IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 23, 2018, accepted November 20, 2018, date of publication November 27, 2018, date of current version December 27, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2883674

Latency-Sensitive Data Allocation and Workload Consolidation for Cloud Storage

SONG YANG[1], (Member, IEEE), PHILIPP WIEDER[2], MUZZAMIL AZIZ[2], RAMIN YAHYAPOUR[2,3], XIAOMING FU[3], (Senior Member, IEEE), AND XU CHEN[4], (Member, IEEE)

[1]School of Computer Science, Beijing Institute of Technology, Beijing 100081, China
[2]Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, 37077 Göttingen, Germany
[3]Institute of Computer Science, University of Göttingen, 37077 Göttingen, Germany
[4]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China

Corresponding author: Song Yang (S.Yang@bit.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61802018 and Grant U1711265, in part by the Beijing Institute of Technology Research Fund Program for Young Scholars, in part by the EU FP7 Marie Curie Actions CleanSky Project under Grant 607584, and in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X355.

ABSTRACT Customers often suffer from the variability of data access time in (edge) cloud storage service, caused by network congestion, load dynamics, and so on. One efficient solution to guarantee a reliable latency-sensitive service (e.g., for industrial Internet of Things application) is to issue requests with multiple download/upload sessions which access the required data (replicas) stored in one or more servers, and use the earliest response from those sessions. In order to minimize the total 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 deal with. In this paper, we study the latency-sensitive data allocation problem, the latency-sensitive data reallocation problem and the latency-sensitive workload consolidation problem for cloud storage. We model the data access time as a given distribution whose cumulative density function is known, and prove that these three problems are NP-hard. To solve them, we propose an exact integer nonlinear program (INLP) and a Tabu Search-based heuristic. The simulation results reveal that the INLP can always achieve the best performance in terms of lower number of used nodes and higher storage and throughput utilization, but this comes at the expense of much higher running time. The Tabu Search-based heuristic, on the other hand, can obtain close-to-optimal performance, but in a much lower running time.

INDEX TERMS Cloud Storage, data allocation, latency, workload consolidation.

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 in the cloud and access it anywhere over the Internet. It is reported that the number of the registered users of dropbox has 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. This uncertainty also exists in mobile edge computing or Internet of Things (IoT) network because of wireless transmitting channel attenuation, radio access

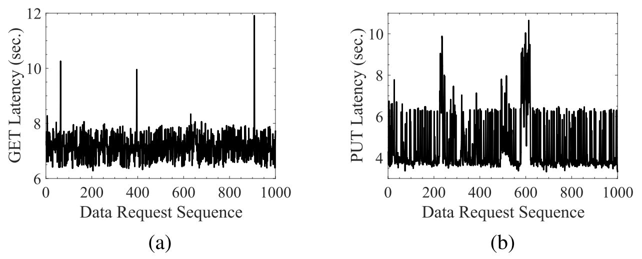


FIGURE 1. Latency for 32 Mb data file among 1000 requests in Amazon S3: (a) GET latency (b) PUT latency. (a) GET latency. (b) PUT latency.

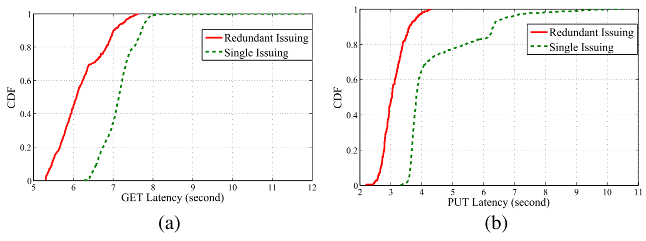


FIGURE 2. Cumulative Density Function (CDF) of accessing 32 Mb data file in Amazon S3 with 3 replicas (redundant) issuing and no replicas (single) issuing. (a) CDF of GET latency. (b) CDF of PUT latency.

control [6], etc. 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 [7], [8]. Therefore, how to guarantee a reliable latency-sensitive service remains to be a crucial issue for the cloud provider to tackle.

According to [5], [9], and [10] , 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 this comes at the cost of the increased network resource (e.g., bandwidth).

Another approach is to increase throughput by increasing the I/O rate so as to reduce latency. For example, different on-demand I/O parameters can be selected by customers in Google Cloud Storage service [11]. Accordingly, a higher I/O rate provisioning may decrease latency but also charges higher since it consumes more computing resources. Nevertheless, we need to guarantee that all the demands' volume on one certain server should not exceed its maximum workload, otherwise the overloaded server will become a bottleneck and thereby rendering response time significantly higher. For example, in [12], for a data block with a size of 32 Mb in a VM on Windows Azure, it is measured that the total data access time over 1000 requests is more than 10 seconds at 50 reads/sec and 20 writes/sec, compared to 20 reads/sec and 10 writes/sec, respectively.

Similar to [13] and [14], 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. In [15], we studied how to allocate data file(s) on a minimum 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. In this paper, we extend our work [15] to further study (i) latency-sensitive data reallocation problem and (ii) latency-sensitive workload consolidation problem. The addressed problems in this paper can be applied to and benefit industrial fields and applications. For instance, the cloud provider can implement the proposed algorithm in a "software-defined" manner (similar to the concept of software-defined storage in [16]). In this way, based on data accessing requests, the requested data files will be placed on the minimized number of servers (saving electricity bills for the enterprise) without violating the requested delay constraint. Moreover, our work can also be applied to Internet-of-Things (IoT) related scenarios. For instance, the requested data can also refer to the Industrial Internet-of-Things (IIoT) data, which are collected from e.g., wireless sensors. Since the IIoT service has a severe latency requirement [17], [18], our proposed algorithms can guide how to place the requested data in a minimum number of edge and/or cloud servers without violating the delay constraints. Our key contributions are as follows:

- We present a latency-sensitive cloud storage service model.
- We propose the Latency-Sensitive Data Allocation (LSDA) problem and its two variants, and analyze their complexities.
- We propose both exact and heuristic solutions to solve all the proposed problems.
- We evaluate the proposed algorithms in terms of performance via simulations.

The remainder of this paper is organized as follows: Section II presents the related work. In Section III, we present the proposed latency-sensitive cloud storage service model. In Section IV, we propose the Latency-Sensitive Data Allocation (LSDA) problem, the Latency-Sensitive Data Reallocation (LSDR) problem, and the Latency-Sensitive Workload Consolidation (LSWC) problem. We also analyze their complexities in this section. In Section V, we propose an exact Integer Nonlinear Program (INLP) and a Tabu Search-based heuristic to solve each of these 3 problems. Section VI provides our simulation results, and we conclude in Section VII.

II. RELATED WORK

A survey about data replication techniques in the cloud environment can be found in [19]. Mansouri *et al.* [20] propose a survey that covers key aspects of cloud-based data storage issues, e.g., data model, data consistency, data management, etc. Wu *et al.* [21] conduct a comprehensive measurement study of the performance (e.g., delay, failure rates, etc.) from five popular cloud storage service providers based on their provided Web APIs.

Lin *et al.* [22] 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.* [22] 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.* [22] 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.* [23] 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. Kumar *et al.* [24] present an exact and a greedy algorithm to solve the latency-aware data placement problem in distributed Cloud-based Internet of Things (IoT) Networks, in which the IoT gateways and distributed clouds are connected. Boukhelef *et al.* [25] propose two cost based object placement strategies (a genetic-based heuristic and an ad-hoc heuristic based on incremental optimization) for hybrid storage systems, which consists of the solid state drives and hard disk drives. However, the data access time is not considered in [24] and [25]. Liu *et al.* [26] target how to allocate data items on different datacenters with minimized prices such that the requested data access time is obeyed. They propose a genetic algorithm to solve this problem.

In order to minimize the latency variation, Arumugam *et al.* [27] present a dynamic I/O redirection and caching mechanism called VirtCache. Shekhar *et al.* [28] propose an online data-driven model which applies Gaussian Processes-based machine learning techniques to predict the performance of the system under different levels of interference. They subsequently utilize the proposed model to scale resources such that performance interference is minimized and latency requirements of applications are satisfied.

Liu *et al.* [14] 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 that 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.* [13] address how to determine the smallest number of servers under a two interactive job classes model. Suppose that 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 the Poisson process and the service times are also not always exponential [29]. The solutions of [13] and [14] become invalid for bursty traffic and non-exponential service time.

TABLE 1. Notations.

Notation	Description
s, S	The set of $\|S\|$ storage servers and one server $s \in S$
$L(s)$	Maximum workload of server s
$C(s)$	Storage limit of server s
$f^b_s(x)$	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

III. THE 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 a whole/entire data file can only be placed on one server, i.e., it cannot be separated and stored on separate servers. The notations used in this paper are summarized in Table 1.

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 workload without degrading the performance and $C(s)$ represents its storage limit. Suppose there are g possible data access I/O rates, $a_1, a_2, \ldots, a_g$. We assume that the data access (GET or PUT) time follows a given distribution [13], [14]. Therefore, its Cumulative Density Function (CDF) $f^b_s(x)$ represents the probability that one data chunk can be retrieved at most x time units.[1] Under the server's maximum workload, 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 workload is close to its maximum workload, i.e., the "upper bound" of the latency distribution function. By doing this, (1) the latency constraint can be maximally guaranteed, although this comes at the expense 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 $L(s)$.

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:

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

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 that at least one session on one server can access D within time T.

Let us take an example to better illustrate it, where we do not differentiate GET and PUT for simplicity. Suppose there are two servers A and B, and their Probability Density Functions (PDF) of accessing a data file d ($|d| = 20$ Gb) within time x (in ms) follow:

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

We first assume that server A has a storage of 100 Gb and an available server load of 60 Mb/s, and server B has a storage of 120 Gb and an available server load of 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

[1] It is worthwhile 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.

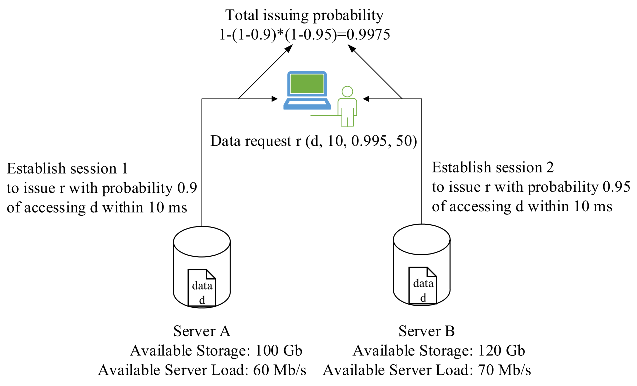

Total issuing probability
1-(1-0.9)*(1-0.95)=0.9975
Data request r (d, 10, 0.995, 50)
Establish session 1
to issue r with probability 0.9
of accessing d within 10 ms
Establish session 2
to issue r with probability 0.95
of accessing d within 10 ms
data d
data d
Server A
Available Storage: 100 Gb
Available Server Load: 60 Mb/s
Server B
Available Storage: 120 Gb
Available Server Load: 70 Mb/s

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

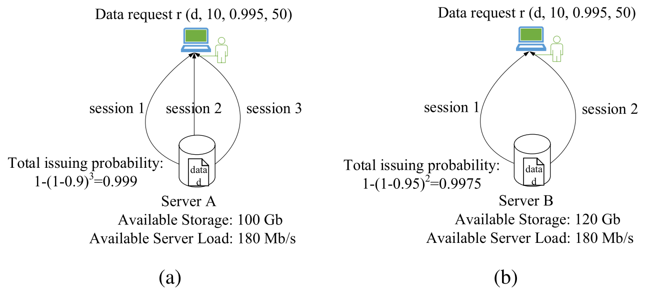

Data request r (d, 10, 0.995, 50)
session 1
session 2
session 3
Total issuing probability:
1-(1-0.9)^3=0.999
data d
Server A
Available Storage: 100 Gb
Available Server Load: 180 Mb/s
(a)
Data request r (d, 10, 0.995, 50)
session 1
session 2
Total issuing probability:
1-(1-0.95)^2=0.9975
data d
Server B
Available Storage: 120 Gb
Available Server Load: 180 Mb/s
(b)

FIGURE 4. Placing data file on single server using multiple sessions to satisfy accessing data latency probability. (a) Establishing 3 sessions on *A*. (b) Establishing 2 sessions on *B*.

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, 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 us 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 the case when multiple sessions to be established on the same server. For example, we assume that both server A and B have an available server load of 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 to be established. For instance, on server A, three sessions are needed to simultaneously 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 $1 - (1 - 0.9)^3 > \delta$. Similarly, 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.

IV. PROBLEMS DEFINITION AND ANALYSIS

A. LATENCY-SENSITIVE DATA ALLOCATION

Definition 1: Assuming that a cloud storage system consists of a set of $|S|$ servers S. For a server $s \in S$, $L(s)$ denotes its maximum affordable workload or I/O rates without degrading the performance, and $C(s)$ represents its storage limit. Given a set of R data requests $\mathcal{R}$, the Latency-Sensitive Data Allocation (LSDA) problem for cloud storage is to place all the requested data $d \in \Delta$ in a minimum number of servers without violating each server's storage and maximum workload, such that for each request $r(d, T, \delta, \alpha) \in \mathcal{R}$, 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 1: Let us first introduce the Bin-Packing problem: Given n items with sizes $e_1, e_2, \ldots, e_n$, and a set of m bins with capacity $c_1, c_2, \ldots, c_m$, 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 [30].

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 time 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.

B. LATENCY-SENSITIVE DATA REALLOCATION

In realistic cloud storage applications, the data accessing frequency often varies with time. For instance, for a certain data file (e.g., on-line video resource), we call it "hot data" when it is accessed much more frequently than the average, and call it "cold data" when it is seldom visited. Consequently, when a data file becomes "hot", the current servers that store it will be overloaded, leading to a performance degradation in terms of latency. To solve it, we need to place more of this data file' replicas on some other (empty) servers. When a data becomes "cold", we could also reduce/remove some of its replicas in order to save more capacity. Formally, the Latency-Sensitive Data Reallocation (LSDR) problem can be defined as follows:

Definition 2: Given is a set of servers $S_e \subseteq S$ which store some existing data file Δ_e. For a set of $|R|$ data requests $\mathcal{R}$, the Latency-Sensitive Data Reallocation (LSDR) problem for cloud storage is to place all the requested data Δ in a minimum number of servers $S_x \subseteq S$, such that:

1) $\exists s \in S \backslash S_e$ if $s \in S_x$, then $S_e \subset S_x$
2) For each request $r(d, T, \delta, \alpha) \in \mathcal{R}$, the probability of accessing data d within time T is at least δ.

Constraint 1 indicates that we need to firstly allocate already used servers in S_e to issue requests by placing data files in order to better utilize storage resource. Only when all the servers in S_e cannot be placed by more data files, we could place requested data on servers in $S \backslash S_e$. More specifically, suppose servers S_e are in use to store some data files Δ_e for the current being. In next time period, when a set of requests R arrive, we need to try to use those $|S_e|$ servers to accommodate it. If those $|S_e|$ servers are not enough, we need to add a minimum number of servers in $S \backslash S_e$ together with S_e to issue all the requests. If $|S_e|$ servers are over-provisioned for R, we could use a portion of the servers in S_e to issue it, and "turn off" the rest of servers in S_e in order to save storage costs or reduce the extra maintaining expenditures (e.g., energy). Considering that duplicating data may be time consuming especially for large data files, the data reallocation should be done before the requests in the next period arrive, otherwise it will incur more waiting time for customers and hence prolong the data access latency. We suggest to apply traffic prediction such as Markov chain [31], Auto Regressive (AR) process [32], etc. However, we omit the details about those traffic prediction mechanisms since it is out of the scope of this paper. We refer readers for more details about traffic prediction to [33]. Instead, we assume in the LSDR problem that, $\mathcal{R}$ is known or given.

Theorem 2: The LSDR problem is NP-hard.

Proof 2: The LSDR problem is equivalent to the LSDA problem when $S_e = \emptyset$. Since the LSDA problem is NP-hard according to Theorem 1, the LSDR problem is therefore NP-hard.

C. LATENCY-SENSITIVE WORKLOAD CONSOLIDATION

The aforementioned LSDA and LSDR problem only deal with how to place data files in a cost-efficient manner. For instance, in the LSDR problem, two servers are assumed to have sufficient free capacity and each of them stores only one different data file. When these two data files are accessed by requests, we have to use these two servers to accommodate the requests, even though these two servers are not well utilized. In order to further improve servers' utilization in terms of capacity, we could also move some data file(s) from a not well-utilized server to a well-utilized server, and switch off the empty server. This is known as workload consolidation. Formally, the Latency-Sensitive Workload Consolidation problem can be defined as follows:

Definition 3: Given is a set of servers S, and each server $s \in S$ stores some data files to accommodate $|R_s|$ requests. The Latency-Sensitive Workload Consolidation (LSWC) problem is to use a minimum number of a subset of servers in S to provision all the requests without violating their latency requirement such that the total data moving cost is no greater than MC.

Here the data moving cost MC for a data file is calculated as the data size multiplied by a coefficient γ. For simplicity, we set $\gamma = 1$ in the LSWC problem.

Theorem 3: The LSWC problem is NP-hard.

Proof 3: The LSWC problem is equivalent to the LSDA problem when $MC = +\infty$. Since the LSDA problem is NP-hard according to Theorem 1, the LSWC problem is therefore NP-hard.

V. EXACT AND HEURISTIC ALGORITHMS

A. EXACT SOLUTION FOR THE LSDA AND LSDR PROBLEM

In this subsection, we propose an exact solution to solve both the LSDA problem and the LSDR problem. Since the LSDA problem is a special case of the LSDR problem, we propose a general INLP to solve the LSDR problem, which in turn can also solve the LSDA problem. We start by some necessary notations and variables:

INLP notation:

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

Δ: The set of the requested data files.

S: The set of servers.

Δ_e: The set of data files already stored in existing servers S_e, where $S_e \subseteq S$.

$L(s)$ and $C(s)$: The maximum workload 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.

$A[s][d]$: A given boolean array value indicating whether server $s \in S$ has already stored data file d.

INLP variable:

$P_{s,i}^r$: 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:

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

Constraints:

Data request time probability constraint:

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

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:

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

Server allocation constraint:

$$A[s][d] \cdot Y_s^d \geq Y_{s'}^d \quad \forall d \in \Delta_e,\ s \in S_e,\ s' \in S \backslash S_e \tag{7}$$

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 is 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 $s \in S$ do not exceed its maximum workload. 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. Eq. (7) ensures that the server which stores existing data file should be firstly used to accommodate the request. More specifically, when the current loaded servers S_e are not sufficient to issue the requests, Eq. (7) ensures to firstly allocate the servers in S_e to accommodate the requests as many as possible. After that, the INLP in Eqs. (2)-(6) will assign (minimum) new empty servers to accommodate the rest of the requests. On the other hand, when the traffic requests can be issued by less than $|S_e|$ servers, Eq. (7) ensures to allocate some of existing $|S_e|$ servers to serve all the requests and set the variable $Y[d][s]$ to be 0 for the other servers in S_e.

In all, Eqs. (2)-(7) can solve the LSDR problem. Meanwhile, when $S_e = \emptyset$ and all the entities in $A[s][d]$ are 0 (the input of the LSDA problem), Eqs. (2)-(7) can solve the LSDA problem as well.

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$(P)$.
```

B. EXACT SOLUTION FOR THE LSWC PROBLEM

To solve the LSWC problem, we additionally define the following INLP notations.

$H[r][s]$: A given/known array indicating whether request $r \in \mathcal{R}$ is originally issued by the server $s \in S$.

MC: The data moving cost upperbound.

Migration cost constraint:

$$\sum_{r\in\mathcal{R}, s\in S} (1 - H[r][s]) \cdot (\max_{i=1}^{N_s} P^r_{s,i}) \cdot r.|d| \leq MC \qquad (8)$$

Eq. (8) ensures that the total data moving cost is less than the specified. As a result, Eqs. (2)-(7) together with Eq. (8) can solve the LSWC problem exactly, where $S_e = \emptyset$ and all the entities in $A[s][d]$ are 0.

C. HEURISTIC

In this subsection, we propose a Tabu Search-based heuristic to solve all 3 proposed problems in Section IV. We first solve the LSDA problem. Tabu search is an advanced local search algorithm, which is introduced by Glover [34], [35]. The local search algorithm starts with 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.

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 $\eta$ percentage 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 $G2$ 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 $L_i$.
```

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 a decreasing order.

$$m(s) = \frac{C(s) \cdot L(s)}{\mu(s)} \qquad (9)$$

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 workload, 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. (10):

$$u(s) = \frac{c_s}{C(s)} + \frac{l_s}{L(s)} + \gamma \frac{r_s}{R} \tag{10}$$

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 are 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 be 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. (10), 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. (9) 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 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.

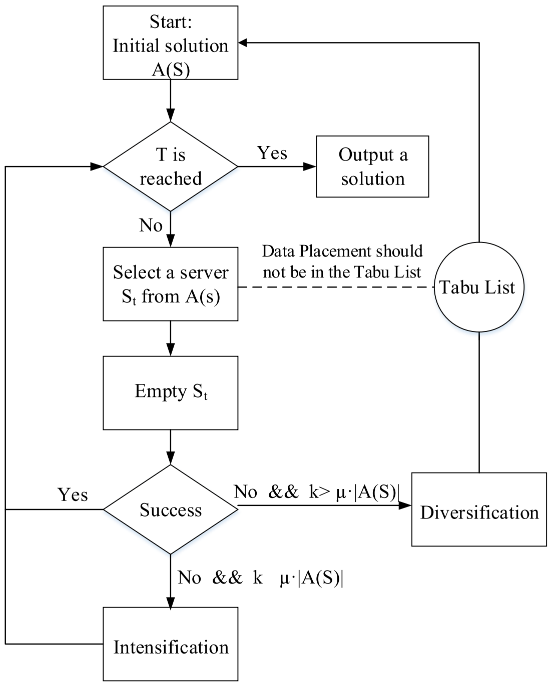


FIGURE 5. An overview of TSDA.

We could slightly modify Algorithm 1 (and keep all the others same) to solve the LSDR problem. Denote S_e as the set of servers that currently store data files, and S_g as the set of all the other free servers such that $S_e \cup S_g = S$. The (minor) modification is on selecting a target server to empty in Step 2 and 13. More specifically, in Step 2, for request $r(T, d, \alpha, \delta)$, we need to firstly allocate the server which already stores d as an initial solution, otherwise we select a server from S_g. In Step 13, $s \in S_e$ can be selected as a targeted server to be emptied only when none of the servers in S_g store data in the solution $A(S)$. The reason is that we must guarantee that servers in S_e should be firstly used to issue the requests. When they are not sufficient enough, we use other servers to issue the rest of the requests.

To solve the LSWC problem, we can still use Algorithm 1. The only difference is that the initial solution stored in L_i on Step 3 in Algorithm 1 is $|S|$ used servers together with $R - |S|$ dummy servers and each server is accommodating one request.

VI. SIMULATION-BASED EVALUATION

A. SIMULATION SETUP

We simulate a cloud storage system consisting of 30 servers, in which the simulation setup is in line with the real cloud storage latency performance as in [21]. The storage capacity and maximum workload of each (free) server are assumed to be 1 TB and 1.26 Gb/s,[2] respectively. There are in total 40 data files of 5 types. The size in Gb is (randomly) chosen

[2]We use a fractional number 0.7 in [12] to multiply the referred value 1.8 in [36], which leads to $1.8 * 0.7 = 1.26$ Gb/s=1260 Mb/s.

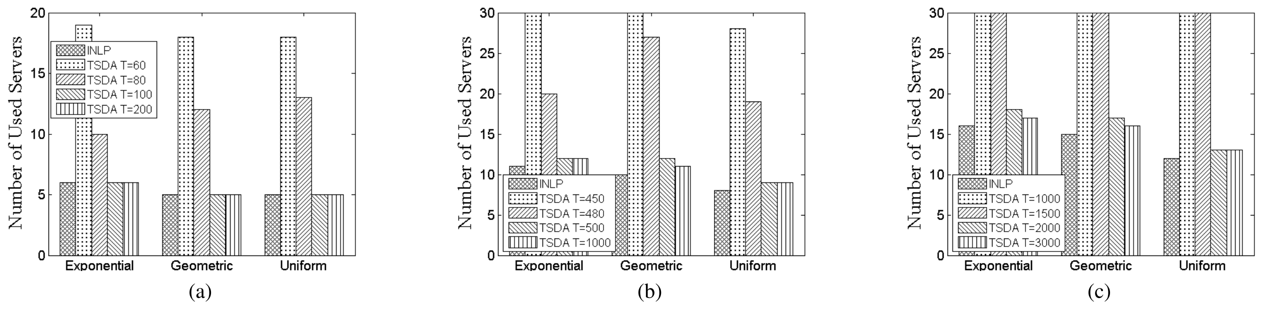


FIGURE 6. The number of used servers in the LSDA problem for different number of traffic requests (R): (a) $R = 100$ (b) $R = 200$ (c) $R = 300$.

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 5 aforementioned 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. 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 $\frac{x-\tau}{\beta-\tau}$, $\tau = 0$ and β is chosen from the intervals [20, 25], [16, 20], [12, 16], [11, 14] and [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 [37], 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 is 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 above simulation setup is for the LSDA problem. On basis of it, for the LSDR problem, we randomly choose 7 out of 30 servers to store 5 out of 40 data files. The storage and maximum workload of these "opened" servers are set to 700 Gb and 700 Mb/s, respectively.

The simulations are run on a desktop PC with 2.7 GHz and 8 GB memory. We use an Intel(R)Core(TM)i5-4310M CPU 2.70GHz x64-based processor in our simulations. 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 $R = 300$, 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 3 hours for $R = 300$ requests. TSDA is implemented by C# and compiled on Visual Studio 2015 (using .NET Framework 4.5). 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 $R = 100$, 200 and 300, respectively, and it ends by 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 $R = 300$. We also found a similar trend for the LSDR and the LSWC problem, but for simplicity, we only show the results of TSDA in the LSDR problem and the LSWC problem for $R = 100, 200, 300$ when $T = 200, 1000, 3000$, respectively. Moreover, we set $M = 30$, $\gamma = 10$, $\eta = 20$ and $\mu = 0.8$.

B. SIMULATION RESULTS FOR THE LSDA PROBLEM

We first evaluate the algorithms in terms of the number of used servers. From Fig. 6, we see that the INLP and TSDA 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 regard to (1) storage utilization percentage: (total used storage in Gb) divided by (1000∗ number of used servers in S_g+700∗ number of used

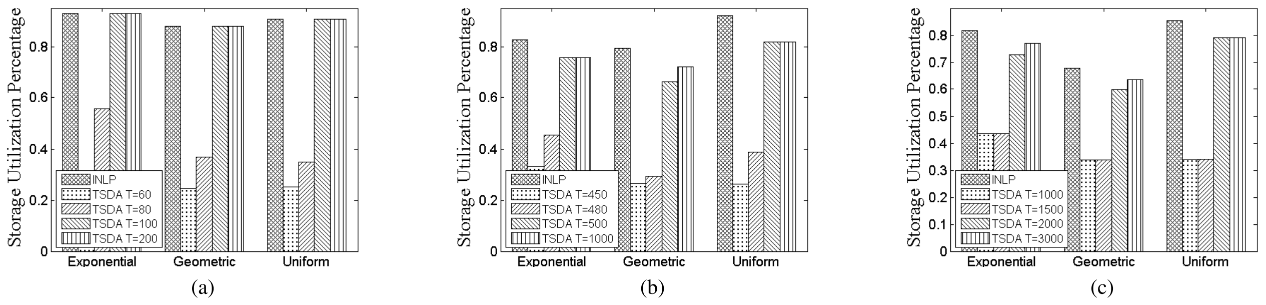


FIGURE 7. Storage utilization percentage in the LSDA problem for different number of traffic requests (R): (a) $R = 100$ (b) $R = 200$ (c) $R = 300$.

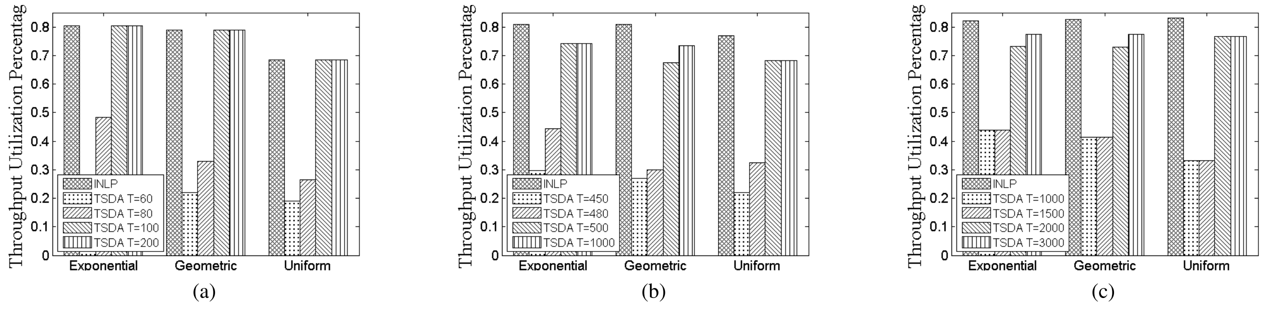


FIGURE 8. Throughput utilization percentage in the LSDA problem for different number of traffic requests (R): (a) $R = 100$ (b) $R = 200$ (c) $R = 300$.

servers in S_e), and (2) Throughput utilization percentage: (total consumed I/O rates in Mb/s) divided by (1260$*$ number of used servers in S_g + 700$*$ number of used servers in S_e). 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 ($\leq$ 3 sec.) is significantly less than INLP ($\geq$ 1 hour) and it can obtain a close-to-optimal performance. The running time comparison between INLP and TSDA (under the maximum time limit) is depicted in Fig. 9.

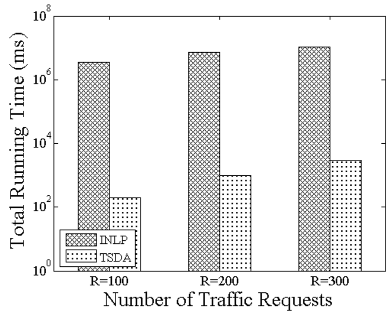


FIGURE 9. Running time.

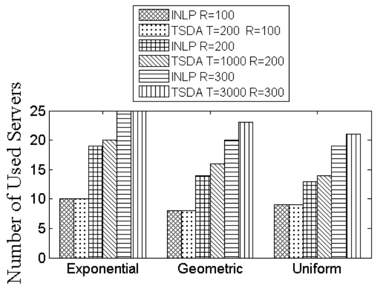


FIGURE 10. Number of used servers for the LSDR problem.

C. SIMULATION RESULTS FOR THE LSDR PROBLEM

Fig. 10 illustrates the number of used servers of these two algorithms for the LSDR problem. Since 7 servers are assumed to be in use and their available storage and workload are reduced, the number of used servers in Fig. 10 is larger than Fig. 6 for the same set of data requests. We see that the exact INLP has equal or less server usage than TSDA for different R. This also verifies its correctness. The TSDA achieves the same performance with the INLP when $R = 100$ and has a slightly higher server consumption than the INLP when $R = 200$ and 300, but its running

TABLE 2. Storage utilization percentage of two algorithms for the LSDR problem.

	INLP Exp.	TSDA Exp.	INLP Geo.	TSDA Geo.	INLP Uni.	TSDA Uni.
$R = 100$	73%	73%	70%	70%	74%	74%
$R = 200$	74.7%	72.8%	68%	60%	74%	70%
$R = 300$	60%	57%	61%	56%	64.7%	59.6%

TABLE 3. Throughput utilization percentage of two algorithms for the LSDR problem.

	INLP Exp.	TSDA Exp.	INLP Geo.	TSDA Geo.	INLP Uni.	TSDA Uni.
$R = 100$	47%	47%	53%	53%	44.9%	44.9%
$R = 200$	64.6%	62.4%	57.7%	51%	51.2%	47.9%
$R = 300$	58%	55%	61.8%	54%	51%	48.5%

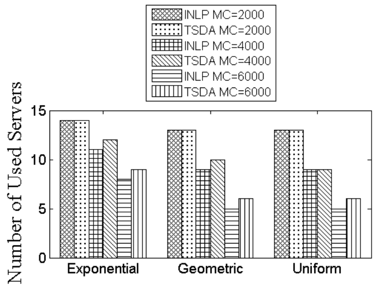


FIGURE 11. Number of used servers for the LSWC problem.

TABLE 4. Storage utilization percentage of two algorithms for the LSWC problem.

	INLP Exp.	TSDA Exp.	INLP Geo.	TSDA Geo.	INLP Uni.	TSDA Uni.
$MC = 2000$	38.7%	38.7%	43%	43%	52%	52%
$MC = 4000$	50%	45.8%	57%	52%	64.5%	64.5%
$MC = 6000$	60%	54%	87%	73%	77.52%	68%

TABLE 5. Throughput utilization percentage of two algorithms for the LSWC problem.

	INLP Exp.	TSDA Exp.	INLP Geo.	TSDA Geo.	INLP Uni.	TSDA Uni.
$MC = 2000$	38.3%	38.3%	31%	31%	25%	25%
$MC = 4000$	39.3%	36%	41%	38%	28.4%	28.4%
$MC = 6000$	54.5%	50%	72%	63%	48.8%	41.2%

time ($\leq$ 3 sec.) is much less than the INLP ($\geq$ 1 hour) as Fig. 9 shows. Tables 2 and 3 show the storage and throughput utilization percentage. When $R = 200$ and 300, the INLP has a higher utilization in terms of both storage and throughput than TSDA, since it consumes a smaller number of servers than TSDA. When $R = 100$, both algorithms have the same storage and throughput utilization.

D. SIMULATION RESULTS FOR THE LSWC PROBLEM

In terms of the LSWC problem, the total data moving cost limit MC is set to 2000, 4000 and 6000, respectively. We assume that 15 servers are in use to accommodate requests, and the number of traffic requests accommodated by each server varies in [3, 10]. Fig. 11 presents the number of used servers after workload consolidation. As expected, the INLP can always save (switch off) more servers after data moving than the heuristic. We also see that with MC increasing, more servers can be emptied (and switched off) by workload consolidation for both INLP and TSDA. Finally, Tables 4 and 5 show the storage and throughput utilization percentage for the INLP and TSDA.

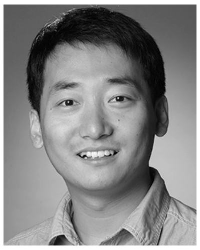
VII. CONCLUSION

In this paper, we have studied the Latency-Sensitive Data Allocation (LSDA) problem, the Latency-Sensitive Data Reallocation (LSDR) problem and the Latency-Sensitive Workload Consolidation (LSWC) problem. Under the assumption that the data access time follows a given distribution and its CDF is known, we have proved these 3 problems are all NP-hard. Subsequently, we propose an exact Integer Nonlinear Program (INLP) and a Tabu Search-based heuristic to solve them. 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.

REFERENCES

[1] *Netflix Finishes Its Massive Migration to the Amazon Cloud*. Accessed: Nov. 23, 2018. [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*. Accessed: Nov. 23, 2018. [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 Trans. Netw.*, vol. 22, no. 6, pp. 2012–2025, Dec. 2014.
[4] S. Garfinkel, "An evaluation of Amazon's grid computing services: EC2, S3, and SQS," Harvard Computer Science Group, Harvard Univ., Cambridge, MA, USA, Tech. Rep. TR-08-07, 2007.
[5] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[6] D.-J. Deng, S.-Y. Lien, C.-C. Lin, S.-C. Hung, and W.-B. Chen, "Latency control in software-defined mobile-edge vehicular networking," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 87–93, Aug. 2017.
[7] *Amazon Found Every 100ms of Latency Cost Them 1% in Sales*. Accessed: Nov. 23, 2018. [Online]. Available: http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/
[8] M. Shen, K. Xu, F. Li, K. Yang, L. Zhu, and L. Guan, "Elastic and efficient virtual network provisioning for cloud-based multi-tier applications," in *Proc. 44th IEEE Int. Conf. Parallel Process.*, Sep. 2015, pp. 929–938.
[9] Z. Wu, C. Yu, and H. V. Madhyastha, "CosTLO: Cost-effective redundancy for lower latency variance on cloud storage services," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2015, pp. 543–557.
[10] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 715–722, Feb. 2016.
[11] *Google Storage: On-demand I/O*. Accessed: Nov. 23, 2018. [Online]. Available: https://cloud.google.com/storage/docs/on-demand-io/
[12] G.-W. You, S.-W. Hwang, and N. Jain, "Ursa: Scalable load and power management in cloud storage systems," *ACM Trans. Storage*, vol. 9, no. 1, pp. 1–29, 2013.
[13] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proc. Conf. Center Adv. Stud. Collaborative Res.* Armonk, NY, USA: IBM, 2009, pp. 101–111.

[14] G. Liu, H. Shen, and H. Wang, "Deadline guaranteed service for multi-tenant cloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2851–2865, Oct. 2016.
[15] S. Yang, P. Wieder, M. Aziz, R. Yahyapour, and X. Fu, "Latency-sensitive data allocation for cloud storage," in *Proc. 15th IFIP/IEEE Symp. Integr. Netw. Service Manage.*, May 2017, pp. 1–9.
[16] P.-J. Maenhaut, H. Moens, B. Volckaert, V. Ongenae, and F. De Turck, "A dynamic tenant-defined storage system for efficient resource management in cloud applications," *J. Netw. Comput. Appl.*, vol. 93, pp. 182–196, Sep. 2017.
[17] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "IoT-based big data storage systems in cloud computing: Perspectives and challenges," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 75–87, Jan. 2017.
[18] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive transmission optimization in SDN-based industrial Internet of Things with edge computing," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1351–1360, Jun. 2018.
[19] B. A. Milani and N. J. Navimipour, "A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions," *J. Netw. Comput. Appl.*, vol. 64, pp. 229–238, Apr. 2016.
[20] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–51, 2017.
[21] G. Wu *et al.*, "On the performance of cloud storage applications with global measurement," in *Proc. IEEE/ACM 24th Int. Symp. Qual. Service (IWQoS)*, Jun. 2016, pp. 1–10.
[22] J.-W. Lin, C.-H. Chen, and J. M. Chang, "QoS-aware data replication for data-intensive applications in cloud computing systems," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 101–115, Jan. 2013.
[23] 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*. Berlin, Germany: Springer, 2013, pp. 124–133.
[24] A. Kumar, N. C. Narendra, and U. Bellur, "Uploading and replicating Internet of Things (IoT) data on distributed cloud storage," in *Proc. IEEE 9th Int. Conf. Cloud Comput.*, Jun./Jul. 2016, pp. 670–677.
[25] D. Boukhelef, K. Boukhalfa, J. Boukhobza, H. Ouarnoughi, and L. Lemarchand, "COPS: Cost based object placement strategies on hybrid storage system for DBaaS cloud," in *Proc. IEEE/ACM CCGRID*, May 2017, pp. 659–664.
[26] G. Liu, H. Shen, and H. Wang, "An economical and SLO-guaranteed cloud storage service across multiple cloud service providers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2440–2453, Sep. 2017.
[27] R. V. Arumugam, Q. Xu, H. Shi, Q. Cai, and Y. Wen, "Virt cache: Managing virtual disk performance variation in distributed file systems for the cloud," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2014, pp. 210–217.
[28] S. Shekhar, H. Abdel-Aziz, A. Bhattacharjee, A. Gokhale, and X. Koutsoukos, "Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 82–89.
[29] K. Salah, K. Elbadawi, and R. Boutaba, "An analytical model for estimating cloud resources of elastic services," *J. Netw. Syst. Manage.*, vol. 24, no. 2, pp. 285–308, 2016.
[30] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman and Company, 1979.
[31] S. Pacheco-Sanchez, G. Casale, B. Scotney, S. McClean, G. Parr, and S. Dawson, "Markovian workload characterization for QoS prediction in the cloud," in *Proc. IEEE 4th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2011, pp. 147–154.
[32] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. 10th IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2007, pp. 119–128.
[33] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. Hoboken, NJ, USA: Wiley, 2015.
[34] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.
[35] F. Glover, "Tabu search—Part II," *ORSA J. Comput.*, vol. 2, no. 1, pp. 4–32, 1990.
[36] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the Internet with nano data centers," in *Proc. 5th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, 2009, pp. 37–48.
[37] 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 *Proc. 7th IEEE Int. Conf. Cloud Comput. (CLOUD)*, Jun./Jul. 2014, pp. 826–832.



SONG YANG received the Ph.D. degree from the Delft University of Technology, The Netherlands, in 2015. From 2015 to 2017, he was a Post-Doctoral Researcher for the EU FP7 Marie Curie Actions CleanSky Project at Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Göttingen, Germany. He is currently an Associate Professor at the School of Computer Science, Beijing Institute of Technology, China. His research interests focus on data communication networks, cloud/edge computing, and network function virtualization.

PHILIPP WIEDER received the Ph.D. degree from TU Dortmund, Germany. He is the Deputy Leader of Datacenter of the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, University of Göttingen, Germany. He is active in the research areas on clouds, grid, and service-oriented infrastructures for several years. His research interest lies in distributed system, service level agreements, and resource scheduling. He has been actively involved in the FP7 IP PaaSage, SLA@SOI, and SLA4D-Grid projects.

MUZZAMIL AZIZ received the Ph.D. degree in computer sciences from RWTH University in 2017. He is a Scientific Researcher at the eScience Group, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen. His areas of expertise are software-defined networking, cloud computing, and distributed systems. He is currently involved in EU funded as well as some in-house projects in relation with the design of data centers services for big data networks. Earlier in his career, he worked in various R&D projects for telecommunication industry associated to services and solutions for next generation networks.

RAMIN YAHYAPOUR received the Ph.D. degree in electrical engineering. He is a Full Professor at the Georg-August University of Göttingen. He is the Managing Director of the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, a joint compute and IT competence center of the university and the Max Planck Society. His research interest lies in the area of efficient resource allocation in its application to service-oriented infrastructures, clouds, and data management. He is especially interested in data and computing services for eScience. He gives lectures on parallel processing systems, service computing, distributed systems, cloud computing, and grid technologies. He was and is active in several national and international research projects. He serves regularly as a reviewer for funding agencies and consultant for IT organizations. He is an organizer and a program committee member of conferences and workshops as well as a reviewer for journals.

XIAOMING FU (SM'09) received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2000. He was then a Research Staff at the Technical University Berlin until joining the University of Göttingen, Germany, in 2002, where he has been a Professor in computer science and heading the Computer Networks Group since 2007. He has spent research visits at the universities of Cambridge, Uppsala, UPMC, Columbia, UCLA, Tsinghua, Nanjing, Fudan, and PolyU of Hong Kong. His research interests include network architectures, protocols, and applications. He is an IEEE Communications Society Distinguished Lecturer. He is currently an Editorial Board Member of the *IEEE Communications Magazine*, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and *Computer Communications* (Elsevier), and has served on the organization or program committees of leading conferences, such as INFOCOM, ICNP, ICDCS, MOBICOM, MOBIHOC, CoNEXT, ICN, and COSN.

XU CHEN received the Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012. He was a Post-Doctoral Research Associate at Arizona State University, Tempe, AZ, USA from 2012 to 2014, and a Humboldt Scholar Fellow at the Institute of Computer Science, University of Göttingen, Germany, from 2014 to 2016. He is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the Vice Director of the National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He was a recipient of the Prestigious Humboldt Research Fellowship awarded by the Alexander von Humboldt Foundation of Germany, the 2014 Hong Kong Young Scientist Runner-Up Award, the 2016 Thousand Talents Plan Award for Young Professionals of China, the 2017 IEEE Communication Society Asia-Pacific Outstanding Young Researcher Award, the 2017 IEEE ComSoc Young Professional Best Paper Award, the Honorable Mention Award of 2010 IEEE International Conference on Intelligence and Security Informatics, the Best Paper Runner-Up Award of 2014 IEEE International Conference on Computer Communications, and the Best Paper Award of 2017 IEEE International Conference on Communications. He is currently an Associate Editor of the IEEE INTERNET OF THINGS and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Series on Network Softwarization and Enablers.