i} p^m_{j,r} \cdot A^i_j(t) \geq \lambda_s \tag{7}$$

Therefore our system performance goals for our replication scheme are twofold. Firstly, we aim to maximize the number of satisfied clients as a metric for evaluating the quality of provided service. And secondly, we tend to minimizing the amount of system's bandwidth and storage usage, by adjusting properly the resource allocation over storage elements in order to serve $\mathcal{R}$.

4 Evaluation

Our evaluation has two main goals: (i) to evaluate the performance of Caju in providing storage for cloud users on top of edge devices, including operator-edge devices, so-called small-sized datacenters; (ii) to compare and evaluate challenges of two replication schemes: uniform with fixed number of replicas and non-collaborative caching.

The evaluation scenario (Figure 3) includes 2002 (numbered) nodes arranged across two Storage Domains (SD). There are one operator-edge device (nodes

1 and 1001) and 1000 consumer-edge devices per Storage Domain. Storage and network capacities differ accordingly to the class of device. Each operator-edge device has 10TB of storage capacity and 4Gbps as network capacity. Consumer-edge devices contribute with a smaller storage capacity per device, 100GB, and are equipped with a full-duplex access link of 100Mbps per consumer device. We consider that consumer-edge devices that belong to the same Storage Domain are geographically close to each other, and that a maximum bandwidth limit of 80% is enforced to aggregated traffic of consumer-edge devices on edge network level. The workload was carefully set-up to match to multimedia popular content distribution, as described in recent studies [3]. Tables 1 and 2 list default values for evaluation scenario and workload parameters respectively. SLA contracts differ to each other by transfer rate λ_s. Thus, we consider three SLA classes, in chunks per second: (a) 41, (b) 21, and (c) 14 chunks/s. To each customer is assigned a SLA that regards the following distribution: 40% class (a), 40% to (b), and the remaining 20% to (c). We assume that a SLA violation occurs when any transfer of a consumer does not observe her minimum contracted transfer rate.
We use *happiness* or number of customers without SLA violations as a key performance metric. This means P_s is equal to 100% in our model. Along with *happiness*. We also focus on number of SLA violations, number of flows, storage and network capacity usage.
The rest of this section is structured as follows. Subsection 4.1 describes the two replication schemes evaluated in this work. Then, we show our performance analysis for these two schemes in Subsection 4.2.

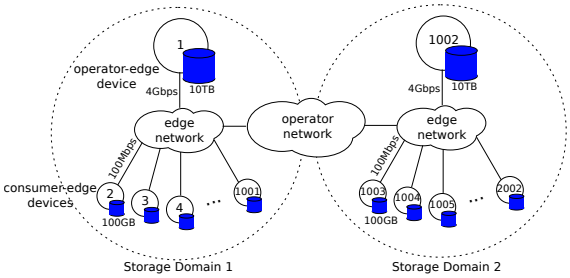

Fig. 3. Evaluation scenario

4.1 Evaluated replication schemes

We have evaluated the performance of two replication schemes with Caju:
Uniform replication scheme with fixed number of replicas This is the simplest approach to replicate objects into a system, that is broadly used in current datacenter deployments. Given a fixed number of replicas n as a parameter, we simulate a chain of object-replication of n stages just after the initial insertion (PUT). Requests are scheduled in order to balance load throughout nodes. Each request might be served by at most R nodes with equal load. The actual number of sources is $r = min(n, R)$.
Non-collaborative LRU caching Simple adaptive replication schemes based on non-collaborative caching, such as those that implements Least Recent Used

Table 1. Default parameter values for the evaluation system

Evaluation scenario	
Number of Storage Domains (SD)	2
Number of Operator-edge devices (OE) per SD	1
Number of Consumer-edge device (CE) (with a home client) per SD	1000
OE storage capacity	10TB
CE storage capacity	100GB
OE network capacity	4Gbps
CE network capacity	100Mbps
Aggregate bandwidth limit for a set of CEs	80%
Chunk size	2MB
Number replicas	2
Maximum parallel flows per request	5

Table 2. Default values for workload parameters

Workload	
Requests per client	uniform
Experiment duration	1h 12min
Object size (follows Pareto)	shape=5 lower bound=70MB upper bound=1GB (mean 93MB)
Mean requests per second	50
Requests division	5% for PUTs 95% for GETs
Popularity growth (follows Weibull)	shape=2 scale $\propto$ duration
Content popularity (Zipf-Mandelbrot)	shape=0.8 cutoff=# of objects
PUTs (Poisson)	λ=PUTs/s

algorithm, are straightforward and easy to implement and deploy. In our implementation, a new replica is created whenever a client, connected to a operator-edge device, performs a GET to any object. LRU replacement is enforced regarding a static percentage of the local storage capacity γ for caching. Request scheduling is quite similar to that of uniform approach. However the number of available sources n changes according to LRU algorithm.

4.2 Performing storage for cloud users at the edge of the network

We analyse the efficiency of delivering popular content with strict SLA definitions using two replication schemes approaches: uniform replication with a fixed number of replicas, and non-collaborative LRU caching.

First, we have evaluated the required number of replicas of uniform replication for different request rates in order to prevent SLA violations. Figure 4 shows *happiness* metric for mean request rates of 50, 100, 150, and 200 requests per second. We have observed that uniform replication schemes require high replication degree in order to cope with strict SLA definitions and popular content. At least 20 replicas are required to prevent violations if the request rate is as high as 150 requests per second. For the highest request rates, uniform replica is not suitable. When we simulated 200 requests per second, there were 799 violations. Despite having being widely used in datacenters and storage clusters, uniform replication scheme relies on over-provision in order to distribute popular content with strict SLA definitions, hence it is not fit for edge network deployments.

To avoid over-provision, we have analysed the storage usage uniform replication with a non-collaborative LRU caching. We simulate different LRU caching sizes

percentages: 1%, 5%, and 10% of the storage capacity. Figure 5 plots storage storage usage and *happiness* metric for 200 requests per second. Even with the smallest cache storage percentage of 1%, a non-collaborative LRU caching approach performs much better than uniform replication. We observed 89 violations with 1% of non-collaborative LRU caching that required a storage usage, 7.82TB, similar to uniform scheme with 2 replicas, 8.92TB.

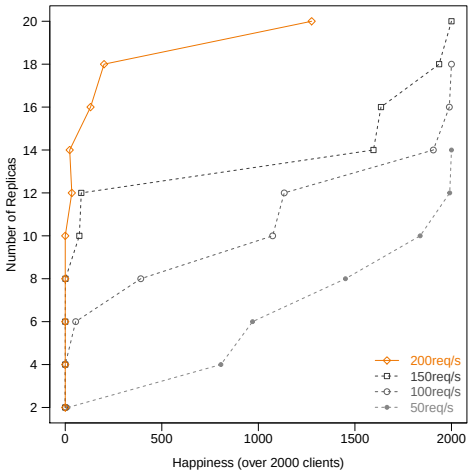

Fig. 4. Happiness metric with uniform replication scheme.

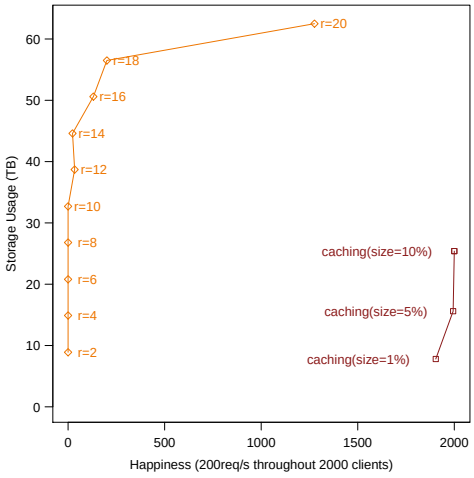

Fig. 5. Storage usage for uniform and caching replication schemes.

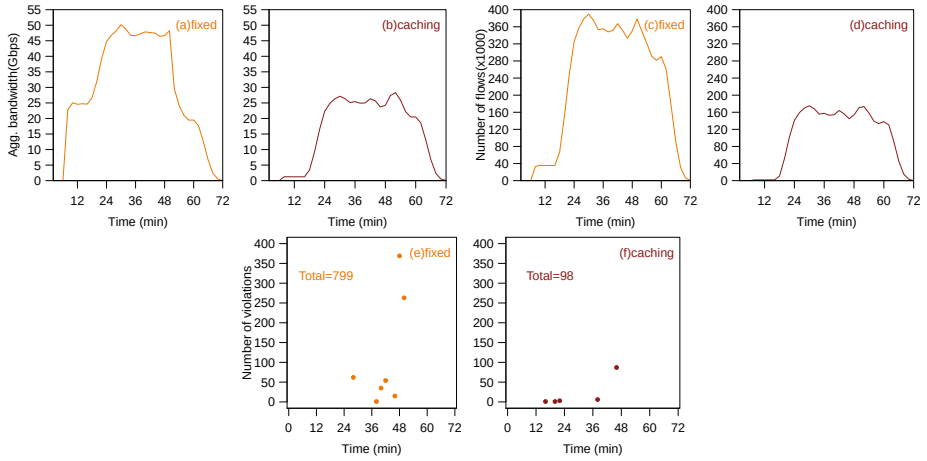

Fig. 6. Aggregate bandwidth (a,b), number of flows (c,d) andnumber of SLA violations (e,f) using fixed number of replicas and caching.

In order to gather more information about the advantages of using non-collaborative caching for distributing popular content instead of uniform replication, we have evaluated and plotted in Figure 6 the throughput, flows, and violations results sampled per second. We selected results from LRU caching with

local cache size of 1% of the node storage capacity, and uniform replication with 20 replicas. By using a non-collaborative LRU caching, we have seen that the number of flows and aggregate bandwidth was reduced by half. We have also verified that the number of violation slashed from 799 to only 98.

5 Conclusions and perspectives

Online storage of big data becomes very popular. Storage providers need to find good trade-offs between replication and storage usage. In this paper, we show that it is important to take content popularity into account: using a fixed replication would lead either to waste storage space or to increase the number of unsatisfied customers. We propose and evaluate Caju, a content distribution system for edge networks. Caju provides the ability to manage storage and network resources from both consumer and operators in a collaborative manner. Our evaluations show that non-collaborative caching consistently outperforms the fixed replication scheme. It provides a eight-fold decrease in the number of SLA violations, requires up to 10 times less of storage capacity for replicas, and reduces aggregate bandwidth and number of flows by half. We are working on the design of new adaptive placement and replication algorithms. Our goal is to enhance non-collaborative caching for popular content delivery.

References

1. Nicolas Bonvin, Thanasis G. Papaioannou, and Karl Aberer. A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In ACM, editor, *ACM Symposium on Cloud Computing 2010 (SOCC2010)*, Indianapolis, USA, June 2010.
2. Cisco visual networking index: Forecast and methodology, 2010-2015. http://www.cisco.com, June 2011.
3. Flavio Figueiredo, Fabrício Benevenuto, and Jussara M. Almeida. The tube over time: characterizing popularity growth of youtube videos. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 745–754, New York, NY, USA, 2011. ACM.
4. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP '03: Proceedings of the 9th ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, October 2003. ACM Press.
5. Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.
6. Mukaddim Pathan and Rajkumar Buyya. A taxonomy of cdns. In Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali, editors, *Content Delivery Networks*, pages 33–77. Springer Berlin Heidelberg, 2008.
7. Haiying Shen. An efficient and adaptive decentralized file replication algorithm in p2p file sharing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 21:827–840, June 2010.
8. Enabling digital media content delivery: Emerging opportunities for network service providers. http://www.velocix.com/formwp.php, March 2010.
9. Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*, November 2006.