Latency-Sensitive Data Allocation for Cloud Storage

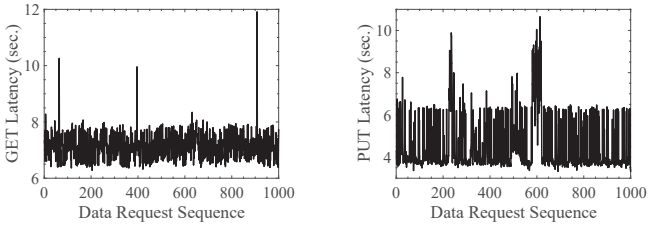
Song Yang[1], Philipp Wieder[1], Muzzamil Aziz[1], Ramin Yahyapour[1,2] and Xiaoming Fu[2]
[1]Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttingen, Germany
[2]Institute of Computer Science, Göttingen University, Germany
{Song.Yang, Philipp.Wieder, Muzzamil.Aziz, Ramin.Yahyapour}@gwdg.de, Fu@cs.uni-goettingen.de

***Abstract*—Customers often suffer from the variability of data access time in cloud storage service, caused by network congestion, load dynamics, etc. One solution to guarantee a reliable latency-sensitive service is to issue requests with multiple download/upload sessions, accessing the required data (replicas) stored in one or more servers. In order to minimize storage costs, how to optimally allocate data in a minimum number of servers without violating latency guarantees remains to be a crucial issue for the cloud provider to tackle. In this paper, we study the latency-sensitive data allocation problem for cloud storage. We model the data access time as a given distribution whose Cumulative Density Function (CDF) is known, and prove that this problem is NP-hard. To solve it, we propose both exact Integer Nonlinear Program (INLP) and Tabu Search-based heuristic. The proposed algorithms are evaluated in terms of the number of used servers, storage utilization and throughput utilization.**

I. Introduction

Cloud storage (e.g., Amazon S3, Windows Azure, Google Cloud Storage) is emerging as a business solution for remote data storage due to its features in ubiquitous network access, low maintenance, elasticity and scalability. The current cloud storage systems have successfully provided data storing and accessing services to both enterprises and individual users. For example, Netflix, one of the most popular Internet video-streaming providers, has put all its content on Amazon S3 storage [1]. Another example is the cloud storage tool such as Dropbox or Google drive, which can enable individuals to store their data on the cloud and access it anywhere over the Internet. It is reported that the registered users in dropbox have raised to 400 million by 2015, with daily 1.2 billion uploaded files [2]. It can be expected that more and more enterprises and individuals will transfer their data workloads to the cloud in the future due to the increasing capital expenditures for maintaining private infrastructures.

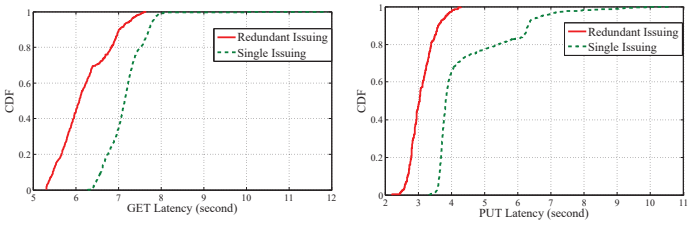
In current cloud storage systems, the data access time or latency is usually uncertain [3] because of network congestion, load dynamics, disk I/O interference, maintenance activity, etc. [4], [5]. For example, for a 32 Mb data file in Amazon S3, we measure the data GET (download) and data PUT (upload) latencies among 1000 requests. Fig. 1 shows that the data access time is random and dynamic for both GET and PUT operations. As a result, the uncertainty of data access time deteriorates the Quality of Service (QoS) to customers and affects the end user's experience, which may reduce the number of customers and hence the profit of the cloud providers [6]. Therefore, how to guarantee reliable latency-sensitive service remains to be a crucial issue for the cloud provider to tackle.



(a) GET Latency (b) PUT Latency

Fig. 1: Latency for 32 Mb data file among 1000 requests in Amazon S3: (a) GET latency (b) PUT latency.

According to [5], [7], to deal with the latency uncertainty so as to guarantee a reliable latency-sensitive storage service, one way is to concurrently issue each request with multiple sessions, and use the earliest response from those sessions. The redundant sessions can be accommodated by one or more replicas on different servers, which mainly depends on server's I/O rate as we will explain later. For example, we conduct experiments on Amazon S3 to concurrently use three replicas of a 32 Mb data file to issue 1000 sequential requests. Fig. 2 shows that this approach can efficiently decrease latency, compared to the approach via no replica provisioning (single issuing) as shown in Fig. 1, although at the cost of increased network resource (e.g., bandwidth).



(a) CDF of GET Latency (b) CDF of PUT Latency

Fig. 2: Cumulative Density Function (CDF) of accessing 32 Mb data file in Amazon S3 with 3 replicas (redundant) issuing and no replicas (single) issuing.

Another approach is to increase throughput by increasing

978-3-901882-89-0 @2017 IFIP

the I/O rate so as to reduce latency. For example, different on-demand I/O [8] parameters can be selected by customers in Google Cloud Storage service. Accordingly, a higher I/O rate provisioning may decrease latency but also charges higher since it consumes more computing resources. On the other hand, we should also guarantee that all the demands' volume on one certain server should not exceed its maximum working load, otherwise the overloaded server will become a bottleneck and thereby rendering response time significantly higher. For example, in [9], for 32 Mb block size in a VM on Windows Azure, it is measured that the total response time over 1000 requests is more than 10 seconds at 50 reads/sec and 20 write/sec in write requests, compared to 20 reads/sec and 10 writes/sec, respectively.

Similar to [10], [11], we formulate the uncertain GET/PUT latency for each storage server as a given distribution based on its historical data. In this paper, we use data access to refer to data GET or PUT operation, and make no difference between latency, data access time and response time. We focus on how to allocate data file(s) on minimized number of servers to concurrently issue the data requests, such that for each request, the probability of accessing a data file within its requested response time is no less than a specified value. Our key contributions are as follows:

- We present a latency-sensitive cloud storage service model (Section II).
- We propose the Latency-Sensitive Data Allocation (LSDA) problem, and analyze its complexity (Section III).
- We propose both exact and heuristic solutions to solve the LSDA problem (Section IV).
- We evaluate the proposed algorithms in terms of performance via simulations (Section V).

II. Latency-Sensitive Cloud Storage Service Model

We assume that a data chunk is the basic data storing unit, whose size is represented by b and hence, it cannot be further partitioned. In this context, the size of any data file d can be represented by $b \cdot |d|$, where $|d|$ represents the number of chunks that constitute it. For simplicity, we assume that one whole/entire data file can only be placed in one server, i.e., it cannot be separated and stored on separate servers. The notations used in this paper are summarized in Table I.

We consider a heterogeneous cloud storage system consisting of a set of $|S|$ servers S. For a particular server $s \in S$ in the cloud storage system, $L(s)$ denotes its maximum affordable I/O rates or maximum working/server load without degrading the performance and $C(s)$ represents its storage limit. Suppose there are g possible data access I/O rates, a_1, a_2, ..., a_g. We assume that the data access (GET or PUT) time follows a given distribution [10], [11]. Therefore, its Cumulative Density Function (CDF) $f_s^b(x)$[1] represents the probability that one data chunk can be retrieved at most x time units. Under the server's maximum working load, the CDF is dependent on server load in some sense, but it does not change much, e.g., when the server load increases, the latency may be a bit higher. To deal with this issue, we only adopt the "conservative" CDF for when the server's working load is close to its maximum working load, i.e., the "upper bound" of latency distribution function. By doing this, (1) the latency constraint can be maximally guaranteed, although this comes at the expenses of a bit higher server usage than optimum. (2) The problem complexity is largely reduced, otherwise we need latency probability functions corresponding to different degrees of server loads. We use the term "session" to represent one GET/PUT thread or process to serve a data request from a data file located/placed in one server to the user's end. In this context, multiple sessions can be established to issue one or more data requests on one server (say s), but the total consumed I/O rates should not exceed $L(s)$.

[1]It is worthy to mention that $f_s^b(x)$ does not mean the failure probability and we also do not consider the server failure probability in this paper.

TABLE I: Notations.

Notation	Description
s, S	The set of $\|S\|$ storage servers and one server $s \in S$
$L(s)$	Maximum working load of server s
$C(s)$	Storage limit of server s
$f_s^b(x)$	The probability that one data chunk b can be retrieved within time x on server s
$R(d, T, \delta, \alpha)$	The set of data requests R. For each $r(d, T, \delta, \alpha) \in R$, T indicates the requested data access time for data file d with size $\|d\|$, δ implies the requested probability of retrieving d within time T, and α represents requested I/O rate
N_s	The number of established sessions on server s
γ	The parameter used for determining the next loaded server to empty in TSDA
M	Maximum times to call Intensification function in TSDA
μ	The parameter used for controlling TSDA whether to call Diversification function or find next loaded server

Without loss of generality, for a data request $r(d, T, \delta, \alpha)$, T indicates the requested data access time for data file d with size $|d|$, δ implies the requested probability of retrieving d within time T, and α represents the requested I/O rate. Suppose r has been issued by k servers (replicas), where N_s sessions are established on server s. Let us use $f_s^d(x)$ to denote the probability of accessing d within time x on server s. Consequently, the total probability of accessing d within T is:

$$1 - \prod_{s=1}^{k} \left((1 - f_s^d(T))^{N_s} \right) \tag{1}$$

where $(1 - f_s^d(T))^{N_s}$ denotes the unsuccessful probability of accessing D within time T on server s for N_s sessions, and $\prod_{s=1}^{k} \left((1 - f_s^d(T))^{N_s} \right)$ is the unsuccessful probability of accessing D within time T on k servers for all their respective sessions. As a result, $1 - \prod_{s=1}^{k} \left((1 - f_s^d(T))^{N_s} \right)$ indicates the probability of at least one session on one server can access D within time T.

Let's take an example to better illustrate it, where we do not differentiate GET and PUT for simplicity. Given two servers

A and B, and their Probability Density Function (PDF) of accessing a data file d ($|d| = 20$ Gb) within time x (in ms) follow:

$$f_A^d(x) = \begin{cases} 0.9 : x \leq 10 \\ 0.05 : 10 < x \leq 15, \\ 0.05 : x > 15 \end{cases} \quad f_B^d(x) = \begin{cases} 0.75 : x \leq 6 \\ 0.2 : 6 < x \leq 10 \\ 0.05 : x > 10 \end{cases}$$

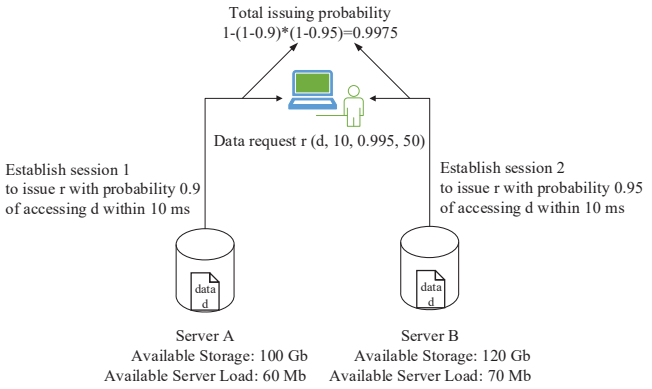
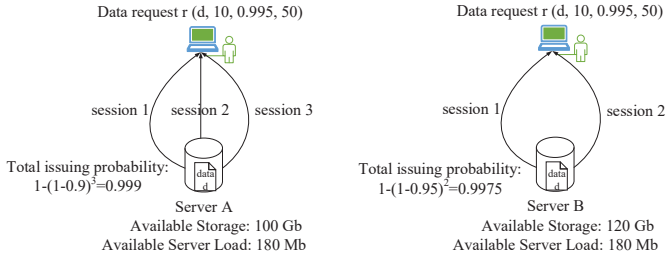


Fig. 3: Placing data on two servers using one session on each server to satisfy accessing data latency probability.



(a) Establishing 3 sessions on A (b) Establishing 2 sessions on B

Fig. 4: Placing data on single server using multiple sessions to satisfy accessing data latency probability.

We first assume server A has storage 100 Gb and available server load 60 Mb/s, and server B has storage 120 Gb and available server load 70 Mb/s. Suppose there arrives a request r to access data file d under I/O rate $\alpha = 50$ Mb/s with $T = 10$ ms, and $\delta = 0.995$. According to their PDFs, placing d on server A alone can guarantee a probability 0.9 with latency no less than 10 ms, and placing d on server B solely can promise probability 0.95 with latency at most 10 ms. Since only one session can be established on each of servers (either $60 < 2 \cdot 50 = 100$ or $70 < 2 \cdot 50 = 100$), placing D on either server A or B cannot satisfy the requested δ. On the other hand, as Fig. 3 shows that simultaneously placing d on both A and B leads to a probability of $1-(1-0.9)\cdot(1-0.95) = 0.995$ with latency at most 10 ms, which can meet the requested probability.

Next, let's consider the scenario when each server can simultaneously support multiple sessions for data access. Since CDF $f_s^b(x)$ is calculated based on server s' historical latency performance, it already includes the scenario when multiple sessions may access the shared resources such as buffers on the same server. Therefore each session on server A can statically guarantee a probability of accessing data file within time 10 at least 0.9. In this sense, Eq. (1) still holds for multiple sessions to be established on the same server. For example, we assume that both server A and B have available server load 180 Mb/s. In this sense, to satisfy the requested latency probability, we do not need to place d on two servers. Instead, we can issue the request with only one copy stored in any server but more sessions are needed. For instance, on server A, three sessions are needed to simultaneously to accommodate the request (shown in Fig. 4(a)), since the probabilities of using two sessions and three sessions are $1-(1-0.9)^2 = 0.99 < \delta$ and $1-(1-0.9)^3 > \delta$, respectively. On the other hand, on server B, two sessions are needed (shown in Fig. 4(b)) since $1-(1-0.95)^2 = 0.9975 > \delta$. From the above example we see that in order to satisfy the latency probability constraint, more sessions are sometimes needed to establish from different data copies/locations (Fig. 3) or the same data location (Fig. 4). Clearly, the latter case can save more storage space.

III. Problem Definition and Analysis

A. Latency-Sensitive Data Allocation

Formally, the Latency-Sensitive Data Allocation (LSDA) problem can be defined as follows:

Definition 1: Assume a cloud storage system consisting of a set of $|S|$ servers S. For a server $s \in S$, $L(s)$ denotes its maximum affordable working load or I/O rates without degrading the performance, and $C(s)$ represents its storage limit. Given a set of R data requests $\mathcal{R}$, for a request $r(d, T, \delta, \alpha) \in \mathcal{R}$, T indicates the requested data access time for data file d with size $|d|$, δ implies the requested probability of retrieving d within time T, and α represents the requested I/O rate. The Latency-Sensitive Data Allocation (LSDA) problem for cloud storage is to place all the requested data $d \in \Delta$ in minimized number of servers without violating each server's storage and maximum working load, such that for each request $r(d, T, \delta, \alpha)$, the probability of accessing data d within time T is at least δ.

In the LSDA problem, we mention that if a data file d_m for one request r_1 is placed on a server s, when another request r_2 which stills requests d_m is issued by the server s, there is no need to place d_m on server s again. In this sense, issuing r_1 and r_2 by server s only consumes one time of storage $|d_m|$ but consumes the sum of their requested I/O rates. Moreover, for a certain data GET/PUT request, perhaps more than one redundant sessions are established to issue it, the client will use the first finished response from those redundant sessions and then drop the other sessions. This will of course consume more network resource such as bandwidth, but the specified probability of accessing data within requested latency can be guaranteed.

Theorem 1: The LSDA problem is NP-hard.

Proof: Let's first introduce the Bin-Packing problem: Given n items with sizes e_1, e_2, ..., e_n, and a set of m bins with capacity c_1, c_2, ..., c_m, the Bin-Packing problem is to pack all the items into minimized number of bins without violating the bin capacity size. The Bin-Packing problem is proved NP-hard [12].

Assume for each request $r(d, T, \delta, \alpha) \in \mathcal{R}$, placing d on any single server (say s) is enough to guarantee that $f_s^d(T) \geq \delta$. We also assume that each server has enough storage capacity but each server has limited I/O rate. That is to say, the data access latency constraint and server's storage constraint are not considered. Now, if we map/regard the item with its size in the bin packing problem to the request with its requested I/O rate in the LSDA problem, respectively, the LSDA problem is equivalent to the bin-packing problem and hence is NP-hard. ■

IV. Exact and Heuristic Algorithms

A. Exact Solution

In this subsection, we propose an exact solution to solve the LSDA problem. We start by some necessary notations and variables:

$\mathcal{R}(T, d, \alpha, \delta)$: The set of R requests.

Δ: The set of the requested data files.

S: The set of servers.

$L(s)$ and $C(s)$: The maximum working load and storage limit of server s, respectively, where $s \in S$.

$CDF_s^{d,\alpha}(x)$: The CDF for server s for accessing (GET or PUT operation which depends on the request) data with size d for I/O rate α.

N_s: Maximum number of sessions that can be established on server $s \in S$ for one request.

$P_{s,i}^r$: A boolean value indicating whether request r is accommodated by placing its requested data d on server s and is served by i-th session.

Y_s^d: A boolean value indicating whether data $d \in \Delta$ is stored in server $s \in S$.

Objective:

$$\min \quad \sum_{s \in S} \max_{d \in D} Y_s^d \tag{2}$$

Data request time probability constraint:

$$1 - \prod_{s \in S} \prod_{i=1}^{N_s} \left(1 - P_{s,i}^r \cdot CDF_s^{|d|,\alpha}(T)\right) \geq \delta \quad \forall r(T, d, \alpha, \delta) \in \mathcal{R} \tag{3}$$

I/O rate constraint:

$$\sum_{r(T,d,\alpha,\delta) \in \mathcal{R}} \sum_{i=1}^{N} P_{s,i}^r \cdot \alpha \leq L(s) \quad \forall s \in S \tag{4}$$

Determine data allocation on a specific server:

$$Y_{d'}^s = \max_{1 \leq i \leq N, r \in \mathcal{R}} P_{s,i}^r \quad \forall d' \in \Delta, s \in S, \text{where } r.d = d' \tag{5}$$

Server capacity constraint:

$$\sum_{d \in \Delta} Y_d^s \cdot |d| \leq C(s) \quad \forall s \in S \tag{6}$$

Eq. (2) minimizes the number of total used servers. For instance, we first calculate the maximum value of Y_s^d for server $s \in S$, and as long as $Y_s^d = 1$ for some $d \in \Delta$, it means that server s in use to store some data d. After that, we take the sum of $\max_{d \in D} Y_s^d$ for each server $s \in S$ and try to minimize this value. Eq. (3) ensures that for each request $r(T, d, \alpha, \delta) \in \mathcal{R}$, the probability of accessing data d within time T is at least δ. More specifically, $P_{s,i}^r \cdot CDF_s^{|d|,\alpha}(T)$ denotes the probability of accessing D within time T on server s by establishing i-th session. In this sense, by taking into account all N_s possible sessions on each server $s \in S$, Eq. (3) represents that at least one session on one server $s \in S$ has a probability of accessing D within time T no less than δ. Eq. (4) ensures that the total consumed I/O rates on each server $s \in S$ does not exceed its maximum working load. Eq. (5) determines whether a data file $d \in \Delta$ is placed on server $s \in S$. Eq. (6) ensures that each server does not violate its storage limit.

B. Heuristic

Algorithm 1 TSDA $(S, \mathcal{R}, M, T, \mu, \gamma)$

1: Sort the servers in decreasing order according to $m(s)$
2: Accommodate each request with each server that first fits in the order. In case servers are not enough, dummy servers are created with tiny value of $m(s)$.
3: $P \leftarrow \emptyset$, $L_i \leftarrow \emptyset$, $L_s \leftarrow \emptyset$, $k \leftarrow 0$ and store this initial solution into L_i.
4: Select the target loaded server s_t with minimum value of $\frac{l}{L} + \frac{c}{C} + \gamma \frac{r}{R}$
5: **While** time limit T is not reached **do**
6: $fd \leftarrow$ false;
7: $A(S) =$Search $(S, M, s_t, fd, L_i, L_s)$
8: **If** $fd == true$ **then**
9: $k \leftarrow 0$; In case there exists a server s_u satisfying $m(s_u) < m(s_t)$, then use s_t to host all the requests from s_u and empty s_u if possible.
10: **Else if** $fd == false$ and $k < \mu \cdot |A(S)|$ **then**
11: $k++$;
12: **Else** Call Diversification$(A(S), L_i, L_s, P)$
13: Determine next loaded server s_t with minimum $\frac{l}{L} + \frac{c}{C} + \gamma \frac{r}{R}$
14: return $\min(P)$.

In this subsection, we propose a Tabu Search-based heuristic to solve the LSDA problem. Tabu search is an advanced local search algorithm, which is introduced by Glover [13], [14].

Algorithm 2 Search $(S, M, s_t, fd, L_i, L_s)$

1: counter $\leftarrow 0$
2: **While** counter $\leq M$
3: **Foreach** request r in s_t
4: Try to issue it with the server(s) which already stores its required data file, otherwise use first fit server to host the request. The server allocation should not be in L_s. Denote $A(S)$ as the result of server allocation for the requests.
5: **If** all the requests have been accommodated by current loaded servers **then**
6: $fd \leftarrow true$; Return $A(S)$
7: **Else**
8: Call Intensification($A(S), L_i, L_s, s_t$)
9: counter++
10: return $A(S)$

Algorithm 3 Intensification $(A(S), L_i, L_s, s_t)$

1: Sort the loaded servers in $A(S)$ except for s_t according to $\frac{c_s}{C(s)} \cdot \frac{l_s}{L(s)}$ in decreasing order.
2: Assign the first 20% of servers to Group $G1$, and put all the other servers in Group $G2$.
3: Use one server in $G1$ to issue the accommodated request(s) from one server in $G2$ to if possible.
4: Swap accommodated request(s) between two servers in $G1$ if possible.
5: Record the L_s with the original data placement in each server before its change.

Algorithm 4 Diversification $(A(S), L_i, L_s, P)$

1: $P \leftarrow A(S)$.
2: Remove from solution $A(S)$ the $\frac{1}{2}$ servers with smallest $u(s)$ value, and issue each request from a removed server with each empty server. The $1:1$ request to server allocation should not be in the Tabu list L_i.
3: Reset L_s to empty and add the initial solution to L_i.

The local search algorithm starts from an initial solution, and improves this solution by moving to a better neighbor solution iteratively. It will stop if the current solution cannot be further optimized. Different from local search, Tabu search allows the solution to deteriorate in the searching procedure. By keeping track of recent moves in a so-called Tabu list, cyclic moves in the search space are banned in order to save running time. Moreover, Tabu Search can be enhanced by (1) Intensification: to (slightly) modify the current neighbor solution to encourage move combinations and make search region more attractive and (2) Diversification: to guide the search toward the unexplored areas of search space.

Our proposed heuristic, called Tabu Search-based Data Allocation (TSDA) algorithm, is presented in Algorithm 1. In Step 1, we first sort the servers based on the following equation in decreasing order.

$$m(s) = \frac{C(s) \cdot L(s)}{\mu(s)} \tag{7}$$

where $C(s)$ and $L(s)$ imply the capacity and maximum load of server s, and $\mu(s) = \int_0^\infty (1 - CDF(x))d(x)$ denotes the mean value of a given distribution based on its cumulative distributed function. In this context, a bigger $m(s)$ value indicates that server s is "good" in terms of having greater capacity and working load, and "faster" response time. Following that, in Step 2, each request is accommodated by the first suitable server in that order. We store this initial solution in T_i in Step 3, and define T_s to be the Tabu list during the searching procedure. In particular, T_s stores which request cannot be issued by which server. We also define a queue P to store the solution that cannot be further optimized during searching round (local optimal). k is initially equal to 0 and denotes the times that the current solution is not improved after calling Search function in Algorithm 2. After that, we first select one server with the minimum value of the metric in Eq. (8):

$$u(s) = \frac{c_s}{C(s)} + \frac{l_s}{L(s)} + \gamma \frac{r_s}{R} \tag{8}$$

where c_s and l_s represent the total consumed capacity and I/O rates by all the data requests stored on it, r_s denotes the number of requests that is accommodated by it, and γ is an user specified parameter. Therefore, a bigger $u(s)$ indicates that this server is well utilized, and a smaller $u(s)$ value implies that this server is not well utilized. As long as the time limit T is not reached, Step 5-Step 13 is going to search a solution with a minimum number of servers to issue all the requests. In Step 6, fd is initially set to false which represents whether the selected server is emptied. The general idea of Steps 7-13 is to first select a not well-utilized server s_t according to Eq. (8), and try to use other current loaded servers to issue all the requests from s_t (Algorithm 2 which we will specify later). More specifically, if calling Algorithm 2 in Step 7 returns a feasible solution, then k is set to 0 and s_t can be emptied in Step 9. In case a "better" server s_t is emptied, s_t will replace the current loaded server s_u which has a smaller metric in terms of Eq. (7) than it if possible. If s_t cannot be emptied but k is less than $\mu \cdot |A(S)|$, where μ is a fractional number and $|A(S)|$ denotes the current total number of used servers, k is increased by 1 in Step 11. Otherwise we call Diversification function in Algorithm 4 in Step 12. The general idea of Diversification function is first to store the current found solution $A(S)$ in P. Then it empties half of the (not well-utilized) servers and accommodates each of those requests from those emptied servers by each empty server. We then select next server s_t to be emptied in Step 13. When the time limit is exceeded, in Step 14, we return a so-far best solution from P.

Algorithm 2 performs how to empty a selected server, by using current loaded servers to host all the requests from s_t. In Step 1, a variable counter is set to 0 and denotes the maximum times that it can call Intensification function. As

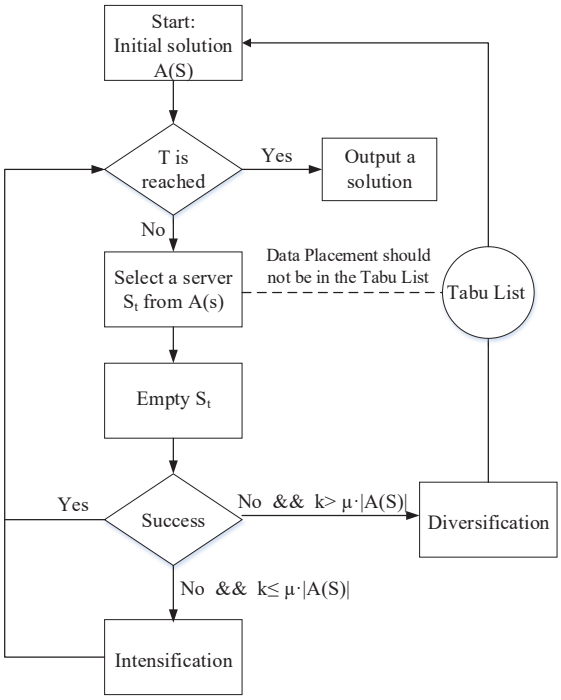


Fig. 5: An overview of TSDA.

long as it is less than M, for each request r which is originally accommodated by s_t, Step 4 searches for one (or more) current loaded server(s) to issue it. If all the requests originally issued by s_t have been accommodated by the other servers, then we return this solution in Step 6. Otherwise, we call Intensification function in Algorithm 3 and increase counter value by 1. In Algorithm 3, if a data file d which is stored in one server s_1 in group $G1$ or $G2$ has been accessed by multiple requests, then we prefer to swap or move its whole associated requests. Otherwise we only "swap" or "move" single request issued by the servers in different groups. In all, Fig. 5 depicts an overview of TSDA.

V. Simulation-based Evaluation

A. Simulation Setup

Due to the lack of experimental resources, we cannot measure the maximum working loads and CDF of data access time for dozens of servers in cloud storage environment. Instead, we simulate a cloud storage system consisting of 30 servers. The storage capacity and maximum working load of each (free) server are assumed to be 1 TB and 1.26 Gb [2], respectively. There are in total 40 data files of 5 types. The size in Gb is (randomly) chosen from one of the intervals: $[1,1]$, $[4,9]$, $[25,40]$, $[45,60]$, $[95,110]$, for each type respectively. By doing this, we want data sizes are different from each other in order to make the optimal solution not easy to find. The requested I/O rate is in the set of $\{5,10,20,50,100\}$ in Mb/s. We randomly generate 100, 200 and 300 requests. In order that at most $N_s = 5$ sessions are enough for accommodating any data request on any empty server $s \in S$, the simulation parameters are set like this: for each request $r(d,T,\delta,\alpha)$, d and α are randomly generated. T is chosen among $[8,20]$, $[40,100]$, $[250,600]$, $[500,900]$ and $[800,2000]$ in ms for aforementioned 5 types of data files, respectively. δ is selected among $[50\%,70\%]$, $[70\%,85\%]$, $[80\%,90\%]$, $[85\%,95\%]$ and $[90\%,98\%]$ for $\alpha = 5,10,20,50$, and 100 Mb/s, respectively. For simplicity, we assume that the GET latency and PUT latency of a data file follow a totally identical distribution on the same sever. We assume three kinds of distributions for data access time of the server, namely (1) exponential distribution, (2) geometric distribution and (3) uniform distribution. For each distribution, different parameters are assigned to each server in order that these servers are "heterogeneous" for simulation. More specifically, in the exponential distribution $1-e^{-\lambda x}$, λ is chosen from the intervals $[0.01,0.03]$, $[0.04,0.06]$, $[0.08,0.1]$, $[0.12,0.15]$ and $[0.2,0.25]$ for $\alpha = 5,10,20,50$, and 100 Mb/s, respectively. In the uniform distribution $\frac{x-\alpha}{\beta-\alpha}$, $\alpha = 0$ and β is chosen from the intervals $[20,25]$, $[16,20]$, $[12,16]$, $[11,14]$ and $\in [8,13]$ for $\alpha = 5,10,20,50$ and 100 Mb/s, respectively. In the geometric distribution $1-(1-p)^x$, p is selected from the intervals $[0.07,0.09]$, $[0.1,0.12]$, $[0.12,0.15]$, $[0.15,0.18]$ and $[0.18,0.2]$ for $\alpha = 5,10,20,50$ and 100 Mb/s, respectively. According to [16], the data access time is increasing approximately linearly with the data size. Therefore, for simplicity, we assume that if a data file consisting of c data chunks stored in server s, it follows that $CDF_s^{cb}(x) = CDF_s^b(\frac{x}{c})$, where $b = 1$ Gb in this context.

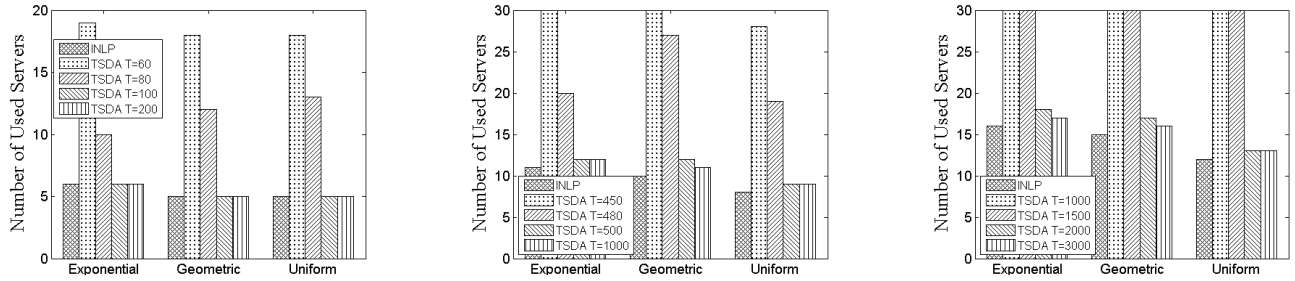
The simulation is run on a desktop PC with 2.7 GHz and 8 GB memory. We use IBM ILOG CPLEX 12.6 to implement the proposed INLP, but we found it is very time consuming to return a (final) solution because of its nonlinear constraint in Eq. (3). For example, for exponential distribution when $R = 100$, the INLP keeps on running for one day and still does not terminate. In order to let the INLP return the result in a reasonable time, we set a running time limit of 1, 2 and 3 hours[3] for 100, 200 and 300 requests, respectively. We use C# to implement the proposed heuristic TSDA. From our simulations on TSDA for the LSDA problem, we found that it starts to return a "reasonable" solution at $T = 60,450,1000$ ms for when requests' amount $R = 100$, 200 and 300, respectively, and it ends in returning a relatively stable solution at $T = 100$, 500 and 2000, respectively. Therefore, we set the time limit $T = 60,80,100,200$ ms for $R = 100$, $T = 450,480,500,1000$ ms for $R = 200$, and $T = 1000,1500,2000,3000$ ms for $R = 300$. Moreover, we set $M = 30$, $\gamma = 10$ and $\mu = 0.8$.

B. Simulation results

We first evaluate the algorithms in terms of the number of used servers. From Fig. 6, we see that the INLP and TSDA

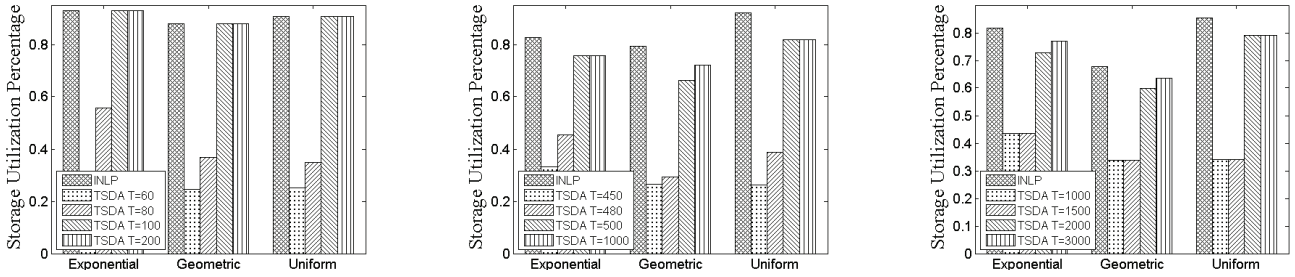
[2]We use a fractional number 0.7 in [9] to multiply the referred value 1.8 in [15], which leads to $1.8 * 0.7 = 1.26$ Gb/s=1260 Mb/s.

[3]We also set longer running time (e.g., 10 hours) for the INLP for 200 and 300 requests. We found that the results are the same.



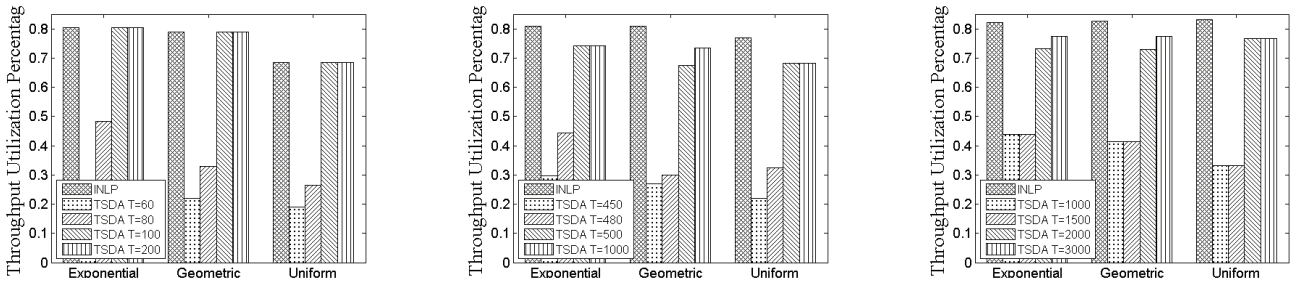
(a) R=100 (b) R=200 (c) R=300

Fig. 6: The number of used servers for different number of traffic requests (R): (a) $R = 100$ (b) $R = 200$ (c) $R = 300$.



(a) R=100 (b) R=200 (c) R=300

Fig. 7: Storage utilization percentage for different number of traffic requests (R): (a) $R = 100$ (b) $R = 200$ (c) $R = 300$.



(a) R=100 (b) R=200 (c) R=300

Fig. 8: Throughput utilization percentage for different number of traffic requests (R): (a) $R = 100$ (b) $R = 200$ (c) $R = 300$.

for $T = 100, 200$ have the same performance when $R = 100$. Except for that, the INLP always consumes the minimum number of servers for different amount of requests, which also validates its correctness. The number of servers consumed by TSDA decreases when the time limit T increases, and TSDA can achieve a close-to-optimal performance with the exact INLP when $(T, R) = \{(200, 100), (1000, 200), (3000, 300)\}$. In particular, TSDA performs poorly when the time limit T is not sufficient enough. For example, when $T = 450$ and $R = 200$ for exponential distribution scenario, TSDA uses more than 30 servers. Since there are in total 30 available servers to issue requests, the extra servers are dummy servers according to Alg. 1. For comparison reason, if more than 30 servers are consumed by TSDA, we regard that it consumes 30 servers.

Next, we compare the algorithms with respect to (1) storage utilization percentage: (total used storage in Gb) divided by ($1000*$ number of used servers), and (2) Throughput utilization percentage: (total consumed I/O rates in Mb/s) divided by ($1260*$ number of used servers). Fig. 7 and 8 plot the performance of INLP and TSDA in terms of these two metrics. As expected, the INLP obtains the maximum utilization of storage and throughput (or equal utilization with TSDA for $T = 100, 200$ when $R = 100$), since it consumes a minimum number of servers (or equal number of used servers with TSDA for $T = 100, 200$ when $R = 100$) as shown in Fig. (6). The value achieved by TSDA increases when T grows. We see that it obtains the same or close performance with INLP when $T = 200, 1000, 3000$ for $R = 100, 200, 300$, respectively. In all, the exact INLP can always achieve the best performance, so it can be used when the computation speed is not a big concern. This is because its running time

will increase exponentially when the problem size grows. On the other hand, TSDA can be a preferred choice especially for when data requests arrive in a bursty manner, since its running time (≤ 3 sec.) is significantly less than INLP (≥ 1 hour) and it can obtain a close-to-optimal performance.

VI. Related Work

Lin *et al.* [17] study the QoS-aware Data Replication (QADR) problem, which is to find an optimal data replica placement strategy, such that both the total replication cost of all data blocks and the total number of QoS-violated (i.e., data access time) data replicas are minimized. Lin *et al.* [17] first formulate the QADR problem as an ILP. Subsequently, they transform the QADR problem to the Min-Cost Max-Flow problem and solve it in polynomial time. However, Lin *et al.* [17] assume that the data access time is a deterministic value, which is not the case in reality as we mentioned earlier in Fig. 1. Bai *et al.* [18] deal with how to place data replicas in cloud storage systems without violating latency constraint. It is assumed that the expected service time of a data file d on a server s is calculated as $\frac{|d|\cdot R_s}{NTC_s}$, where R_s denotes the number of concurrent requests on s for d, and NTC_s represents the network transmission capability of server s. Subsequently, they propose a graph-based heuristic to allocate data replicas with data access latency guarantee. However, using expected value to model "uncertain" or "probabilistic" data access time cannot accurately or comprehensively reflect the "uncertainty" of data access time in realistic cloud storage systems. On the contrary, this paper adopts the cumulative density function to model uncertain data access time, which is more practical.

Liu *et al.* [11] tackle the deadline-guaranteed data reallocation problem in cloud storage. They first formulate this problem as an INLP formulation, with the objective of minimizing both the number of used servers and traffic load caused by replication through a network. It is assumed each server is modeled as an M/M/1 queuing system to serve requests. They calculate upperbound $\lambda^g_{s_n}$ of service rate for server s_k to guarantee that each request from tenant t_k has a latency no longer than d_{d_k} and its realizing probability is no less than P_{t_k}. Subsequently, they propose the Parallel Deadline Guarantee (PDG) scheme, which dynamically reallocates data from an overloaded server to an underloaded server to guarantee that, for each tenant t_k, at most ϵ_{t_k} percentage of all requests have service latency larger than a given deadline d_{t_k}. Hu *et al.* [10] address how to determine the smallest number of servers under a two interactive job classes model. Assuming the job arrival process is Poisson with rate λ and service time distribution is exponential with mean $\frac{1}{\mu}$. They propose a Probability dependent priority (PDP) scheduling policy to maximize the probability of meeting a given response time goal. However, in reality, the arrival rate of traffic requests such as bursty traffic does not always follow Poisson process and the service times are also not always exponential [19]. The solutions of [10], [11] become invalid for bursty traffic and non-exponential service time.

You *et al.* [9] study the load management problem in cloud storage. They first formulate an exact ILP to solve the optimization problem, that transfers the data from overloaded servers to underloaded servers by using minimum bandwidth migration costs. They further speed up the proposed ILP by reducing the unnecessary number of its decision variables, and applying Divide-and-Conquer and relaxation techniques. Cheng *et al.* [20] implement an Elastic Replication Management System (ERMS). ERMS can automatically manage the replication number of severs based on data's access frequency. However, [9], [20] do not take data access time into account when dealing with data (replication) placement.

VII. Discussion

We highlight that in reality, the user does not have control over the servers but the cloud provider makes decisions on how to allocate data (files) on servers. The purpose of the addressed LSDA problem in this paper is to provide cloud provider a cost-efficient solution to deal with data allocation issues without violating latency constraints. We could also apply the LSDA problem in the scenario when traffic requests arrive asynchronously. For instance, traffic requests arrive at different times during a day in realistic application. In that context, the network lifetime can be partitioned into a set of equivalent-length time slots[4] and on each time slot we can have a set of traffic requests. Consequently, the cloud provider can perform the proposed algorithms in order to get solutions on how to allocate requested date files at each time slot.

Moreover, considering that duplicating data may be time-consuming especially for large data files, so for time-sensitive application, data allocation should be done before the requests in the next period arrive, otherwise it may incur additional waiting time for customers and hence prolong the data access time. In that case, we suggest to apply traffic prediction technique [21] such as Markov chain [22] and Auto Regressive (AR) process [23] to retrieve traffic requests as the input of the LSDA problem.

VIII. Conclusion

In this paper, we have studied the Latency-Sensitive Data Allocation (LSDA) problem. Under the assumption that the data access time follows a given distribution and its CDF is known, we have proved that the LSDA problem is NP-hard. Subsequently, we propose an exact Integer Nonlinear Program (INLP) and a Tabu Search-based heuristic to solve it. Simulation results reveal that the exact INLP can always achieve the best performance in terms of the number of used servers, storage utilization percentage and throughput utilization percentage. The Tabu Search-based heuristic, on the other hand, can achieve a close-to-optimal value with the INLP, but its running time is much less than the INLP.

Acknowledgement

This work has been funded by EU FP7 ITN CleanSky (No. 607584).

[4]Each time slot can for instance span one minute, two minutes, etc. which depends on the real application.

REFERENCES

[1] "Netflix finishes its massive migration to the amazon cloud." [Online]. Available: http://arstechnica.com/information-technology/2016/02/netflix-finishes-its-massive-migration-to-the-amazon-cloud/

[2] "By the numbers: 12 interesting dropbox statistics." [Online]. Available: http://expandedramblings.com/index.php/dropbox-statistics

[3] G. Liang and U. C. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 2012–2025, 2014.

[4] S. L. Garfinkel and S. L. Garfinkel, "An evaluation of amazons grid computing services: EC2, S3, and SQS," in *Technical Report TR-08-07*. Harvard University, 2007.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[6] "Amazon found every 100ms of latency cost them 1% in sales." [Online]. Available: http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/

[7] Z. Wu, C. Yu, and H. V. Madhyastha, "Costlo: Cost-effective redundancy for lower latency variance on cloud storage services," in *Proc. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[8] "Google storage: On-demand I/O." [Online]. Available: https://cloud.google.com/storage/docs/on-demand-io/

[9] G.-W. You, S.-W. Hwang, and N. Jain, "Ursa: Scalable load and power management in cloud storage systems," *ACM Transactions on Storage (TOS)*, vol. 9, no. 1, pp. 1–29, 2013.

[10] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 2009, pp. 101–111.

[11] G. Liu, H. Shen, and H. Wang, "Deadline guaranteed service for multi-tenant cloud storage," *IEEE Transactions on Parallel and Distributed Systems*, no. 99, pp. 1–14, 2016.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co., 1979.

[13] F. Glover, "Tabu search-part i," *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[14] ——, "Tabu searchpart ii," *ORSA Journal on computing*, vol. 2, no. 1, pp. 4–32, 1990.

[15] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the internet with nano data centers," in *Proceedings of the 5th ACM international conference on Emerging networking experiments and technologies*, 2009, pp. 37–48.

[16] M. Kwon, Z. Dou, W. Heinzelman, T. Soyata, H. Ba, and J. Shi, "Use of network latency profiling and redundancy for cloud server selection," in *7th IEEE International Conference on Cloud Computing (CLOUD)*, 2014, pp. 826–832.

[17] J.-W. Lin, C.-H. Chen, and J. M. Chang, "Qos-aware data replication for data-intensive applications in cloud computing systems," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 101–115, 2013.

[18] X. Bai, H. Jin, X. Liao, X. Shi, and Z. Shao, "RTRM: A response time-based replica management strategy for cloud storage system," in *Grid and Pervasive Computing*. Springer, 2013, pp. 124–133.

[19] K. Salah, K. Elbadawi, and R. Boutaba, "An analytical model for estimating cloud resources of elastic services," *Journal of Network and Systems Management*, pp. 1–24, 2015.

[20] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, N. Zhang, and G. Guan, "ERMS: an elastic replication management system for HDFS," in *IEEE International Conference on Cluster Computing Workshops (CLUSTER WORKSHOPS)*, 2012, pp. 32–40.

[21] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[22] S. Pacheco-Sanchez, G. Casale, B. Scotney, S. McClean, G. Parr, and S. Dawson, "Markovian workload characterization for QoS prediction in the cloud," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2011, pp. 147–154.

[23] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 119–128.