Brokering Algorithms for Optimizing the Availability and Cost of Cloud Storage Services

Yaser Mansouri, Adel Nadjaran Toosi and Rajkumar Buyya
Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: {yase, adeln}@student.unimelb.edu.au, rbuyya@unimelb.edu.au

***Abstract*—In recent years, cloud storage providers have gained popularity for personal and organizational data, and provided highly reliable, scalable and flexible resources to cloud users. Although cloud providers bring advantages to their users, most cloud providers suffer outages from time-to-time. Therefore, relying on a single cloud storage services threatens service availability of cloud users. We believe that using multi-cloud broker is a plausible solution to remove single point of failure and to achieve very high availability. Since highly reliable cloud storage services impose enormous cost to the user, and also as the size of data objects in the cloud storage reaches magnitude of exabyte, optimal selection among a set of cloud storage providers is a crucial decision for users. To solve this problem, we propose an algorithm that determines the minimum replication cost of objects such that the expected availability for users is guaranteed. We also propose an algorithm to optimally select data centers for striped objects such that the expected availability under a given budget is maximized. Simulation experiments are conducted to evaluate our algorithms, using failure probability and storage cost taken from real cloud storage providers.**

***keywords*- Cloud Computing, Cloud Storage, Broker, Availability, Data replication, Data striping, Dynamic Programming, Cost minimization**

I. Introduction

Cloud storage is a novel paradigm for storing user objects[1] on a remote location in large scale. During recent years, some of the cloud storage companies, such as Amazon, Rackspace, Google, etc. have provided on-line mass storage to cloud users. Since every storage service belongs to a different company, they offer services in different terms and costs.

A typical cloud storage Service Level Agreement (SLA) articulates precise levels of the services such as availability of the services which are in operation. In the context of intense economic competition, different cloud storage providers supply a variety of services with different SLAs, which are proportional to the cost. That is, users interested in more reliable SLA must pay more. Moreover, as the total size of user objects in the cloud storage reach up to several exabyte (2^{60} bytes), it can impose enormous cost on users. Therefore, optimal selection of cloud storage providers in terms of higher availability and lower cost is a crucial decision to users.

Availability of service as an imperative criterion in the SLA is listed as one of the top ten obstacles to the growth of cloud computing [1]. Although the most well known cloud storage providers such as Amazon, Rackspace and Google, etc. warranty availability of services in high level, software bugs, user errors, administrator errors, malicious insiders, and natural catastrophes endangering availability are inevitable and unpredictable [2]. This is why some well-known cloud providers have experienced outages in their data centers [1], and the number of vulnerability incidents has doubled from 2009 to 2011 [3]. Availability of services is defined as the ratio of the total time that the storage services of a cloud provider is accessible during a given interval (e.g., one year) to the length of the interval. The metric which we use for availability is number of nines [4]. For example, if the availability of the system is 99.9% then we refer it as three nines. The system with three nines availability is expected to have 8.75 hours downtime per year.

One simple way to get the desired availability is to replicate objects in multiple data centers. This approach is costly because as the number of replicas increases, the storage cost of the object raises. Therefore, minimizing cost with the aim of achieving desired availability as a required Quality of Service (QoS) is a key decision to user, which has not been studied very well.

Data Lock-in is another main problem among the top ten obstacles in regard to cloud computing. This is undesirable for users because they are vulnerable to rise in price, to decrease in availability and even to the cloud provider's bankruptcy [1]. Users also lose a chance to migrate from a cloud storage provider to another when new cloud providers emerge with better services or with lower price in the market. This is because some cloud storage providers charge the users for download service, which imposes heavy cost on users especially when one requires a large storage volume of a particular cloud storage provider. Moreover, transferring large objects from one cloud storage provider to another through the network is time consuming and most often is impossible.

One solution to mitigate the data lock-in and to allow users to migrate quickly from a cloud provider to another in reaction to any provider changes is placing an object at a fine granularity rather than a coarse one [5]. That is, the object, for example a database table or an archival object, is split to chunks, and stored in different cloud storages. Therefore, we need an algorithm to find the optimal placement of chunks

[1]Data and object are used interchangeably in this paper.

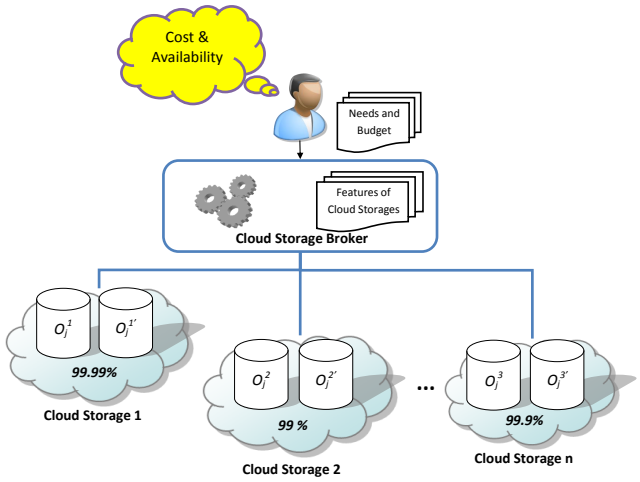


Fig. 1. Cloud Storage Broker

according to the user's needs and budget. This algorithm can be used in a *cloud storage broker*, which provides transparent object access among several cloud storage providers [5]. The broker, as illustrated in Fig. 1, gathers all features of cloud storage providers and assists users in finding the right cloud storage based on their required QoS.

In this paper, we propose algorithms that can be embedded in a cloud storage broker. These algorithms help the user to find a suitable placement of objects according to the required QoS. The first problem we focus on is to minimize the storage cost of objects while a given expected availability is met. The second problem is how to select the optimal placement of each strip of object such that the expected availability under a given budget is maximized. Due to the growing number of cloud storage providers with different characteristics and price, both problems are challenging and important.

The main **contributions** of this paper are:

- a mathematical model for the data center selection problem in which the objective function, cost function and constraints are clearly defined,
- an algorithm to select a subset of a given data centers to minimize the storage cost for objects when the expected availability is given, and
- a dynamic algorithm to select cloud providers optimally for storing objects that are split into chunks and each chunk is replicated a fixed number of times, such that the expected availability under a given budget is maximized.

The rest of this paper is organized as follows: Section II is devoted to an overview of related works. Section III describes an algorithm employed to calculate the minimum storage cost for storing objects subject to a given expected availability. Section IV presents the dynamic algorithm proposed to determine optimal data centers for storing striped objects to maximize expected availability with a given budget. Section V reports the experimental results of our simulation. Finally, conclusions and future works are stated in section VI.

II. RELATED WORK

Cloud computing has captured significant attention from both industry and academic in the recent decade. Fox et al. [1] provided a comprehensive overview on classification, obstacles, and opportunities for cloud providers. Our work focuses on alleviating two main obstacles of cloud computing: *availability of services* and *data lock-in*.

Le et al. [6] propose a CloudCmp tool that compares four well-known cloud providers in order to measure the elastic computing, persistent storage, and network services which directly affect performance of user application. This tool helps users to select a cloud based on their requirements and budget. Another tool, RACS (Redundant Array of Cloud Storage), is a cloud storage proxy that transparently splits an object among multiple cloud providers [5]. The goal of this study is to increase availability and mitigate object lock-in. The idea behind the study is the same as one of well-known techniques at the level of disks and file systems, RAID5. They evaluated the performance of RACS, and concluded the cost of migration from a cloud provider to another can be decreased seven-fold by using RACS. Their work differs from ours since our objective functions and constraints are completely different.

Ford et al. [7] investigated object availability in Google's main storage infrastructure, and analyzed availability of components, e.g., machines, racks and multi racks in tens of Google storage clusters. In their analytical measurements, they predict object availability based on Markov chain modelling in a data center. On the contrary, we focus on maximizing object availability for a given budget in a set of data centers. Our work has a similar objective function with the problem studied by Chang et al. [4], but the main difference is an object is split to chunks and replicated across cloud providers in our work, while it is fully replicated in their model. The main motivation for our strategy is agile reacting to any changes in cloud providers [5]. Similar to [7], Bonvin et al. [8] propose a dynamic allocation of resources in a data cloud to host objects so that the availability to different user application is guaranteed in a cost-effective way. Their model is based on a virtual economy, in which each object partition operates as an individual optimizer and decides whether to replicate, migrate or remove itself based on net profit maximization related to the utility provided by the partition and its storage and maintenance. Placek et al. [9] designed a market broker for storage services that allows the owner to exchange the storage.

Sripanidkulchai et al. [10] considered three main requirements of cloud computing: large scale deployment, high availability, and problem resolution. They proposed three ways of achieving cloud services with high availability. One of the three solutions is to extend the architecture across different cloud providers, which is the focus of this paper as well. The second solution is to develop new virtualization technologies, and third one is to look at the technical differences between individual and enterprise sites that results in the observed difference in service availability.

To the best our knowledge, there is no work dealing with selecting a subset of data centers such that the storage cost is minimized with a given expected availability. In addition, achieving maximum availability with a given budget as a constraint such that objects are split into chunks replicated across several cloud storage has never been studied.

III. Minimizing cost with given expected availability

We present an algorithm to find subsets of data centers to store replicas of all objects such that the replication cost of them is minimized, and expected availability of objects as the user's QoS is satisfied. Note that the subset of data centers for each object can be different. From the user perspective, one type of object may be popular, whilst another type may be seldom used (e.g., archival objects) and they can tolerate lower expected availability as QoS. So, it is reasonable to allow popular objects to be stored in more available data centers at higher cost, and guarantee lower QoS for non-popular objects at lower cost. Our algorithms guarantee this criterion for objects. That is, the expected availability of objects is met according to the priority attached to their type. In other words, as the priority of a type of objects increases, the objects of that type are replicated in data centers with more availability. In order to introduce our algorithm, we first present the following notations and definitions.

Consider a set of independent data centers represented by $DC = \{d_1, d_2, ..., d_n\}$, where $d_i \in DC$ $(1 \leq i \leq n)$ is an individual data center. Assume that a pair of characteristics for each data center: a weight $s(d_i)$ representing the *storage cost* of an object in the data center d_i, and $f(d_i)$ the *failure probability* of d_i. Also, suppose that the replica set denoted by δ is a subset of data centers in DC such that each d_i hosts a replica of the object. Let M be the number of objects with equal size, and TN is the total number of nines for M objects. Assume that t is the average number of nines per object requested by a user as QoS. The value of TN either can be computed by $M \times t$ or can be a arbitrary value (that is the value of TN) which is determined by user. We define three types of QoS for objects, in descending order of priority: *Gold (G)*, *Silver (S)*, and *Bronze (B)*. A priority parameter P_Q is assigned to each QoS type Q such that the sum of priority parameters for three types of QoS is 1. That is, $\sum_Q P_Q = 1$, where $Q = \{G, S, B\}$ $(P_G \geq P_S \geq P_B)$. Note that the value of P_Q is used in a general sense in our algorithms. It can be interpreted as different priority measures such as the access probability of objects on average for a type of QoS, or the importance of objects to the user. Also, let all objects of QoS type Q be denoted by set J_Q, and EA_Q and EF_Q be the expected availability and expected failure of objects in J_Q, respectively.

Definition 1. (Objects Placement): Let $\Phi = \bigcup_{j=1}^{M}\{\Phi_j\}$ be a placement of objects, where Φ_j indicates a subset of data centers (i.e, $\Phi_j \subseteq DC$) that contains the jth object with r replicas. Therefore, for all Φ_j, $|\Phi_j| = r$ and $|\Phi| = M$.

Definition 2. (Replication Cost of Objects): Assume that the cost of object j is the sum of the replication cost of the object in a set of data centers Φ_j. Thus, the replication cost of Φ, referred to the cost of replicas for M objects, is defined as:

$$C(\Phi) = \sum_{j=1}^{M} C(\Phi_j), \tag{1}$$

where $C(\Phi_j) = \sum_{d_l \in \Phi_j} s(d_l)$ is the cost of storing object j with duplication factor r.

Definition 3. (Expected Availability of Objects): Let X_j be a discrete random variable with numerical values of $\{0, 1\}$ which shows whether the object j is available under the set Φ_j $(X_j = 1)$ or not $(X_j = 0)$. Thus, we have $E(X_j) = \sum_{x_j \in \{0,1\}} x_j P(X_j = x_j) = P(X_j = 1)$. Since X_j is referred to the availability of the jth object under Φ_j, the value of X_j is 0 only when all data centers in Φ_j fail. Otherwise, the value of X_j is 1 when at least one of data centers in set Φ_j containing object j is available. As a result, $E(\Phi_j) = E(X_j) = 1 - \prod_{d_i \in \Phi_j} f(d_i)$.

The expected availability for M objects is equal to the sum of the expected availability of each object j because the expected availability of the sum of independent random variables is the sum of the expected availability of these random variables. Thus, we have:

$$E(\Phi) = \sum_{j=1}^{M} E(\Phi_j) = \sum_{j=1}^{M}(1 - \prod_{d_i \in \Phi_j} f(d_i)). \tag{2}$$

In this sense, $E(\Phi)$ can be viewed as the expected availability of M objects under $\Phi = \sum_{j=1}^{M}\{\Phi_j\}$, and expressed as a number of nines.

In the rest of this section, we formally define the object placement problem, and then a dynamic algorithm to tackle this problem is proposed. Based on the above definitions, the problem can be defined as follows. Assume that a set of data centers DC, M objects with duplication factor r, the QoS requirement for user $E(\Phi)$, measured in the number of nines, and each QoS type associated with P_Q where $Q = \{G, S, B\}$ are given. The objective is to find a subset of data centers $\Phi_j \subseteq DC$ for each object j so that $E(\Phi)$ is satisfied, the replication cost of objects $C(\Phi)$ is minimized, and the expected availability for each type of objects is proportional to the priority parameter of that type. This is translated to:

$$\min \;\; C(\Phi) = \sum_{j=1}^{M} C(\Phi_j), \tag{3}$$

$$s.t. \;\; E(\Phi) = \sum_{j=1}^{M}(1 - \prod_{d_i \in \Phi_j} f(d_i)) \geq TN, \tag{4}$$

$$\forall J_Q, \;\; E(J_Q) \propto P_Q, \;\; Q = \{G, S, B\}, \tag{5}$$

where $E(J_Q)$ is the expected availability of objects which belong to QoS type Q.

In order to get a maximization problem, we mathematically redefine (3) as follows:

$$\max \ \ 1/\sum_{j=1}^{M} C(\Phi_j), \tag{6}$$

while the constraints are (4) and (5).

Before we propose our algorithm, let us express the feasibility of the problem. The sum of nines of all data centers $d_i \in \Phi$ must not be less than TN, if there exists a feasible solution for the above problem. That is, $E(\Phi) \geq TN$.

We propose a dynamic algorithm called *Minimum Cost Fixed Expected Availability* to solve the aforementioned objects placement problem. With no loss of generality, and assuming all objects have equal priority (that is, QoS type for objects is ignored), recursive equations of the dynamic algorithm can be obtained as explained below.

The first step of the dynamic algorithm is to define a recursive solution for two cases. *Case 1*: $j > 1$. This solution is calculated to $MC[j][tn]$, which means the minimum cost for the jth object[2] $(1 < j \leq M)$ with a given number of nines tn $(1 \leq tn \leq TN)$. To obtain the best placement for the jth object, we first find all combinations of r distinct data centers that can be chosen from DC. Assume that each combination is denoted by δ $(|\delta| = r)$. Since the availability of jth object is $E(\delta)$, we have the expected availability $tn - E(\delta)$ for the $j-1$ objects. It means we should consider all possible cases with $tn - E(\delta)$ for the first $j-1$ objects, assuming δ is a set of data centers, which contains the jth object. Therefore, if we place the jth object in the set δ of data centers, then the minimum cost of replication j objects in data centers equals to the minimum cost of replication $j-1$ objects with the expected availability $tn - E(\delta)$ plus the cost of data centers in the set δ. This mathematically translated into:

$$MC[j][tn] = 1/(\max_{\delta}(1/MC[j-1][tn - E(\delta)]) + C(\delta)). \tag{7}$$

Case 2: $j = 1$. All the possible subsets $\delta \subseteq DC$ so that $tn = E(\delta)$ are considered, and then the subset δ with the minimum cost is selected. In other words, $\Phi_1 = \delta$, where δ is a subset of data centers with the minimum cost of replication for the first object. Thus, the recursive function for $j = 1$ with fixed tn can be obtained as:

$$MC[1][tn] = \max_{\delta}(\frac{1}{C(\delta)}, MC[1][tn]). \tag{8}$$

The second step of the algorithm is the termination conditions. First, if $tn - E(\delta) < 0$, then $MC[j][tn] = 0$, which means the subset $\delta \subseteq DC$ should not be considered. Second, if $j = 1$ and there is not a subset of data centers such that $tn = E(\delta)$, $MC[1][tn]$ also is assigned to zero. Third, the value of MC should be infinity when j and tn are both zero. According to the above discussion, the proposed algorithm is outlined in Algorithm 1.

[2]In this section, we henceforth consider each object has r replicas, unless otherwise mentioned.

Algorithm 1 Minimum Cost Fixed Expected Availability

Input: $DC, M, TN, r, f(d_i), s(d_i)$
Output: $\frac{1}{MC[M][TN]}$

for $tn \leftarrow 1$ to TN **do**
  $MC[0][tn] \leftarrow +\infty$
end for
for $j \leftarrow 1$ to M **do**
  $MC[j][0] \leftarrow 0$
  for $tn \leftarrow 1$ to TN **do**
    $MC[j][tn] \leftarrow 0$
    for each *combination* $\delta \in (DC, r)$ **do**
      if $((tn - E(\delta)) \geq 0)$ **then**
        if $(j = 1)$ and $(tn = E(\delta))$ **then**
          $MC[1][tn] \leftarrow (\frac{1}{C(\delta)}, MC[1][tn])$
        end if
        if $(j > 1)$ **then**
          if $(MC[j-1][tn - E(\delta)] = 0)$ **then**
            $MC[j][tn] \leftarrow 0$
          else
            $C \leftarrow 1/MC[j\text{-}1][tn\text{-}E(\delta)] + C(\delta)$
            $MC[j][tn] \leftarrow \max(\frac{1}{C}, MC[j][tn])$
          end if
        end if
      end if
    end for
  end for
end for
Return $\frac{1}{MC[M][TN]}$

In order to consider constraint (5), we slightly revise Algorithm 1. First, $E(J_Q)$ is computed by $\lfloor tn \times P_Q \rfloor$ where $1 \leq tn \leq TN$. To guarantee constraint (5) accurately, $E(J_Q)$ is sorted decreasingly by dif_Q where $0 \leq dif_Q = tn \times P_Q - E(J_Q) < 1$, and then the first $tn - \sum_Q E(J_Q)$ of $E(J_Q)$ is increased by one. Second, having calculated $E(J_Q)$ for all objects $j \in J_Q$, it is sufficient to replace TN with $E(J_Q)$ in Algorithm 1.

IV. Maximum Expected Availability with Given Budget

One way to prevent object lock-in in the cloud storage is to store the object at a fine granularity rather than in coarse one [5]. Due to this advantage, in this section, our aim is to introduce a dynamic algorithm to provide the best placement for chunks of an object across the cloud providers so that the expected availability is maximized under a given budget. In the rest of the section, we define some preliminaries and definitions, and then we present the optimization problem in details.

In addition to notations in the previous section, we assume that each object is split to m chunks with the same size and replicated with duplication factor r. Since it is assumed that each replica of chunks of each object is placed in a separate

Algorithm 2 Maximum Expected Availability with a Given Budget

Input: $DC, M, m, B, r, f(d_i), s(d_i)$
Output: $E[M][B]$

```
for b ← 0 to B do
    E[0][b] ← 0
end for
for j ← 1 to M do
    E[j][0] ← −∞
end for
for j ← 1 to M do
    for b ← 1 to B do
        E[j][b] ← E[j][b − 1]
        for each combination δ ∈ (DC, m × r) do
            if (b − C(δ)) ≥ 0) then
                E(δ) ← Call OCP(δ, r, m)
                e ← E[j − 1][b − C(δ)] + E(δ)
                E[j][b] ← max(E[j][b], e)
            end if
        end for
    end for
end for
Return E[M][B]
```

data center, the number of data centers, n, must be at least $m \times r$ ($n \geq m \times r$). In fact, storing each object requires at least $m \times r$ independent data centers to gain maximum performance of striping [5].

Definition 4. (Chunks Placement): Assume that $\Phi_j = \bigcup_{k=1}^{m} \{\varphi_{j,k}\}$ is a placement set for chunks of object j, where $\varphi_{j,k}$ represents a subset of data centers (i.e, $\varphi_{j,k} \subset DC$) that containing r replicas of kth chunk. Therefore, for all $\varphi_{j,k}$ and Φ_j, we have $|\varphi_{j,k}| = r$ and $|\Phi_j| = m \times r$, respectively.

Definition 5. (Replication Cost of Chunks): Let $C(\varphi_{j,k})$ denote the cost of the kth chunk with r replicas of object j. Since cost per object in data center d_i is $s(d_i)$ and each object consists of m chunks, $C(\varphi_{j,k}) = \sum_{d_l \in \varphi_{j,k}} \lceil s(d_l) \rceil / m$. Therefore, the total replication cost of m chunks with duplication factor r of object j is given by:

$$C(\Phi_j) = \sum_{k=1}^{m} C(\varphi_{j,k}). \tag{9}$$

The total cost of M objects can be written as:

$$C(\Phi) = \sum_{j=1}^{M} \sum_{k=1}^{m} C(\varphi_{j,k}). \tag{10}$$

Definition 6. (Availability of Chunks): Suppose that X_j is a random variable as defined in Definition 3. Since object j is split to m chunks, we recalculate $E(X_j)$ as follows. The kth chunk with r replicas of object j is not available if all data centers $d_l \in \varphi_{j,k}$ fail. Thus, the failure probability of the kth chunk with r replicas is $\prod_{d_l \in \varphi_{j,k}} f(d_l)$. As a result, the kth chunk of object j is available if at least one replica of that is available, which results in $1 - \prod_{d_l \in \varphi_{j,k}} f(d_l)$ as availability of the kth chunk with r replicas. Obviously, the jth object can be retrieved, if all m chunks are available. Therefore, the expected availability of object j consisting of m chunks with duplication factor r under set Φ_j can be calculated as:

$$E(X_j) = E(\Phi_j) = \prod_{k=1}^{m} (1 - \prod_{d_l \in \varphi_{j,k}} f(d_l)). \tag{11}$$

Similar to the previous section, the expected availability of M objects termed by $E(\Phi)$ is the sum of $E(\Phi_j)$. Thus, we have:

$$E(\Phi) = \sum_{j=1}^{M} E(\Phi_j) = \sum_{j=1}^{M} (\prod_{k=1}^{m} (1 - \prod_{d_l \in \varphi_{j,k}} f(d_l)). \tag{12}$$

Now, we express the optimization problem, which lies in the above definitions and notations. Assume that a data center set DC, M objects consisting m chunks with duplication factor r and a budget B are given, and also suppose that each QoS type Q is associated with P_Q. The objective is to find a subset $\Phi_j \subseteq DC$ for each object j such that $E(\Phi)$ is maximized whilst $C(\Phi) \leq B$ and the expected availability of each type of objects is proportional to the priority parameter of that type. This is translated into:

$$\max \ E(\Phi) \ s.t. \ C(\Phi) \leq B \ and \ E(J_Q) \propto P_Q. \tag{13}$$

In order to solve (13), we propose a dynamic algorithm called *Maximum Expected Availability with a Given Budget*, in which the *Optimal Chunks Placement (OCP)* algorithm is called to provide optimal placement of chunks of an object. Algorithm 2 is presented without considering the constraint $E(J_Q) \propto P_Q$. That is, all objects have the same priority from the user perspective, and then this constraint is applied to the proposed algorithm.

In the proposed dynamic algorithm, let $E[M][B]$ be the expected availability for M objects with a given budget B. In the first step, we obtain a recursive formulation for $E[j][b]$, where $1 \leq j \leq M$ and $0 \leq b \leq B$. In order to store the jth object in the set DC of data centers, all possible δs of DC ($|\delta| = m \times r$) are checked, and then $C(\delta)$ by using (9) and $E(\delta)$ based on the OCP algorithm are calculated. Since the replication cost of the jth object is $C(\delta)$, we have the budget $b - C(\delta)$ to consider for storing the $j-1$ objects. Thus, considering all possible δs ($\delta \subseteq DC$) and all possible cases with budget $b - C(\delta)$ for $j-1$ objects, $E[j][b]$ is calculated as follows.

$$E[j][b] = \max_{\delta}((E[j-1][b - C(\delta)]) + E(\delta)). \tag{14}$$

In the second step, terminal conditions are considered. clearly, if $b - C(\delta) < 0$ then the subset δ is ignored and $E(\delta)$ is set to negative infinity. Also, if $b = 0$ and $j = 0$, $E[j][b]$ is initialized to zero. The proposed algorithm is outlined in Algorithm 2.

A. Optimal Chunks Placement (OCP) Algorithm

In this section, we discuss the OCP algorithm and its objective. Based on the Definition 4, an object has m chunks and each of them is replicated in r separate data centers. Without any special policy to select data centers for storing the chunks of an object, it might be some replicas of a chunk placed in more reliable data centers (that is data centers with less failure probability) whilst other replicas of another chunks are stored in less reliable ones. As a results, Equation (11) is not maximized. In order to maximize that, we should maximize availability of each chunk, that is $(1-\prod_{d_l \in \varphi_{j,k}} f(d_l))$, which is between 0 and 1. Therefore, the availability of all chunks should be close to each other as much as possible. Ideally, the availability of all chunks should be equal to each other. If it is feasible, $\forall k \neq k', f_{c_k} = f_{c_{k'}}$, where f_{c_k} is the failure of kth chunk with r replicas.

Since n is a small constant [1] and the number of replicas, r, is 2 or 3 at most [11], it is possible to search all the problem space in order to find the optimal placement of chunks. Thus, we present the OCP algorithm to find the optimal placement for the replication of chunks as follows.

Algorithm 3 Optimal Chunks Placement

Input: δ, r, m
Output: $CA[k][S]$
1: $S = C(\delta, r)$
2: **Procedure** OCP(S, r, m)
3: **foreach** $(\varphi \in S)$ **do**
4: $PCA[0][\varphi] \leftarrow 1$
5: **end for**
6: **for** $k \leftarrow 1$ to m **do**
7: $CA[k][S] \leftarrow 0$
8: **foreach** $(\varphi \in S)$ **do**
9: $P \leftarrow \forall \varphi' \in S | \forall d_i, d_i \in \varphi' \wedge d_i \notin \varphi$
10: $CA[k][\varphi] \leftarrow A(\varphi) \times PCA[k-1][P]$
11: **if** $(k == 1)$ **then**
12: $PCA[k][P] \leftarrow \max_{\varphi' \in P}(CA(\varphi'))$
13: **end if**
14: **if** $(k > 1$ and $kr < mr)$ **then**
15: $PCA[k][P] \leftarrow$ OCP$(P, r, k-1)$
16: **end if**
17: $CA[k][S] \leftarrow max(CA[k][\varphi], CA[k][S])$
18: **end for**
19: **end for**
20: Return $E[\delta] \leftarrow CA[k][S]$

The way the OCP algorithm works is by computing two functions, namely CA and PCA, each with two entries. One entry for the kth chunk and another one for all possible combinations of size r from δ, denoted by S $(|S| = \binom{|\delta|}{r})$, where δ is a qualified set of data centers which is determined by Algorithm 2. $\varphi \in S$ refers to any element of S, which is an r-combination of δ $(|\varphi| = r)$. In more details, $CA[k][\varphi]$ is the maximum availability of kth chunk with r replicas if its replicas are stored in all $d_i \in \varphi$. Associated to each $\varphi \in S$, P is a subset of S including all elements $\varphi' \in S$, such that φ' does not include any data center $d_i \in \varphi$. $PCA[k][P]$ denotes the maximum availability of k chunks with r replicas that are replicated in P.

We derive a general recursive equation for $CA[k][\varphi]$. First, we enumerate all possible $\varphi \in S$ that could store the kth chunk with r replicas, as if we were placing the kth chunk. Second, we consider all possible placement of the first $(k-1)$ chunks with r replicas, which are placed into P. Thus, if set φ is considered for kth chunks, the availability of k chunks will be the multiplication of the maximum availability from the first $(k-1)$ chunks with r replicas, which are placed into P (i.e., $PCA[k-1][P]$), and the availability of the kth chunk, $A(\varphi)$, when we use φ. Thus, we have:

$$CA[k][\varphi] = A(\varphi) \times PCA[k-1][P]. \tag{15}$$

The computation for $PCA[k][P]$ is a recursive approach as follows. If ($k > 1$ and $kr < mr$), it is assumed that the first chunk with r replicas placed in $\varphi \in S$, and the remaining $(k-1)$ chunks, with duplication factor r should be optimally placed into P. Thus, OCP algorithm called recursively with appropriate parameters. This means OCP$(P, r, k-1)$. Otherwise, if($k = 1$), assumed as the terminal condition of the OCP algorithm, $PCA[1][P]$ is maximum availability of all $\varphi' \in P$ as if the chunk with r replicas is replicated in $\varphi' \in P$. Therefore, the recursive equation for PCA can be obtained as follows.

$$PCA[k][P] = \begin{cases} \max_{\varphi' \in P}(A(\varphi')), \text{if } k = 1 \\ \text{OCP}(P, r, k-1) \text{ if } k > 1 \text{ and } (k \times r < m \times r) \end{cases}$$

Thus, from the derived recursive equations for CA and PCA, Algorithm 3 gives the pseudo-code for the OCP problem.

Similar to the previous section, we apply constraint $E(J_Q) \propto P_Q$ to Algorithm 2. First, budget B is allocated to each QoS type Q proportional to P_Q. That is, $b(J_Q) = \lfloor b \times P_Q \rfloor$, where $b(J_Q)$ is the budget allocated to QoS type Q. To guarantee constraint $E(J_Q) \propto P_Q$, $b(J_Q)$ is sorted decreasingly by $difb(J_Q)$, where $0 \leq difb(J_Q) = b \times P_Q - b(J_Q) < 1$, and then the first $B - \sum_Q b(J_Q)$ of $b(J_Q)$ is increased by one. Second, we substitute B with $b(J_Q)$ in Algorithm 2 to hold the constraint.

TABLE I
OBJECTIVE AND CONSTRAINT OF THE PROPOSED ALGORITHMS

Algorithm	Replication Availability (Expected Availability, EA)	Replication Cost (Budget, B)
Algorithm 1	Limited by EA	Minimize
Algorithm 2	Maximize	Limited by B

V. Performance Evaluation

A. Simulation Setting

We performed several experiments to assess the performance of our algorithms. Table I summarizes the objective and constraint of the proposed algorithms. We first present the parameters setting of cloud providers used in the performance

TABLE II
DATA CENTERS PARAMETERS

DC#	FP	CPO	DC#	FP	CPO
d_1	0.0001	48	d_6	0.004	12
d_2	0.0002	36	d_7	0.01	6
d_3	0.0004	30	d_8	0.04	4
d_4	0.001	24	d_9	0.1	2
d_5	0.002	18			

evaluation. Although the failure probability of most cloud providers are not disclosed, some of them have revealed this parameter. For example, Amazon S3 provides two level of storage services: *Standard Storage* has eleven nines as durability and four nines as availability whilst the other service, *Reduced Redundancy Storage (RRS)*, provides four nines (99.99%) for availability and durability. Storage cost for both services depends on the region of the cloud provider and the level of service. In all regions the storage cost of standard storage is more than that of RRS. Google, another well-known cloud provider, has not disclosed failure probability; however, researchers [7] have done extensive studies on Google's main storage infrastructure and they have measured failure probability, which is about 0.045(that is, 3 nines) on average. Rackspace guarantees 3 nines (99.9%) availability within its SLA. The other providers such as Nirvanix, EMC Atoms, etc. do not disclose failure probability, but they consider credits to compensate availability violation as noted in the SLA.

According to the above description, we determine a set of data centers with two parameters, *Failure Probability (FP)* and *Cost Per Object (CPO)* for our simulation as shown in Table II. Since we have the failure probability of Amazon S3, Rackspace and Google's storage infrastructure, we use them as baseline, and add 6 data centers with the assumption that as availability of service is increased, the storage cost of object is raised [4]. It is also assumed that the cost and the failure probability reported in Table II remains constant during the simulation.

In our simulation, we set the number of objects to 100, while duplication factor r is fixed to 2 since the number of replicas is a small constant in practice (e.g., 2 or 3) [11].

B. Algorithm 1: Minimum Cost Fixed Expected Availability

We have performed experiments to evaluate this algorithm in order to measure the minimum cost of replication whilst the expected availability in the form of number of nines is satisfied. In the first experiment, we relax the constraint (4) and assume that all objects have the same priority from the user's perspective.

The result of this experiment are depicted in Fig. 2. We can say that if the user wants to have the expected availability with 4 *nines*, the minimum cost of replication is imposed to the user is about 600. Thus, if the user randomly chooses an object among the objects, he is able to access that object with the probability of 99.99%. Fig. 2 also demonstrates that the minimum cost of replication experiences almost two stages of significant increment. We observed that the algorithm explores most data centers in the range from d_9 to d_6 to provide 4

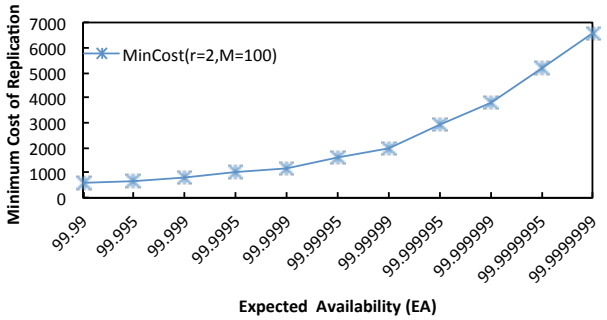


Fig. 2. Minimum cost of replication versus expected availability of objects

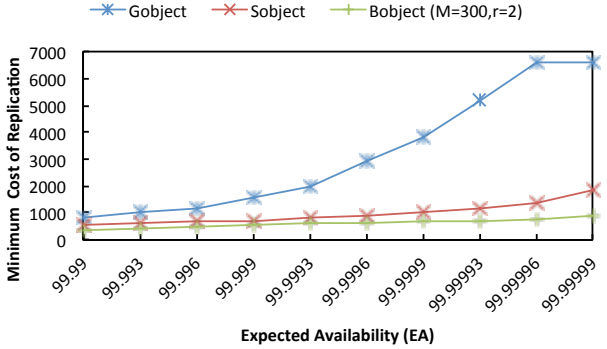
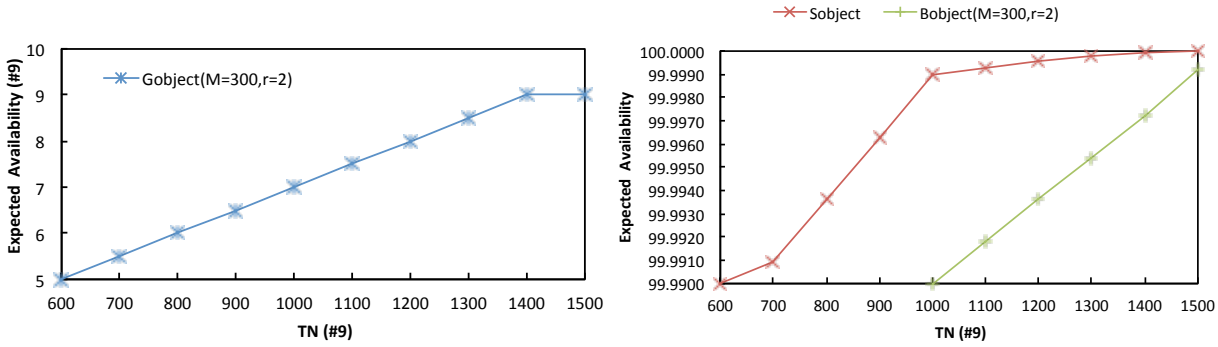


Fig. 3. Minimum cost of replication versus expected availability for three types of QoS

nines and 5 *nines* as expected availability. As the value of EA is increased from 5 to 7 *nines*, the algorithm dynamically switches to the more expensive data centers, where the first stage of increment in the minimum cost of replication happens. The second stage of increment incurs when the value of EA is augmented from 7 *nines* to 9 *nines*, which results in three times increment in the minimum cost of replication.

To evaluate Algorithm 1 with constraints (4) (i.e., the revised Algorithm 1), it is assumed that the user asks to store Gold, Silver and Bronze objects in data centers subject to, for example, $P_G = 50\%$, $P_S = 30\%$ and $P_B = 20\%$. Furthermore, the number of objects in each type of QoS is set to 100. With the above assumptions, the revised Algorithm 1 is run and its results are illustrated in Fig. 3. This figure demonstrates as the EA value is increased, the minimum cost of replication of Gold objects significantly grows compared to that of two other types. For example, with EA=7, the minimum cost of replication for Gold objects is approximately three and seven times more than that of the required for Silver and Bronze objects, respectively. This is because the revised Algorithm 1 explores more reliable data centers, which in turn are more expensive ones, to host Gold objects in comparison with two other QoS types.

In Fig. 4, as expected, the hierarchy between the QoS types is respected, i.e. the value of EA_G is higher than EA_S which in turn is higher than EA_B. For example, when TN=600, we have EA_G=5 *nines* and EA_S=4 *nines* whilst about 60% of Bronze objects are stored in data centers such that EA_B=4 *nines* (which is not plotted in Fig. 4(b)). This is because that Gold and Silver objects have stricter requirements. Figs. 4(a) and 4(b) also show that as the TN



(a) Expected availability versus TN for Gold objects

(b) Expected availability versus TN for Silver and Bronze objects

Fig. 4. Expected availability versus TN for three types of QoS

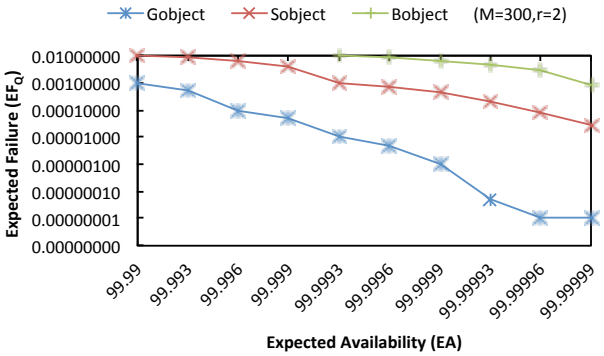


Fig. 5. Expected failure (EF_Q) versus Expected Availability for three types of QoS

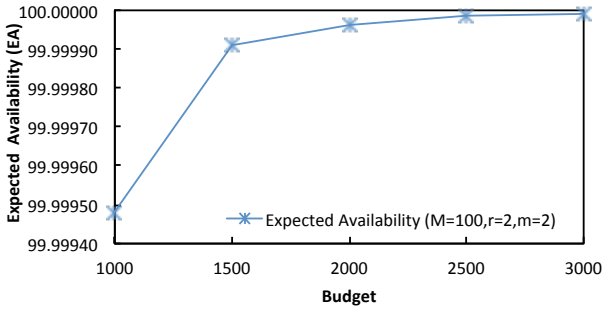


Fig. 6. Expected availability versus Budget

value rises, the value of EA_Q increases according to P_Q. Fig. 5 plots EA against EF_Q. As it can be see, the higher EA value, the lower EF_Q value for three types of QoS. In addition, the value of EF_G is lower than EF_S and EF_B because the value of P_Q prioritizes Gold, Silver and Bronze objects. For example, when EA=7 nines , $EF_G \approx 1.6\ EF_S$ and $EF_G \approx 3\ EF_B$.

C. Algorithm 2: Maximum Expected Availability with a Given Budget

In this algorithm, since n and r are small constants, the value of m is small and varies between 2 and $\lfloor n/r \rfloor$. The value of m can be defined according to a trade-off between the switching cost and the availability of objects. This is because as m is increased the switching cost and the availability of objects are decreased. Finding the optimal value of m based on this trade-off is beyond the scope of this paper, and we leave it as a future work. What it is important for us is to investigate how Algorithms 2 and 3 are able to find optimal placement of chunks for each arbitrary value of m. Therefore, in order to evaluate them, we fix the values of M, r, and m and vary the value of budget. Fig. 6 shows that the value of EA for objects when the budget is varied from 1000 to 3000. The following observations can be made from the results. First, with increasing the budget, the value of EA is increased. This should be attributed to the fact that Algorithm 2 explores the more expensive data centers with lower failure probability to store objects as the budget grows. Second, as the budget is increased, the rate of increment in the value of EA becomes smaller. Because Algorithm 2 chooses a subset of data centers from DC as a *main* set of data centers such that the cost of replication for all objects gained the maximum expected availability is minimized. As the budget escalates, the algorithm dynamically changes some data centers in *main* set, which is termed as *auxiliary* subset, until the budget allows the algorithm to find a new *main* set of data centers. This new *main* set increases the value of EA with a smaller rate. This happens because marginal increment in EA value of using more expensive of data centers in *auxiliary* subset is decreasing. As a result, the expected availability is increased less with the same amount of additional budget. For example, when the budget increases from 1000 to 1500, the value of EA increases one nine that is the same as that of from 1500 to 2500. Fig. 7 illustrates the EF value for objects versus the budget. As it can be seen, the EF value is decreased with the increase of budget. Furthermore, the decrement in the value of EF becomes less when we have an increment in the budget, which confirms the second observation in Fig. 6.

Fig. 8 plots the value of EA_Q against the budget that is varied between 2000 and 6000. The higher budget results in the higher value of EA_Q for all types of QoS. As expected, the Gold objects achieve the highest expected availability and Bronze objects achieve the lowest one, because the revised Algorithm 3 divides the budget among each type of QoS according to P_Q. Fig. 8 also depicts that EA_B value is constant when the budget is varied from 2000 and 5000. The reason is since the revised algorithm allocates the lowest budget to

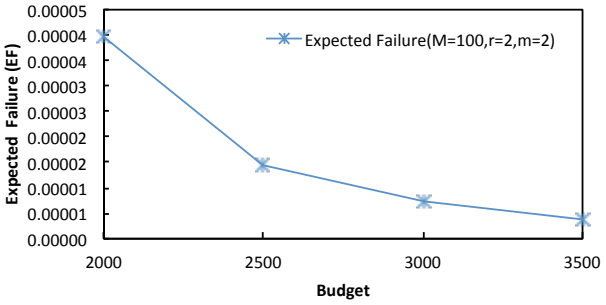


Fig. 7. Expected failure versus Budget

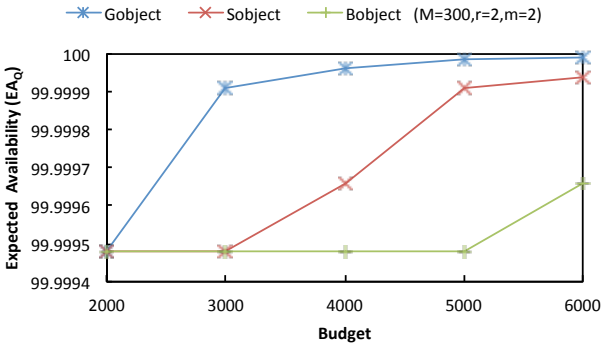


Fig. 8. Expected availability for three types of QoS Versus Budget

Bronze objects, it is not possible to store all Bronze objects in the data centers. In this experiment, 60%, 40% and 20% Bronze objects are not placed in data centers when the budget is 2000, 3000, and 4000, respectively. This figure for Silver objects is about 40% and 20% when the budget is 2000 and 3000, respectively. Placing all Silver and Bronze objects happens when the amount of budget reaches 3000 and 5000, respectively.

VI. Conclusions and Future Works

We addressed two issues related to placing replicas of the objects in multi-cloud environment. In order to tackle these issues, we propose efficient algorithms. The first algorithm has been designed to minimize the replication cost and the expected availability of objects as the user's QoS is met. The second one is proposed to maximize the expected availability of objects under a given budget with the assumption that the objects are split to chunks. We also have conducted extensive simulation experiments to evaluate the effectiveness of our algorithms. The experiments show that the proposed algorithms are efficient to determine the optimal location of the replicas for objects with a given constraint.

As our future work, we will first propose an algorithm to find the minimum replication cost with a given expected availability for striped objects. Second, since in this work we have conducted trade-off between the storage cost and the availability, we aim to consider the data transfer in/out cost, the queries and the processing cost when we select the cloud provider. Third, the cost of migration from one cloud provider to another due to urgent needs should also be taken into consideration in selection of cloud storages.

Acknowledgment

The authors would like to thank Rodrigo N. Calheiros, Deepak Poola, Nikolay Grozev and Atefeh Khosravi for their helpful discussions and suggestions.

References

[1] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, 2009.

[2] R. Kotla, L. Alvisi, and M. Dahlin, "Safestore: a durable and practical storage system," in *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ser. ATC'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 10:1–10:14.

[3] S. S. G. L. Ryan Ko, "Cloud computing vulnerability incidents: A statistical overview," 2013.

[4] C.-W. Chang, P. Liu, and J.-J. Wu, "Probability-based cloud storage providers selection algorithms with maximum availability," *2012 41st International Conference on Parallel Processing*, vol. 0, pp. 199–208, 2012.

[5] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: a case for cloud storage diversity," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 229–240.

[6] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14.

[7] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association, 2010, pp. 1–7.

[8] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 205–216.

[9] M. Placek and R. Buyya, "Storage exchange: a global trading platform for storage services," in *Proceedings of the 12th international conference on Parallel Processing*, ser. Euro-Par'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 425–436.

[10] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai, "Are clouds ready for large distributed applications?" *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 18–23, Apr. 2010.

[11] M. J. Fischer, X. Su, and Y. Yin, "Assigning tasks for efficiency in hadoop: extended abstract," in *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, ser. SPAA '10. New York, NY, USA: ACM, 2010, pp. 30–39.