A Mathematical Programming Approach to Multi-cloud Storage

Makhlouf Hadji
Technological Research Institute SystemX, Palaiseau, Saclay, France
{*makhlouf.hadji*}*@irt-systemx.fr*

Keywords: Cloud Computing, Distributed Storage, Data Replication, Encryption, Broker, Optimization.

Abstract: This paper addresses encrypted data storage in multi-cloud environments. New mathematical models and algorithms are introduced to place and replicate encrypted data chunks and ensure high availability of the data. To enhance data availability, we present two cost-efficient algorithms based on a complete description of a linear programming approach of the multi-cloud storage problem. Performance assessment results, using simulations, show the scalability and cost-efficiency of the proposed multi-cloud distributed storage solutions.

1 INTRODUCTION

Cloud storage has emerged as a new paradigm to host user and enterprize data in cloud providers and data centers. Cloud storage providers (such as Amazon, Google, etc.) store large amounts of data and various distributed applications (AWS, 2014) with differentiated prices. Amazon provides for example storage services at a fraction of a dollar per Terabyte per month (AWS, 2014)). Cloud service providers propose also different SLAs in their storage offers. These SLAs reflect the different cost of proposed availability guarantees. End-users interested in more reliable SLAs, must pay more, and this leads to cause high costs when storing large amounts of data. The cloud storage providers to attract users do not charge for initial storage or put operations. Retrieval becomes unfortunately a hurdle, a costly process and users are likely to experience data availability problems. A way to avoid unavailability of data is to rely on multiple providers by replicating the data and actually chunk the data and distribute it across the providers so none of them can actually reconstruct the data to protect it from any misuse. This paper aims at improving this type of distributed storage across multiple providers to achieve high availability at reasonable (minimum) storage service costs by proposing new scalable and efficient algorithms to select providers for distributed storage. The objective is to optimally replicate data chunks and store the replicates in a distributed fashion across the providers. In order to protect the data even further, the chunks are encrypted.

1.1 Paper Contributions and Structure

We propose data chunk placement algorithms to tradeoff data availability and storage cost and provide some guarantees on the performance of the distributed storage. We assume end-users involved in PUT (write) and GET (read) operations of data objects stored in an encrypted manner and distributed optimally in different data centers require a specified level of data availability during data retrieval. More specifically, after data encryption and partition operations which consist to split the data into encrypted chunks to be distributed across multiple data centers, our main work focuses on improving and optimizing two operations:

- **Data Chunks Placement Optimization:** through a novel, efficient, scalable algorithm that minimizes placement cost and meets data availability requirements given probabilities of failure (or unavailability) of the storage systems and hence the stored data.
- **Chunk Replication:** to meet a required high level of availability of the data using optimal replication of chunks to reduce the risk of inaccessibility of the data due to data center failures (or storage service degradations).

To realize these objectives, we derive a number of mathematical models to be used by a broker (real or logical broker) to select the storage service providers leading to cost-efficient and reliable data storage. The proposed broker collaborates with the providers having different storage costs and reliability (storage service availability), as depicted in detail in Figure 1.

We assume that the providers propose storage services to the broker and to end-users with same reliability but with different prices (prices for a real broker for instance will be lower than those proposed to end-users).

It is consequently assumed that there exist benefits for a storage service brokerage that optimally distributes encrypted data across the most appropriate providers. Thus, the aim of this paper consists to propose a scalable and polynomial algorithm spanning a cost efficient chunk placement model that can achieve optimal solutions, when guaranteeing high data availability to end-users.

Section 2 presents related work on cloud storage and optimization. In Section 3, we use the well known Advanced Encryption Standard (AES) algorithm (Seungmin et al., 2014) to encrypt end-user data and divide them into $|\mathcal{N}|$ chunks. In the same section, we propose mathematical models to deal with chunk placement and replication in an optimal manner for given server costs and availabilities. Performance assessments and results are reported in Section 4. Conclusion and future work are reported in Section 5.

2 RELATED WORK

Data storage and data replication has received a lot of attention at the data management, distribution and application level since the distribution of original data objects and their replicas is crucial to overall system performance, especially in the cloud environment where data are supposed to be protected and highly available in different data centers. The current literature concerns essentially the cloud storage problem in tandem with replication techniques to improve data availability, but to our knowledge, does not consider data transfer in/out costs, or migration costs, etc. We will nevertheless cite some of the related work even if it can not be directly compared to the proposed algorithms in this paper.

In (Mansouri et al., 2013), authors dealt with the problem of multi-cloud storage with a focus on availability and cost criteria. The authors proposed a first algorithm to minimize replication cost and maximize expected availability of objects. The second algorithm has the same objective subject to budget constraints. However, this paper did not embed security aspects apart from dividing the data into chunks or objects. In our work, we propose to divide data into encrypted chunks, that will be optimally stored and distributed through various data centers with minimum costs while satisfying the QoS required by end-users. Moreover, the proposed algorithm in (Mansouri et al., 2013) is a simple heuristic without any convergence guarantee to the optimal solution. Our proposed algorithm converges in few seconds to optimal solutions benchmarked by the Bin-Packing algorithm.

In (Thanasis et al., 2012), authors present Scalia, a system to deal with the problem of multi cloud data storage under availability and durability requirements and constraints. The authors note the NP-Hardness of the considered problem, and propose algorithms to solve small instances of the problem. In our work, we propose a new efficient and scalable solution capable of handling large instances in a few seconds. Clearly, the proposed solution in (Thanasis et al., 2012) suffers from scalability challenges to handle on with larger instances, when our algorithms are able to quickly solve large instances of the defined problem.

To avoid failure and achieve higher availability when storing data in the cloud, reference (Yanzhen and Naixue, 2012) proposes a distributed algorithm to better replicate data objects in different virtual nodes instantiated in physical servers. According to the traffic load of all considered nodes, the authors considered three decisions or actions as replicate, migrate, or suicide to better meet end-user requirements and requests. However, the proposed approach consists only in checking the feasibility of migrating a virtual node, performs suicide actions or replicating a copy of a virtual node, without optimizing the system. In our work, we propose optimization algorithms based on a complete description of the convexe hull of the defined problem, leading to reach optimal solutions even for large instances.

Reference (Srivastava et al., 2012) proposes a simple heuristic to give stored data greater protection and higher availability by splitting a file (data) into subfiles to be placed in different virtual machines belonging to the physical resources (data centers for example) of one provider or different providers. The paper dealt with PUT and GET operations to distribute and retrieve the required subfiles (data) without encrypting them. The proposed heuristic in (Srivastava et al., 2012) can only reach suboptimal solutions, leading to considerable gaps compared to the optimal solutions. We propose a new scalable and cost efficient solution to deal with the multi-cloud storage problem.

Aiming to provide cost-effective availability and improve performance and load balancing of cloud storage, the authors of reference (Qingsong et al., 2010) propose CDRM as a cost-effective dynamic replication management scheme. CDRM consists in maintaining a minimal number of replica for a given availability requirement, and proposes a replica placement based on the blocking probability of data nodes. Moreover, CDRM allows us to dynamically adjust the

replica number according to changing workload and node capacities. However, the paper focuses only on the relationship between availability and replica number, and there is no proposal to deal with the optimal placement of replicas.

To achieve high performance and reduce data loss when we require storage services in the cloud, different papers in the literature propose various algorithms that are useful only for small instances due to the NP-Hardness of the problem. In (Bonvin et al., 2010), the authors propose a key-value store named Skute, which consists in dynamically allocating the resources of a data cloud to several applications in a cost effective and fair way using game theoretical models. To guarantee cloud object storage performance, the authors of (Jindarak and Uthayopas, 2012) propose a dynamic replication scheme to enhance the workload distribution of cloud storage systems. The authors of (Chia-Wei et al., 2012) conduct a study based on a dynamic programming approach, to deal with the problem of selecting cloud providers offering storage services with different costs and failure probabilities.

Reference (Abu-Libdeh et al., 2010) proposes a distributed storage solution named RACS, to avoid vendor lock-in, reduce the cost of switching providers, and better tolerate provider outages. The authors applied erasure coding (see references (Weatherspoon and Kubiatowicz, 2002), (Li and Li, 2013) and (Rodrigo and Liskov, 2005)) to design the proposed solution RACS. In the same spirit, references (Ford et al., 2010), (Myint and Thu, 2011), (Negru et al., 2013) and (Zhang et al., 2012) addressed the cloud storage problem described above, under different constraints including energy consumption, budget limitation, limited storage capacities, and the availability of the stored data.

In (Varghese and Bose, 2013), authors propose a new solution to guarantee the data integrity when stored in a cloud data center. The proposed solution is based on homomorphic verifiable response and hash index hierarchy. This kind of solutions can be integrated to our work to reenforce data security and privacy for reticent users. An other reference on secured multi cloud storage can be found in (Balasaraswathi and Manikandan, 2014). Authors presented a cryptographic data splitting with dynamic approach for securing information. The splitting approach of the proposed solution is not deeply studied. This may lead to not select cost efficient providers.

3 SYSTEM MODEL

To store encrypted data in multiple DCs belonging to various cloud providers system, while optimizing storage costs and failure probabilities, we separate the global problem into a number of combinatorial optimization sub-problems. To derive the model we make a simplifying assumption regarding the pricing scheme between cloud service providers, the broker and end-users. We assume that the proposed storage price by a service cloud provider to end-users is higher than that proposed to the broker. This can be explained by the large amount of demands that will be required by the broker aggregating the demands of a finite set of end-users seeking to avoid vendor lock-in and higher availability. One can assume that prices proposed by cloud providers are smaller as the volume of data is larger. Note that the broker will guarantee a minimum storage cost meeting end-users requirements, ensuring that the proposed cost to end-users can never exceed a certain threshold.

We first propose to use the well known AES (Advanced Encryption Standard) algorithm (Seungmin et al., 2014) for efficient data encryption. This will generate different encrypted chunks to be distributed in the available storage nodes or data centers. This encryption ensures the confidentiality of the stored data. Moreover, the used solution permits to construct diverse chunks (with small sizes) to facilitate PUT and GET requests as is shown in Figures 1 and 2.

We derive two algorithms to handle encrypted data chunk placement and replication to guarantee data high availability, and storage cost efficiency. This can be summarized as follows:

- **Data Chunk Placement:** The first important objective of our paper consists in guaranteeing the availability of all chunks of stored data by optimally distributing them to a cost-efficient set of selected data centers (see Figure 1). This avoids user lock-in, and reduces the total cost of the storage service. This optimization is performed under end-user or data owner constraints and requirements such as the choice of a minimum number of data centers to be involved in storing the chunks of the data. This can reinforce the availability of data for given data centers failure probabilities.
- **Data Chunk Replication:** After optimally storing the encrypted chunks of a data, we determine a replication algorithm based on bipartite graph theory, to derive optimal solutions of the problem of storing replica chunks. This ensures high data availability since content can be retrieved even if some servers or data centers are not available.

Once all data chunk are placed in different data

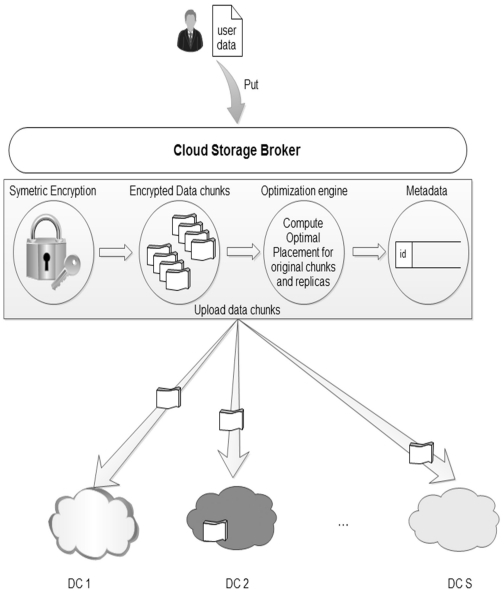

Figure 1: The system model: PUT requests.

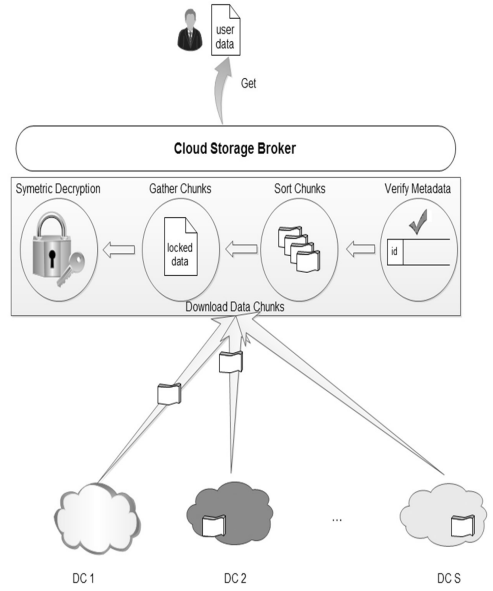

Figure 2: The system model: GET requests.

centers, end-users may solicit the data by GET requests (download data). The broker needs to gather all the data chunks, sort them, decrypt them, and finally deliver the entire data to the end-user. Figure 2 gives more details on GET operations.

In the following, we suppose that each data object (chunk) has r replicas. Finding the optimal number of replicas of each chunk, is not in the scope of this paper. A well-known example on the choice of r is the Google storage solution based on $r = 3$ replicas of each stored data chunk (Ghemawat et al., 2003).

3.1 Data Encryption Algorithm

While consumers have been willing to trade privacy for the convenience of cloud storage services, this is not the case for enterprises and government organizations. To achieve high data security and privacy, we propose to divide the requested user data to store into encrypted chunks. This facilitates PUT and GET requests by considering small files (chunks), and reinforces the security of data (thanks to the encryption) in the same time.

To preserve the confidentiality of data, we seek algorithms that can encrypt and decrypt multiple chunks in a small time. To deal with this problem, we propose to use the symmetric encryption algorithm noted AES for Advanced Encryption Standard (Seungmin et al., 2014). The AES algorithm is a fast solution to handle with large amount of data as it is shown in Figure 3 where three different keys (128 bits, 192 bits and 256 bits) are used to encrypt and decrypt data sizes ranging from 1 Megabyte to 4 Gigabytes in a time interval ranging from 200 seconds to 800 seconds.

The key sizes are chosen by end-users depending on the privacy level of their data. In our proposal, we suppose that the broker proposes an encryption solution in which generated private keys are well kept within end-users with a key size of 128 bits.

Note that more details on the encryption/decrytpion algorithms used in this paper, can be found in the literature (see for example (Seungmin et al., 2014) and (NIST, 2014)). A deep study of these solutions is not in the scope of this paper.

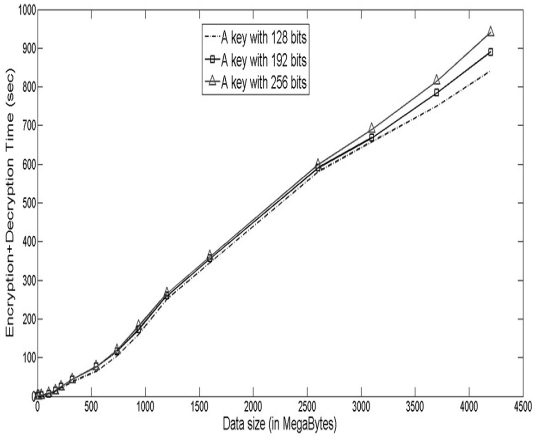

Figure 3: Encryption and decryption's time evolution with data size.

3.2 Data Placement Cost Minimization: b-Matching Formulation

We start data chunks placement model by considering each data D of a user u, as a set of chunks (noted by $\mathcal{N}$), resulting from the AES algorithm. Let S be the set of all available data centers able to host and store end-user data. We investigate an optimal placement by storing all the chunks in the "best" available data centers. Each cloud provider with a data center $s \in S$ proposes a storage cost per Gigabyte and per month noted by μ_s. This price varies for different reasons: varying demands and workloads, data center reliability, geographical constraints, etc. End-user requests are submitted to the broker which will relay them to cloud service providers, in an encrypted form with optimized storage costs. The broker guarantees end-users high data availability with minimum cost by choosing a set of cloud providers (or DCs) meeting the requirements (see Figure 1 for more details).

In the following, we will address chunks placement optimization model based on different constraints as the probability of failure of a data center or a provider, and a limited storage capacity. Each data center (or provider) has a probability of data availability (according to the number of nines in the proposed SLA), and a failure probability (f) is then equal to $1-$probability of data availability. Moreover, the limited storage capacity is given by a storage quota proposed by the provider to the broker according to a negotiated pricing menue.

Our optimization problem is similar to a classical Bin-Packing formulation, in which bins can be represented by the different Data Centers, and the items can be seen as the data chunks. Reference (Korte and Vygen, 2001) has shown a while ago the NP-Hardness of the Bin-Packing problem. Thus, we deduce the complexity (NP-Hardness) of our chunks' placement problem.

For this reason, and the fact that workloads and requests to store date arrive overtime, the broker seeks a dynamic chunk placement solution that will be regularly and rapidly updated to remain cost-effective and ensure data high availability.

Each data chunk $i \in \mathcal{N}$ has a certain volume noted by v_i. We graphically represent the storage of a chunk i in a data center k as an edge $e = (i,k)$ (with the initial extremity ($i = I(e)$) of e corresponding to a chunk, and the terminal extremity ($k = T(e)$) of e) representing the data center (see Figure 4).

Based on this configuration, one can construct a new weighted bipartite graph $G = (\mathcal{N} \cup \mathcal{S}, E)$, where $\mathcal{N}$ is the set of vertices representing encrypted chunks to be stored, and $\mathcal{S}$ is the set of all available data centers (see Figure 4). E is the set of weighted edges between $\mathcal{N}$ and $\mathcal{S}$ constructed as described:
there is an edge $e = (i,k)$ between each encrypted chunk i and each available data center k, and the weight of e is given by $\mu_k \mathrm{v}_i$.

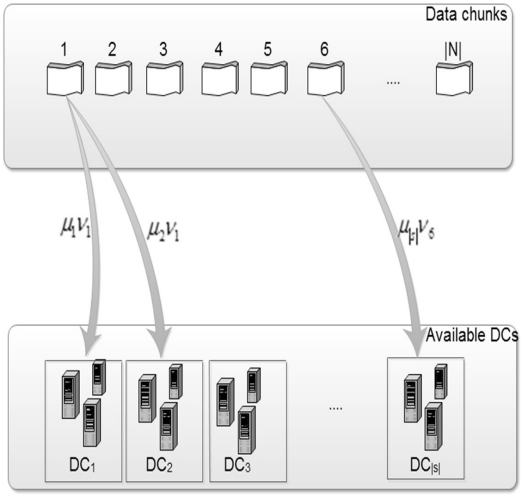

Figure 4: Complete bipartite graph construction.

We now introduce the well known "minimum weight b-matching problem" to build a combinatorial optimization solution. The b-matching is a generalization of the minimum weight matching problem and can be defined as follows (see (Korte and Vygen, 2001) for more details):

Definition Let G be an undirected graph with integral edge capacities: $u : E(G) \to \mathbb{N} \cup \infty$ and numbers $b : V(G) \to \mathbb{N}$. Then a b-matching in G is a function $f : E(G) \to \mathbb{N}$ with $f(e) \leq u(e)$, $\forall e \in E(G)$, and $\sum_{e \in \delta(v)} f(e) \leq b(v)$ for all $v \in \mathcal{V}(G)$.

In the above, $\delta(v)$ represents the set of incident edges of v. To simplify notation, with no loss in generality, we use E and $\mathcal{V}$ for the edges and vertices of G. That is we drop the G in $E(G)$ and $\mathcal{V}(G)$.

From the definition, finding a minimum weight b-matching in a graph G consists in identifying f such that $\sum_{e \in E} \gamma_e f(e)$ is minimum, where γ_e is an associated cost to edge e. This problem can be solved in polynomial time since the full description of its convex hull is given in (Korte and Vygen, 2001).

Proposition 3.1. *Let $G = (\mathcal{N} \cup \mathcal{S}, E)$ be a weighted complete bipartite graph built as described in Figure 4. Then, finding an optimal chunk placement solution is equivalent to an uncapacitated ($u \equiv \infty$) minimum weight b-matching solution, where $b(v) = 1$ if $v \in \mathcal{N}$ (v is a chunk) and for all vertices $v \in \mathcal{S}$, we put $b(0) = 0$, and for $v \geq 1$, we have*

$$b(v) = \left\lceil \frac{|\mathcal{N}| - \sum_{k=0}^{v-1} b(k)}{\beta} \right\rceil \quad (1)$$

where β is the minimum number of data centers to be used to store the data chunks. This parameter is required by end-users to avoid vendor lock-in.

To mathematically formulate our model, we associate a real decision variable x_e to each edge e in the bipartite graph. As shown in Figure 4, each edge links a chunk to a data center. After optimization, if the decision is $x_e = 1$ then chunk i ($i = I(e)$ initial extremity) will be stored in data center j ($j = T(e)$ terminal extremity). Since the solution of a b-matching problem is based on solving a linear program, an integer solution of the minimum weight b-matching is found in polynomial time. This is equivalent to the optimal solution of the chunk placement problem described in this section.

According to the storage costs listed previously and by defining the probability of failure of a data center (or a provider) noted by f, we assign each chunk to the best data center with minimum cost. We note by $Cost_{plac}$ the total cost of placing $|\mathcal{N}|$ chunks in an optimal manner. We can formulate the objective function as follows:

$$\min Cost_{plac} = \sum_{e \in E, e=(i,j)} \left(\frac{\mu_j}{1-f_j} \nu_i \right) x_e \quad (2)$$

where ν_i is the volume of chunk i, and $(1-f)$ is the probability of data center availability (or provider availability).

This optimization is subject to a number of linear constraints. For instance, the broker has to consider the placement of all data chunks, and each chunk will be assigned to one and only one data center (the chunk replication problem will be discussed in the next section). This is represented by (3):

$$\sum_{e \in \delta(v)} x_e = 1, \forall v \in \mathcal{N} \quad (3)$$

Each data center s has a capacity Q_s. This leads to the following constraints:

$$\sum_{C=1}^{|\mathcal{N}|} \nu_C x_{Cs} \leq Q_s, \forall s \in \mathcal{S} \quad (4)$$

According to end-user requirements and to guarantee high data availability, chunks will be deployed in different data centers to avoid vendor lock-in. This is given by the following inequality:

$$\sum_{C=1}^{|\mathcal{N}|} x_{Cs} \leq b(s), \forall s \in \mathcal{S} \quad (5)$$

Using the b-matching model with constraints (4), enables the use of the complete convex hull of b-matching and makes the problem easy in terms of combinatorial complexity theory.

Reference (Korte and Vygen, 2001) gives a complete description of the b-matching convex hull expressed in constraints (3), (4) and (5). These families of constraints are reinforced by *blossom inequalities* to get integer optimal solutions with continuous variables:

$$\sum_{e \in E(G(A))} x_e + x(F) \leq \left\lfloor \frac{\sum_{v \in A} b_v + |F|}{2} \right\rfloor, \forall A \in \mathcal{N} \cup \mathcal{S}, \quad (6)$$

where $F \subseteq \delta(A)$ and $\sum_{v \in A} b_v + |F|$ is odd, and $\delta(A) = \sum_{i \in A, j \in A} x_{(ij)}$. $E(G(A))$ represents a subset of edges of the subgraph $G(A)$ generated by a subset of vertices A. An in depth study of blossom constraints (6) is out of the scope of this paper, but more details can be found in (Grotschel et al., 1985).

Based on the bipartite graph G, we constructed a polynomial time approximation scheme of the data chunks placement problem by identifying the b-matching formulation. The blossom constraints (6) are added to our model to get *optimal integer solutions* of the *placement problem* whose model is finally given by:

$$\begin{array}{l} \min Cost_{plac} = \sum_{s=1}^{|\mathcal{S}|} \sum_{C=1}^{|\mathcal{N}|} \frac{\mu_s}{1-f_s} \nu_C x_{Cs} \\ S.T.: \\ \left\{ \begin{array}{l} \sum_{s=1}^{|\mathcal{S}|} x_{Cs} = 1, \forall C \in \mathcal{N} \\ \sum_{C=1}^{|\mathcal{N}|} \nu_C x_{Cs} \leq Q_s, \forall s \in \mathcal{S} \\ \sum_{C=1}^{|\mathcal{N}|} x_{Cs} \leq b(s), \forall s \in \mathcal{S} \\ \sum_{e \in E(G(A))} x_e + x(F) \leq \left\lfloor \frac{\sum_{v \in A} b_v + |F|}{2} \right\rfloor, \forall A \in \mathcal{N} \cup \mathcal{S} \\ F \subseteq \delta(A), \sum_{v \in A} b_v + |F| \ \ is \ \ odd \\ x_{Cs} \in \mathbb{R}^+, \forall C \in \mathcal{N}, \ \ \forall s \in \mathcal{S} \end{array} \right. \end{array} \quad (7)$$

The variables and constants used in the final model are summarized as follows:

3.3 Data Replication Algorithm

To enhance performance and availability of end-user stored data, we propose a replication model of data chunks depending on data center failure probabilities, and expected availability (noted by A_{expec}) required by each user. The objective consists in finding the optimal trade-off between data center availability and storage costs. This leads to avoiding expensive data centers with high failure probability.

We assume that each data chunk is replicated r times, and reconstituting a file data needs to get one

Table 1: Variables and constants of the model.

Variables	Meaning
$\mathcal{N}$	set of data chunks
$\mathcal{S}$	set of data centers
v_C	volume of a data chunk C
μ_j	storage cost per Gigabyte/month of provider j
x_e	real variable indicating if e is solicited or not
b_v	upper bound of the degree of v
$\delta(A) =$	$\sum_{i\in A, j\in A} x_{(ij)}$
$\delta(v)$	set of incident edges to v
β	minimum number of providers

copy of all chunks (i.e. $|\mathcal{N}|$ chunks among $r\times|\mathcal{N}|$ are necessary to reconstruct a data). Figure 5 gives more details and shows chunks replication procedure.

It is important to note that initially, each encrypted chunk will be replicated by the selected hosting providers within their data centers, and the broker can reinforce this mechanism by proposing to add more replicas guaranteeing higher data availability.

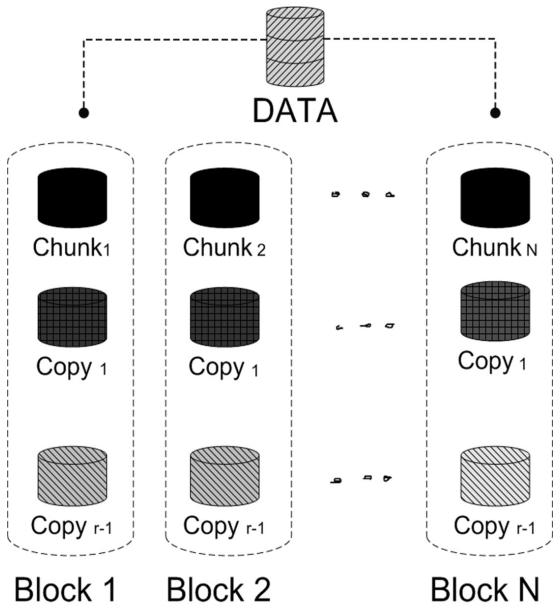

Figure 5: Data replication.

In the following, we would like to replicate $|\mathcal{N}|$ chunks into $|\mathcal{S}|$ data centers according to various costs (storage costs) and performance requirements such as the data availability. We suppose that $S = \{s_1, s_2, \ldots, s_{|S|}\}$ and for the sake of simplicity (due to the problem NP-Hardness), we suppose w.l.o.g. each data center has a large amount of storage resources able to host data chunks and replicas. We associate each data center $s \in \mathcal{S}$ with a probability of failure f_s.

We suppose (as cited above) that each data chunk C ($C = \overline{1,|\mathcal{N}|}$) has r replicas to place in r data centers that do not contain the chunk C. Thus we ask the following question: *How do we replicate data chunks through available data centers so that the total cost of storage is optimal (minimal) and data availability is maximal?*

Thus, for each chunk C, the problem consists in selecting a subset φ_C of r available data centers that do not contain C, leading to a minimum storage cost and a high probability of data availability.

We note by $P(C)$ the probability of chunk C availability (respect. $P(\overline{C})$ is the probability of non-availability of a chunk C). $P(D)$ is the probability of data availability (respect. $P(\overline{D})$ is the probability of non-availability of data D). Note that a chunk C is not available if all of its copies are not available (see Figure 5). In other words, a block in Figure 5 with r replicas is non available if all of the data centers storing this block are non available. By supposing the data centers are independent, we get the following proposition:

Proposition 3.2. $P(\overline{C}) = \prod_{s\in\varphi_C} f_s$

Proof.

$$\begin{aligned} P(\overline{C}) &= P(\overline{C_1} \text{ and } \overline{C_2} \text{ and } \ldots \text{ and } \overline{C_r}) \\ &= P(\overline{C_1}) \times P(\overline{C_2}) \times \ldots \times P(\overline{C_r}) \\ &= \prod_{s\in\varphi_C} f_s \end{aligned}$$

Proposition 3.3. $P(D) = \prod_{C=1}^{|\mathcal{N}|} \left(1 - \prod_{s\in\varphi_C} f_s\right)$

Proof. A data D with $r\times|\mathcal{N}|$ chunks, is entirely available if all chunks are available. According to Proposition (3.2), the probability of data file availability (i.e. $P(D)$) is then given by:

$$\begin{aligned} P(D) &= \prod_{C=1}^{|\mathcal{N}|} P(C) \\ &= \prod_{C=1}^{|\mathcal{N}|} \left(1 - \prod_{s\in\varphi_C} f_s\right) \end{aligned}$$

The QoS requirement for end-users is presented by the data availability. This is noted by A_{expect} (as used in (Mansouri et al., 2013) for example). Thus, to meet end-user QoS requirement, the broker should replicate each D in a selected sub-set of data centers that satisfies:

$$\prod_{C=1}^{|\mathcal{N}|} \left(1 - \prod_{s\in\varphi_C} f_s\right) \geq A_{expect} \tag{8}$$

We derive a mathematical model to efficiently reduce the replication costs noted by $Cost_{rep}$, under the QoS requirements described by the inequality (8). As the number of replicas of each chunk is supposed to be r, we seek an optimal sub-set of data centers of size r to store the replicas of each chunk. Moreover, our solution should not put all the chunks within the same data center to avoid vendor lock-in. Thus, in the following, we address a mathematical optimization model to efficiently replicate all the chunks of a data D.

$$\begin{array}{l} \min_{\varphi_C} Cost_{rep} = \sum_{C=1}^{|\mathcal{N}|} \sum_{s\in\varphi_C} \mu_s \nu_C \\ S.T.: \\ \left\{ \begin{array}{ll} \prod_{C=1}^{|\mathcal{N}|} \left(1-\prod_{s\in\varphi_C} f_s\right) \geq A_{expect}, & ; \\ |\varphi_C| = r, & \forall C = \overline{1,|\mathcal{N}|}; \end{array} \right. \end{array} \quad (9)$$

To solve the model (9), we can resort to dynamic programming approach as the objective function of (9) is separable and monotone. As these methods resort to recursion technique, they can prove to be expensive in certain cases due to the exponential number of data centers subsets to enumerate. For this reason, and for the sake of scalability, we prefer to address a simple, scalable and succinct algorithm to reach near optimal solutions for large instances in few seconds.

Solving the model (9) is equivalent to find a subset of data centers able to host chunks in a cost efficient manner, and that satisfies the requirement (8). We propose a simple and scalable algorithm to solve (9) in few seconds for large number of data centers and data chunks. Without loss of generality, we assume that minimizing a function Z is approximatively equivalent to minimize $\ln(Z)$. Thus, for each chunk C, we seek a subset of data centers that minimizes $\ln(\prod_{s\in\varphi_C} f_s)$. This is equivalent to minimize $\sum_{s\in\varphi_C} \ln(f_s)$. Moreover, We construct a new bipartite graph $G_2 = (V_2 \cup S_2, E_2)$, where V_2 is the set of chunks to be stored and S_2 is the set of all available data centers (see Figure 6). E_2 is the set of weighted edges between the two parts of vertices of G_2. There is an edge between each chunk C and each data center s (not containing a copy of chunk C) with a weight given by $\ln(f_s)$. If a data center s has already stored a copy of chunk C, then the weight of the edge (C,s) is equal to 2. Figure 6 gives more details.

From graph G_2, we identify a minimum weight b-matching with a given vector b as follows :

- for each $v \in V_2$, degree of v is equal to $b(v) = r-1$,
- the degree of each vertex $v \in S_2$ is equal to $b(v)$ given by (1).

To summarize, we give the following algorithm,

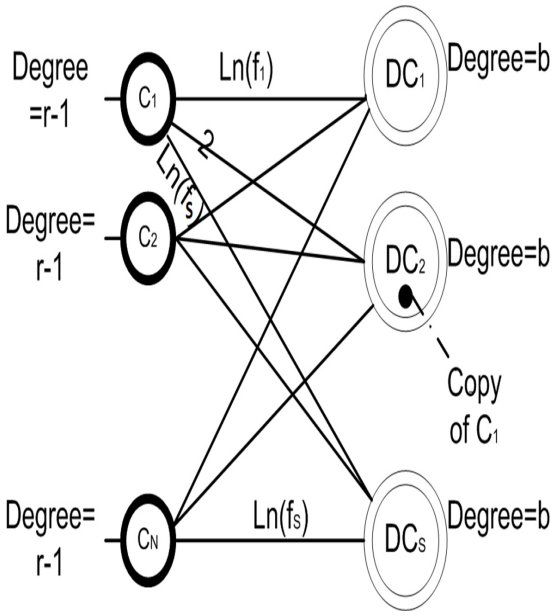

Figure 6: New bipartite graph G_2 to replicate chunks.

leading to find the best subset of data centers to replicate all the chunks in a cost efficient manner, verifying condition (8).

Algorithm 1: Data replication algorithm.

Step 0: Construct the bipartite graph G_2 (see Figure 6);
Step 1: Compute a b-Matching with a minimum cost solution using the vector b;
Step 2: Check if (8) is satisfied;
Step 3: If (8) is not satisfied, GOTO Step 0, by incrementing the degrees of vertices in S_2;

The algorithm 1 is deployed to replicate efficiently $r-1$ copies of each chunk C of a data D.

3.4 Data Chunk Splitting

In this section, we discuss the rational number of chunks ($|\mathcal{N}^*|$) to be used to split the data according to data center failure probabilities (f_s for a DC s), number of replicas (r) of each chunk, and the data availability expected by end-users (A_{expect}).

According to Proposition (3.3), we seek a rational number of encrypted chunks to get after splitting the data when satisfying end-users QoS represented by data availability A_{expect}. We get the following inequality :

$$P_D = \prod_{C=1}^{|\mathcal{N}|} P_C = \prod_{C=1}^{|\mathcal{N}|} \left(1 - \prod_{s\in\varphi_C} f_s\right) \geq A_{expect} \quad (10)$$

As $A_{expect} < 1$ and $\prod_{C=1}^{|\mathcal{N}|}\left(1-\prod_{s\in\varphi_C} f_s\right) < 1$, inequality (10) leads to the following one:

$$\ln\left(\prod_{C=1}^{|\mathcal{N}|}\left(1-\prod_{s\in\varphi_C} f_s\right)\right) \leq \ln\left(A_{expect}\right) \qquad (11)$$

We also note that for each chunk indexed by C, we have r replicas and then $|\varphi_C| = r$. For the sake of simplicity, we also suppose that the failure probability of each data center is close to the average failure probability given by $\overline{f}$. This allows us to deduce :

$$\prod_{s\in\varphi_C} f_s = \left(\overline{f}\right)^r \qquad (12)$$

And following inequality (11), we get:

$$|\mathcal{N}| \times \ln\left(1-\overline{f}^r\right) \leq \ln\left(A_{expect}\right) \qquad (13)$$

According to (13), we deduce the number of data chunks as follows :

$$|\mathcal{N}^*| \geq \frac{\ln\left(A_{expect}\right)}{\ln\left(1-\overline{f}^r\right)} \qquad (14)$$

4 NUMERICAL RESULTS

To evaluate and assess performance, our algorithms have been implemented and evaluated using simulations and an experimental platform managed by an instance of OpenStack (Openstack, 2014). The linear programming solver CPLEX (CPLEX, 2014) was used to derive the b-matching solution and the Bin-Packing solution used to benchmark our heuristic.

As our goal in this paper is to analyze and discuss the applicability and the interest of storage brokering services in interaction with multiple data centers or cloud providers, we devote some numerical results to cross validating our proposed algorithms and assessing their cost efficiency and scalability for large data sizes. It is obvious to remark that the Bin-Packing model used to place data chunks invokes a branch and bound approach leading to explore the entire space of all the existing solutions. This leads to find "optimal" solutions for small data sizes serving as a benchmark for other approaches and algorithms. As the data size increases, the optimal solution for data chunk placement can only be found in exponential time. Thus, for large data, we resort to our heuristic solution based on graph theory and the b-matching approach.

In addition, our performance evaluation seeks to identify the limits of the discussed problem in terms of algorithmic complexity, and its suitability for optimizing real life instances. We will also determine the gap between the suboptimal heuristic solutions and the optimal solution provided by the Branch and Bound model when it can be reached in acceptable times.

4.1 Simulation Environment

The proposed algorithms in this paper were evaluated using a 1.70 GHz server with 6 GBytes of available RAM. We used data files with sizes ranging from 100 Megabytes to 4 Gigabytes. These data were stored in a distributed manner over a number of available data centers or providers ranging from 10 to 50. We associate with each data center, a data price per Gigabyte and per month, uniformly generated between 0 $ and 1 $. Each data is splitting multiple chunks and each chunk size is equal to 1 Megabyte. This configuration leads to construct a full mesh bipartite graph as described above. The number of generated bipartite graphs was set to 100 in our simulations yielding an average value reported in the following curves and tables. Without loss of generality, we suppose that each data center has an unlimited storage capacity.

In addition, we also used a platform of 20 servers running a Havana instance of OpenStack (Openstack, 2014) in a multi-node architecture. Each server (assimilated to a data center in real life) proposes Swift containers (Swift, 2014) to store data chunks. We associate a storage cost to each container (or DC) as described above. It is important to note that we used Swift API only to guarantee PUT and GET operations from and to the broker by intercepting and hosting encrypted chunks, without considering Swift replication policy. To improve our broker functionalities, we will add an S3 compatible interface allowing end-users to request the broker storing their data within Amazon S3.

4.2 Performance Evaluation

The first experiment consists in comparing the Bin-Packing and b-Matching (heuristic) approaches in terms of delay to derive the optimal and suboptimal solutions, respectively. We report different scenarios in Table 2, varying the number of data centers able to store end-users data (from 12 to 700 DCs), and the number of chunks ranging from 50 chunks to 2000 chunks, which is equivalent to store data size of 50 Megabytes to 2000 Megabytes, as each chunk is of 1 Megabyte.

To get a better grasp of the relative performance of the two algorithms, we generate 100 runs and take the average value of each instance, as reported.

The performance of the heuristic algorithm com-

pared to the optimal solution is represented by a gap defined as the percentage difference between the cost of the optimal and the heuristic solutions:

$$Gap(\%) = 100 \times \frac{bM_{sol} - BP_{sol}}{BP_{sol}} \quad (15)$$

where BP_{sol} is the cost of the exact solution provided by the Bin-Packing algorithm (to use as a reference or benchmark) and bM_{sol} is the cost of the b-Matching solution.

Table 2 reports the results of the evaluation and clearly shows the difficulty to reach optimal solutions using the Bin-Packing (Branch and Bound) algorithm whose resolution times become prohibitive for the scenarios of a data file of 2 Gigabytes to be distributed on a selected set of data centers among 300, 500 and 700 providers or data centers. Our heuristic solution performs close to optimal with *Gap* not exceeding 6% for the evaluated cases. More specifically the gap is in the interval [0.65%; 5.93%].

The results shown in Table 2 illustrate the difficulty to optimally solve the data chunks placement problem (see case of a data of 50 Mb with 25 DCs). At the same time, they demonstrate that the heuristic approach can find good and near-optimal solutions whose cost is quite close to the optimum (see case of data with 2000 MB and 700 DCs). Our algorithm provides an excellent trade-off between convergence time, optimality, scalability and cost. With respect to convergence time as seen in the third column of Table 2, it converges in a few seconds for the scenario with 2000 chunks and 700 DCs (54 secs compared to more than 3 hours for Bin-Packing).

To get a better grasp of the relative performance of the two algorithms used in this paper, a data file of 100 Megabytes is used and split into 100 encrypted chunks to be stored in a number of data centers ranging from 20 to 200. Figure 7 shows the characteristics of the algorithms. The b-matching algorithm achieves the best cost performance since it has consistently incurred the smallest cost, very close to the Bin-Packing which does not scale (as seen in Table 2). Exceptionally, one can remark in Figure 7 (the scenario with 20 to 40 available DCs), the cost found by the b-Matching is slightly lower than the cost of the Bin-Packing leading to negligible SLA violations caused by the quality of the upper bound given by equation (1) which should be enhanced in a future work. This is explained by the difficulty to optimally store and place all the data chunks in different data centers.

Another experiment consists in evaluating the proposed heuristic solution to determine the trade-off between storage cost and data availability. We associate with each user a required percentage of its data availability, denoted by A_{expect}. We reformulate A_{expect} in terms of the number of nines required by a user. We simulated a cloud storage market of 15 data centers belonging to different providers having different failure rates. For example, Amazon S3 (AWS, 2014) offers two levels of services: "Standard Storage" witch has 11 nines of storage availability for 0.03$ per Gigabyte per month, while "Amazon S3 Reduced Redundancy Storage (RRS)" has 4 nines of data availability for 0.024$ per GB per month. The simulated market is summarized in Table 3.

We consider a user data of 100 Gigabytes, and we investigate four methods to find the trade-off between a maximum data availability and a minimum price (cost). We use the following scenarios:

Table 2: Encrypted data chunks placement: b-Matching algorithm performances.

$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{S}\rvert$	b-Matching Time (sec)	Bin-Packing Time (sec)	Gap (%)
	12	0.15	0.16	2.24
50	**25**	0.15	0.16	5.93
	40	0.17	0.18	2.06
	25	0.17	0.20	3.08
100	**50**	0.18	0.20	0.65
	75	0.20	0.22	2.98
	100	1.10	2.11	1.94
500	**250**	1.27	3.68	4.37
	350	1.33	4.20	0.97
	200	7.22	12.7	5.36
1000	**400**	8.5	17.5	1.37
	700	10.4	22.6	3.66
	300	30.7	$> 3H$	1.45
2000	**500**	45.2	$> 3H$	4.3
	700	54.8	$> 3H$	0.81

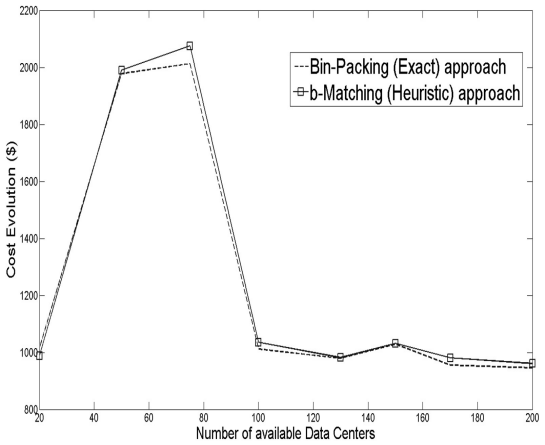

Figure 7: Storage cost gap.

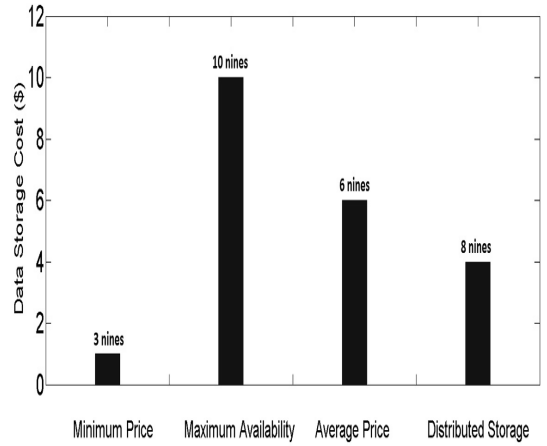

Figure 8: Data storage cost and availability trade-off.

- **Minimum Price:** A user selects simply the cheapest provider in the market (Provider 15 proposing a price of $0.01 per Gigabyte and per month in Table 3) without concerns on data availability (3 nines). Following this approach, the data will be stored with a total minimum costs of 1$ and a weak data availability (3 nines in Figure 8). Moreover, the user is locked-in within one cloud provider with a weak data availability. This can lead to disrupting services and loss of data.
- **Maximum Availability:** A user selects the provider with high availability in the simulated market (Provider 1 with 10 nines in Table 3). According to pricing proposal of Provider 2, the total storage cost is higher than the cost of the first scenario (10 $ in Figure 8). This may also lead to users' lock-in within the same provider.
- **Average Price:** In this case, we use the average price of the market, and we store the data within the provider with equivalent price (Provider 9 with 0.06$ per Gigabyte per month in Table 3). The total data cost in this case is equal to 6$ with 6 nines of data availability (according to the proposal of Provider 6). This scenario presents higher data availability than scenario 1 with a considerable cost increase. In this case, we also solicited one provider to store the data, which may cause user lock-in.
- **Distributed Storage:** We used our proposed approach (Algorithm 1) to find the trade-off between data availability and price. As depicted in Figure 8, our solution reaches a maximum availability of 8 nines with a minimum cost of 4$. This is due to data distribution over a set of selected providers with high availability and reasonable prices, avoiding user lock-in at the same time.

Table 3: Storage market costs and data availability.

Providers	**Price** ($/GB/month)	**Data Availability**
Prov 1	0.1	99.99999999%
Prov 2	0.095	99.99999995%
Prov 3	0.09	99,9999999%
Prov 4	0.085	99,9999995%
Prov 5	0.08	99,999999%
Prov 6	0.075	99,999995%
Prov 7	0.07	99.99999%
Prov 8	0.065	99,99995%
Prov 9	0.06	99,9999%
Prov 10	0.055	99,9995%
Prov 11	0.05	99,999%
Prov 12	0.04	99,995%
Prov 13	0.03	99.99%
Prov 14	0.02	99.95%
Prov 15	0.01	99.9%

A last experiment consists in evaluating the behavior of the number of replicas (noted by r) of each chunk with the evolution of the number of data chunks ($|\mathcal{N}^*|$) identified in (14) for example. We supposed that the average value of data centers failure probability is equal to 10^{-3}, when the expected data availability required by cloud consumers is equal to 99.9999%.

Figure 9 depicts the evolution of r for different chunks number ranging from 1 to 60. Thus, we remark that for a number of chunks $|\mathcal{N}^*| \leq 43$, the number of required replicas is equal to 2, and for $|\mathcal{N}^*| \geq 44$ chunks, the number of replicas converges to 3 and there is no need to replicate more even for larger number of chunks. This may lead to store large data volumes with reduced costs when satisfying the required QoS (data availability). Note that this result is very similar than that determined by the Google File System solution (Ghemawat et al., 2003).

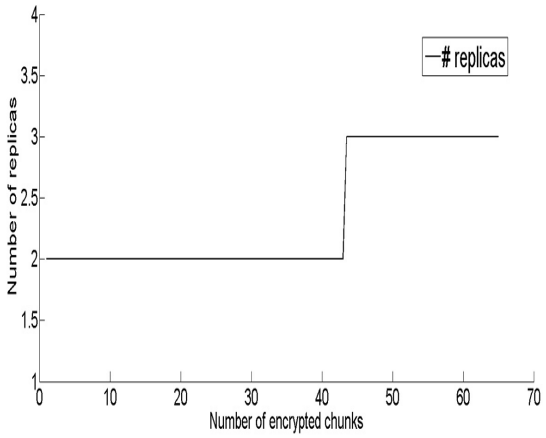

Figure 9: Data chunks replication behavior.

5 SUMMARY AND FUTURE WORK

In this paper we considered an encrypted and distributed solution allowing to store users' data in different providers' data centers offering storage services with different prices and SLAs. To eliminate user lock-in and to liberate user data from a unique provider, we proposed a new efficient and scalable solution based on b-Matching theory to optimize the storage cost and the data failure at the same time. The b-Matching algorithm works in tandem with a replication solution allowing to efficiently increase the data availability of end-users. This replication algorithm is based on a simple and fast approach giving near optimal solutions even for large problem instances.

In future work, we will reinforce our mathematical model of data chunk placement based on b-Matching theory, to consider network constraints when users are involved in PUT and GET operations. This may lead cloud consumers to combine requests of compute (as EC2 instances (EC2, 2014)) services with storage services (as Google Drive (Google, 2014)) at the same time. Thus, we will reinforce our broker's functionalities to give cloud consumers various means to consume proposed cloud resources in a more secure manner with reduced cost.

REFERENCES

Abu-Libdeh, H., Princehouse, L., and Weatherspoon, H. (2010). Racs: A case for cloud storage diversity. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 229–240, New York, NY, USA. ACM.

AWS (2014). http://aws.amazon.com/fr/s3/pricing/.

Balasaraswathi, V. and Manikandan, S. (2014). Enhanced security for multi-cloud storage using cryptographic data splitting with dynamic approach. In *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on*, pages 1190–1194.

Bonvin, N., Papaioannou, T., and Aberer, K. (2010). A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 205–216, New York, NY, USA. ACM.

Chia-Wei, C., Pangfeng, L., and Jan-Jan, W. (2012). Probability-based cloud storage providers selection algorithms with maximum availability. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 199–208.

CPLEX (2014). http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

EC2 (2014). http://aws.amazon.com/fr/ec2/.

Ford, D., Labelle, F., Popovici, F., Stokely, M., Truong, V., Barroso, L., Grimes, C., and Quinlan, S. (2010). Availability in globally distributed storage systems. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*.

Ghemawat, S., Gobioff, H., and Leung, S. (2003). The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43.

Google (2014). drive.google.com/.

Grotschel, M., Lovsz, L., and Shrijver, A. (1985). Geometric algorithms and combinatorial optimization. *Springer*.

Jindarak, K. and Uthayopas, P. (2012). Enhancing cloud object storage performance using dynamic replication approach. In *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pages 800–803.

Korte, B. and Vygen, J. (2001). Combinatorial optimization: theory and algorithms. *Springer*.

Li, J. and Li, B. (2013). Erasure coding for cloud storage systems: A survey. *Tsinghua Science and Technology*, 18(3):259–272.

Mansouri, Y., Toosi, A., and Buyya, R. (2013). Brokering algorithms for optimizing the availability and cost of cloud storage services. In *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science - Volume 01*, CLOUDCOM '13, pages 581–589, Washington, DC, USA. IEEE Computer Society.

Myint, J. and Thu, N. T. (2011). A data placement algorithm with binary weighted tree on pc cluster-based cloud storage system. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 315–320.

Negru, C., Pop, F., Cristea, V., Bessisy, N., and Jing, L. (2013). Energy efficient cloud storage service: Key issues and challenges. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on*, pages 763–766.

NIST (2014). Announcing the advanced encryption standard (aes).

Openstack (2014). https://www.openstack.org/.

Qingsong, W., Veeravalli, B., Bozhao, G., Lingfang, Z., and Dan, F. (2010). Cdrm: A cost-effective dynamic replication management scheme for cloud storage cluster. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 188–196.

Rodrigo, R. and Liskov, B. (2005). High availability in dhts: Erasure coding vs. replication. In *Peer-to-Peer Systems IV 4th International Workshop IPTPS 2005*, Ithaca, New York.

Seungmin, K., Bharadwaj, V., and KhinMiMi, A. (2014). Espresso: An encryption as a service for cloud storage systems. *Monitoring and Securing Virtualized Networks and Services*, pages 15–28.

Srivastava, S., Gupta, V., Yadav, R., and Kant, K. (2012). Enhanced distributed storage on the cloud. In *Computer and Communication Technology (ICCCT), 2012 Third International Conference on*, pages 321–325.

Swift (2014). http://docs.openstack.org/developer/swift/.

Thanasis, G. P., Bonvin, N., and Aberer, K. (2012). Scalia: An adaptive scheme for efficient multi-cloud storage.

In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 20:1–20:10, Los Alamitos, CA, USA. IEEE Computer Society Press.

Varghese, L. and Bose, S. (2013). Integrity verification in multi cloud storage. In *Proceedings of International Conference on Advanced Computing*.

Weatherspoon, H. and Kubiatowicz, J. (2002). Erasure coding vs. replication: A quantitative comparison. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 328–338, London, UK, UK. Springer-Verlag.

Yanzhen, Q. and Naixue, X. (2012). Rfh: A resilient, fault-tolerant and high-efficient replication algorithm for distributed cloud storage. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 520–529.

Zhang, Q., Xue-zeng, P., Yan, S., and Wen-juan, L. (2012). A novel scalable architecture of cloud storage system for small files based on p2p. In *Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on*, pages 41–47.