Journal of Network and Computer Applications 75 (2016) 223–235

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

To move or not to move: Cost optimization in a dual cloud-based storage architecture

Yaser Mansouri *, Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 26 April 2016
Received in revised form
21 July 2016
Accepted 30 August 2016
Available online 2 September 2016

Keywords:
Cloud storage
Cost optimization
Data storage management
Dual cloud-based storage architecture

ABSTRACT

IT enterprises have recently witnessed a dramatic increase in data volume and faced with challenges of storing and retrieving their data. Thanks to the fact that cloud infrastructures offer storage and network resources in several geographically dispersed data centers (DCs), data can be stored and shared in scalable and highly available manner with little or no capital investment. Due to diversity of pricing options and variety of storage and network resources offered by cloud providers, enterprises encounter nontrivial choice of what combination of storage options should be used in order to minimize the monetary cost of managing data in large volumes. To minimize the cost of data storage management in the cloud, we propose two data object placement algorithms, one *optimal* and another *near optimal*, that minimize residential (i.e., storage, data access operations), delay, and potential migration costs in a dual cloud-based storage architecture (i.e., the combination of a temporal and a backup DC). We evaluate our algorithms using real-world traces from Twitter. Results confirm the importance and effectiveness of the proposed algorithms and highlight the benefits of leveraging pricing differences and data migration across cloud storage providers (CSPs).

1. Introduction

Data volume is one of the important characteristics of cloud-based application (e.g., Online Social Network) and has been changed from TB to PB with an inevitable move to ZB in current IT enterprises. From statistical perspective, 8×10^5 PB of data were stored in the world by the year of 2000 and it is expected that this number will increase to 35 ZB by 2020 (Yu et al., 2015). Storing and retrieving such data volume demand a highly available, scalable, and cost-efficient infrastructure.

Thanks to the cloud infrastructures, management of such large volume data has been simplified and the need for capital investment has been removed from IT companies. However, this creates a major concern for these companies regarding the cost of data management in the cloud. The cost of data storage management (simply, cost of data management) is a vital factor from companies' perspective since it is the essential driver behind the migration to the cloud. Thus, companies are in favor of optimizing data management cost in the cloud deployments. In order to optimize data management cost, choosing a suitable storage option across CSPs in the right time becomes a nontrivial task. This happens due to the two following reasons.

First, there is an array of pricing options for the variety of storage and network services across CSPs (e.g., Amazon, Google, and Microsoft Azure). CSPs currently offer at least two classes of storage service: Standard Storage (SS) and Reduced Redundancy Storage (RRS). RRS enables users to reduce cost at the expenses of lower levels of redundancy (i.e., less reliability and availability) as compared to SS. These services provide users with API to *Get* (read) data from storage and to *Put* (write) data into it. In mid-2015, Amazon and Google respectively introduced Infrequent Access Storage (IAS) and Nearline storage services that aims at hosting objects with infrequent *Gets*/*Puts*. Both services charge lower storage cost in comparison to their corresponding RRS but higher cost for *Gets* and *Puts*.

Furthermore, CSPs charge users with different out-network costs to read data from a DC to the Internet (typically in-network data transfer is free). They also offer discounts for data transfer between their DCs. For example, Amazon reduces out-network cost when data is transferred across its DCs in different regions and Google offers free of charge data exchange between its DCs in the same region. Thus, taking the advantage of diversification in price of storage and network (as well as service type) plays an important factor in *residential cost* (i.e., Storage and data access operations) as a major part of the data management cost. Note that data access operations are *Get* and *Put* in this paper.

Second, there is time-varying workload on the object stored in

* Corresponding author.
E-mail addresses: yase@student.unimelb.edu.au (Y. Mansouri), rbuyya@unimelb.edu.au (R. Buyya).

http://dx.doi.org/10.1016/j.jnca.2016.08.029
1084-8045/

the cloud. Presume that an object is a tweet/photo and it is posted on the user's feed (e.g., timeline in Facebook) by herself or her friends. *Get*s and *Put*s are usually high in the early lifetime of the object and we say that such object is in *hot-spot* status. As time passes, *Get*s and *Put*s are reduced and we refer that the object is in *cold-spot* status. Thus, it is cost-efficient to store the object in a DC with lower out-network cost (referred as a *temporal* DC) in its early lifetime, and then migrate it to the DC with lower cost in storage (referred as a *backup* DC). If the object migration happens between a temporal DC and a backup DC, the user incurs *migration cost*. This cost is another part of the data management cost, which is affected by the number of *Get*s, *Put*s, and the object size. It is important to note that the migration cost might be zero in some cases: (i) if both DCs belong to the same provider and are in the same region, then transferring objects between DCs is free, as in Google provider, and (ii) if *temporal* and *backup* DCs are the same and the object is just moved from a storage class to another (i.e., from SS to IAS) within the same DC.

Besides discussed costs, *latency* for reading from (writing into) the data store is also a vital performance criterion from the user's perspective. The latency is defined as the elapsed time between issuing a Get/Put and receiving the required object. To respect this criterion, we convert latency into monetary cost, as a *latency cost*, and integrate it in our cost model.

In summary, by wisely taking into account the discussed differences in prices across CSPs and time-varying workload, we can reduce the data management cost (i.e., residential, latency, and migration costs) as one of the main user's concerns with regard to the cloud deployment. To address this issue, we make the following *contributions*:

- We propose an optimal algorithm that optimizes data management cost in the dual cloud-based architecture when the workload in terms of *Get*s and *Put*s on the objects is known.
- We also propose a near-optimal algorithm that achieves competitive cost as compared to that obtained by the optimal algorithm in the absence of future workload knowledge.
- We demonstrate the effectiveness of the proposed algorithms by using the real-world traces from Twitter in a simulation.

The reminder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents system and cost model. In Section 4, we describe our object placement algorithms to save cost. Section 5 presents our simulation experiments and evaluation of the proposed algorithms. Finally, in Section 6, we conclude this paper with future work issue related to cost optimization of data management across Geo-replicated cloud-based data stores.

2. Related work

We compare our work in this paper with state-of-the-art works in five categories: benefits of cloud deployments, Geo-distributed cloud storage services, cloud-based Content Delivery Network (CDN), hierarchical storage management, and computing resource allocation.

Benefits of cloud deployments: Some recent studies investigate when to use cloud-based services, in particular focusing on how and when to migrate applications from a private cloud to a public one (Hajjat et al., 2010; Tak et al., 2011; Wood et al., 2010). While they considered issues such as monetary cost and latency in wide-area network, none of them leverage pricing differences across CSPs to optimize storage cost.

Geo-distributed cloud storage services: Several previously proposed systems such as RACS (Abu-Libdeh et al., 2010), DEPSKY (Bessani et al., 2011) and SafeStore (Kotla et al., 2007) are deployed across Geo-distributed DCs to mitigate vendor lock-in and enhance availability, durability, and performance. SPANStore (Wu et al., 2013) leveraged pricing differences across Geo-distributed DCs to minimize the cost. Qiu et al. (2015) proposed a dynamic control algorithm to minimize the operational cost across hybrid cloud providers, subject to meeting the response time constraints. Cosplay (Jiao et al., 2014) optimized the monetary cost for an online social network over Geo-DCs while ensuring QoS and the availability of data. This objective is achieved through changing the rule of replicas (i.e., master and slave replica) across DCs that belong to a cloud provider. These systems exploit neither the object migration across DCs nor the object movement between storage classes within a DC in order to minimize cost.

Cloud-based CDN: With the advent of cloud-based storage services, some literature has been devoted to utilize cloud storage in a CDN in order to improve performance and reduce monetary cost. Broberg et al. (2009) proposed MetaCDN that exploits cloud storage to enhance throughput and response time, while ignoring cost optimization. Chen et al. (2012) investigated the problem of placing replicas and distributing requests (issued by users) in order to optimize cost, while meeting QoS requirements in a CDN by utilizing cloud storage. Papagianni et al. (2013) go one step further by optimizing replica placement problem and requests redirection, while satisfying QoS for users and considering capacity constraints on disks and network. Salahuddin et al. (2015) proposed another model that minimizes monetary cost and QoS violation subject to guaranteeing SLA in a cloud-based CDN. In contrast to these works which have proposed greedy algorithms for read-only workload, our work exploits the pricing differences among CSPs for time varying read and write workload in the dual cloud-based storage architecture.

Hierarchical storage management: An architecture mimics a hierarchical storage management (HSM) when data automatically moves between low- and high-cost storage media (Wikipedia, 2013). Puttaswamy et al. (2012) deployed a generalized form of HSM to reduce the cost of operating a file system in a single cloud storage. Unlike our work, they neither require to consider migration cost nor need to deal with latency across DCs. Our approach is different from their proposed solution as we consider the object migration cost between DCs.

Computing resource allocation: The efficient utilization of computing resources is a complex and challenging issue, where *tasks scheduling* is one of them. Fang et al. (2010) proposed a two level task-scheduling mechanism based on load balancing of resources. This mechanism cannot only guarantee the satisfaction of users but also improve the utilization of resources. Van Den Bossche et al. (2013) studied an algorithm to schedule deadline constrained tasks in the hybrid cloud with the aim of minimizing cost while maintaining QoS. Lin et al. (2014) designed a task-scheduling algorithm that considers bandwidth requirements, in addition to CPU and memory, to achieve better performance as compared to bandwidth-only and computation-only algorithms. Rodriguez and Buyya (2015) and Liu et al. (2016) studied task-scheduling algorithms for scientific workload to minimize the cost of used resources whereas guaranteeing the user-defined deadline constraint or reducing execution time of tasks.

Load-balancing is another challenging issue and helps in reducing response time and resources consumption, maximizing scalability, avoiding services bottleneck, etc. One technique to accomplish load balancing is VM migration, which in turn, reduces the energy consumption and environmental impacts (Beloglazov et al., 2012). In the same line, several studies utilize spatial load balancing (Rao et al., 2010; Liu et al., 2011; Gao et al., 2012), temporal load-balancing (Liu et al., 2012; Luo et al., 2014), or spatio-temporal load balancing exploiting both geographical and temporal variation of electricity price (Luo et al., 2015) to achieve

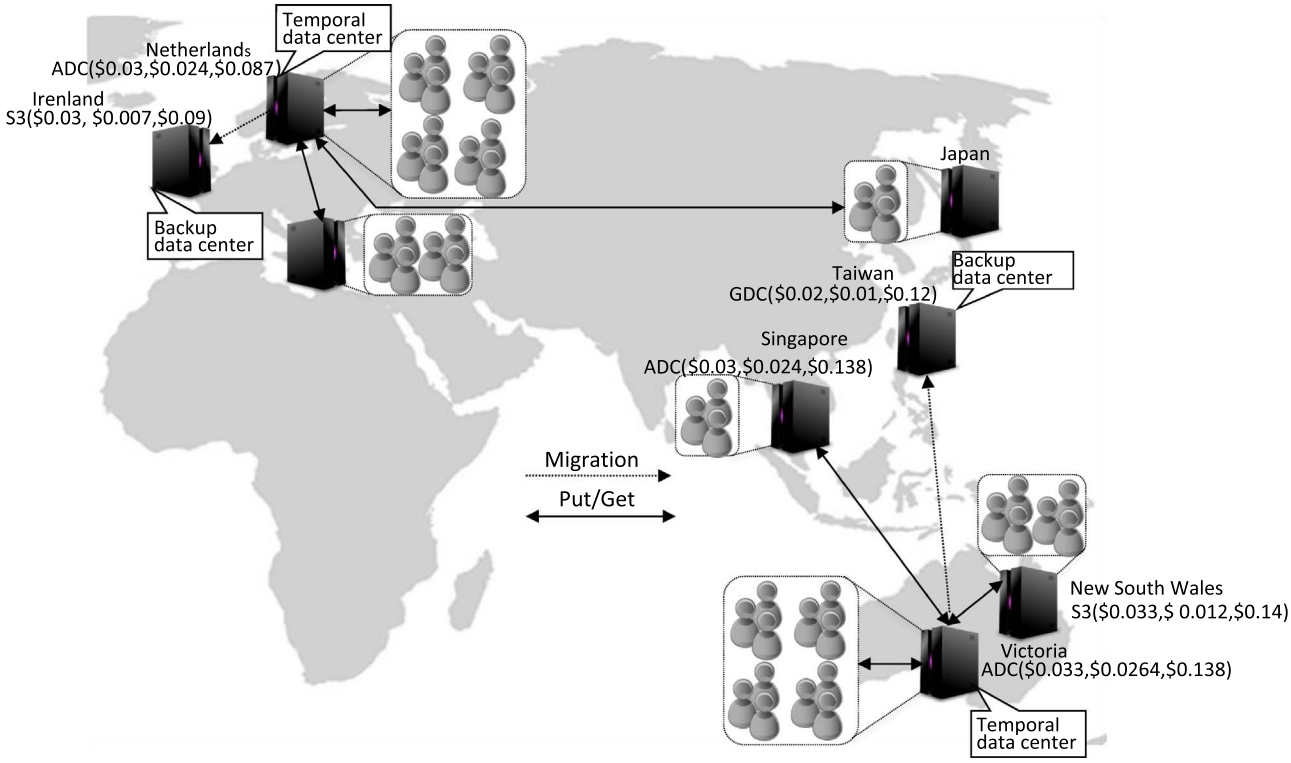

Fig. 1. A scenario of the dual cloud-based storage architecture in the European and Asia-Pacific regions. Parenthesis close to each DC's name shows the storage price (per GB per month) for standard storage, backup storage, and network price (per GB), respectively.

more reduction in energy consumption. However, compared to these studies, our work focuses on storage services to achieve lower monetary cost.

3. System and cost model

We first describe the dual cloud-based architecture, which can lead to reduced monetary cost for applications. Then, we discuss the cost model and the objective function that should be minimized considering the objective and specifications of the architecture.

3.1. System model

In our system model, an object is a tweet or photo posted by users on their feed. As stated earlier, the object is stored in the temporal (resp. backup) DC during its hot-spot (resp. cold-spot) status to benefit from lower network (resp. storage) cost. A transition between hot- and cold-spot probably leads to the object migration. Our architecture uses *stop and copy* migration technique (Elmore et al., 2011) in which *Get*s and *Put*s are respectively served by the temporal and backup DCs during the object migration. Fig. 1 illustrates the scenario of the architecture in two regions. In the architecture, each user is assigned to the closest DC among the DCs which are Geo-graphically located across the world. This DC is referred as the home DC. The determined home DC for each user is then paired with the DC that is selected by the object owner or application provider.[1] The paired DCs are considered as the temporal and backup DCs. As an example, in the Asia-Pacific, a user stores the object in the Victoria Azure DC (Victoria ADC) with low network cost, while their friends in Singapore and New South Wales access the object by issuing their read/write requests from their home DCs. As time passes, the object is migrated to the Taiwan Google DC (Taiwan GDC), which has lower storage cost (i.e., Nearline) in comparison to S3 and ADC in the Asia-Pacific region.

3.2. Cost model

The system model is represented as a set of independent DCs D where each DC d is associated with the following cost elements to manage data:

1. *Storage cost*: $S(d)$ denotes the storage cost per unit size per time.
2. *Network cost*: $O(d)$ is the cost per byte of out-bound bandwidth transferred from DC d (in-bound bandwidth is typically free).
3. *Transaction cost*: t_g and t_p define transaction cost for a *Get* and *Put* request respectively.

Assume that the application hosts a set of objects in time slot $t \in [1...T]$. Each object is associated with $v(t)$, $r(t)$, and $w(t)$ denoting respectively size in byte, the number of read and write operations for the object in time slot t. Also suppose l denotes the latency between the DC that issues a *Put*/*Get* for the object and the DC that hosts the required object. $x_d(t)$ represents whether the object exists in DC d in time slot t ($x_d(t) = 1$) or not ($x_d(t) = 0$).

Residential cost: The residential cost of the object in time slot t is as follows. (i) The storage cost of the object is equal to its size multiplied by the storage price ($S(d)v(t)$). (ii) The read cost of the object is the cost of all Gets ($r(t)t_g(d)$) and the communication cost ($r(t)v(t)O(d)$). (iii) The write cost of the object is the cost of all *Put*s for updating of the object ($w(t)t_p(d)$). Thus, the residential cost C_R is defined as:

$$C_R(x_d, t) = \sum_{x_d} x_d(t)[v(t)(S(d) + r(t)O(d)) + r(t)t_g(d) + w(t)t_p(d)]. \quad (1)$$

[1] We pair each home DC with 21 DCs in the experiment to determine which combination of temporal and backup DCs is cost-efficient.

Delay cost: Time is cost and user-perceived latency for reading and writing the object is a vital criterion. For example, Amazon reported that every 100 ms of latency reduces 1% of its sales (Latency-it costs you). To capture this cost, the incurred latency for *Gets* and *Puts* is considered as a monetary cost (Zhang et al., 2013). We consider "Get/Put delay" as the time taken from when a user issues a Get/Put from the DC d' to when he/she gets a response from the DC d that hosts the object. In fact, the read and write requests are issued from the application hosted by the closest DC to the user. The delay cost of read and write requests, C_D, can be formally defined as:

$$C_D(x_d, t) = \sum_{x_d(t)} x_d[l(d', d)v(t)(r(t) + w(t))l_w], \quad \forall\, d, d' \in D. \tag{2}$$

In Eq. (2), $l(d', d)$ denotes the latency between DC d' that issues requests and DC d that hosts the object. $l(d', d)$ in our formulation and evaluation is based on the round trip times between d and d'. This is reasonable because for the application, the size of objects is typically small (e.g., tweets, photos, small text file), and thus data transitions are dominated by the propagation delays, not by the bandwidth between the two DCs. For applications with large objects, the measured $l(d', d)$ values capture the impact of bandwidth and data size as well. In the above equation, l_w denotes the *latency cost weight*, which converts latency into a monetary cost. It is set by the application based on the importance degree of the latency from the user's perspective. The more importance the latency, the higher l_w will be.

We use *stop and copy* migration technique in our model for two reasons. First, the system performance is not significantly affected by this technique because (i) the transfer time of a bucket (which is at most 50 MB in size (Corbett et al., 2013)) between *temporal* and *backup* DCs is about few seconds, and (ii) a significantly lower number of *Puts* and *Gets* must be served after the transition to the cold-spot status. Second, this technique imposes a lower cost as compared to the log-based technique (Tran et al., 2011). The object migration cost is the cost of retrieving the object from the source DC ($v(t)O(d_s)$) and writing it to the destination DC ($t_p(d_d)$). Thus the migration Cost $C_M(t-1, t)$ is defined as:

$$C_M(t-1, t) = v(t)O(d_s) + t_p(d_d), \tag{3}$$

where d_s and d_d are the source and destination DCs respectively.

Migration cost: As time passes, the number of *Puts* and *Gets* decreases, and it is cost-effective to migrate the object from a *temporal* DC to a *backup* DC with the lower cost in storage.

Cost optimization problem: Considering the aforementioned cost model, we define the objective as to determinate the object placement (i.e., x_d) in each time slot t so that the overall cost (i.e., the sum of residential, delay, and potential migration costs) of the object during $[1\ldots T]$ is minimized. Thus, the objective function can be defined as:

$$\sum_{t=1}^{T} \sum_{x_d(t)} C_R(x_d(t), t) + C_D(x_d(t), t) + C_M(t-1, t), \quad x_d(t) \in \{0, 1\}. \tag{4}$$

4. Data management cost optimization

To solve the aforementioned cost optimization problem, we first propose a dynamic algorithm to minimize the overall cost while the future workload is assumed to be known a priori. Then, we present a heuristic algorithm to achieve competitive cost as compared to the cost of dynamic algorithm for unknown objects workload.

4.1. Optimal object placement (OOP) algorithm

Let $P(d, t)$ be the minimum cost of the object in DC d in time slot t. In order to compute P, we drive a general recursive equation for P as Eq. (5), where $C(d, t)$ is the summation of the residential, delay, and potential migration costs of the object in time slot t (Eqs. (1)–(3)).

We first enumerate all possible DCs that could store the object in time slot t and then calculate the residential and delay costs. Second, we consider all possible placements of the object in time slot $t-1$ and calculate the migration cost from each DC in time slot $t-1$ to the current DC d in time slot t.

If we store the object at DC d in time slot t, then the overall cost (i.e., $P(d, t)$) is the minimum of the overall cost in time slot $t-1$ (i.e., $P(d, t-1)$) plus the residential, delay, and potential migration costs of the object in time slot t (i.e., $C(d, t)$). This recursive equation is terminated when t is zero and its corresponding $P(d, t)$ value is zero. Therefore, we define the recursive equation for the OOP algorithm as:

$$P(d, t) = \begin{cases} \min\limits_{d}[P(d, t-1) + C(d, t)], & t > 0 \\ 0 & t = 0. \end{cases} \tag{5}$$

Once $P(d, t)$ is calculated for all DCs d during $t \in [1\ldots T]$, we calculate $\min_d P(d, T)$ as the minimum cost of the object. It is easy to find the optimal location of the object in time slot t (i.e., $x_d^*(t)$) by backtracking from the minimum cost in time slot T. In each time slot t, if the cost value leads to minimum cost value in time slot $t+1$, then the value of $x_d(t)$ is 1; otherwise is 0. Continuing on the backtrack step from T to 1, we find the value of $x_d^*(t)$ for all $t \in [1\ldots T]$.

The pseudo code in Algorithm 1 shows the discussed OOP. Note that since the value of $x_d^*(t)$ is 1 only for one DC in each time slot t, we can safely initialize $x_d(t)$ with 0 for all DCs d in all time slots t.

Algorithm 1: Optimal Object Placement (OOP) algorithm.

Input: DCs and objects specifications
Output: $x_d^*(t)$ and the optimized overall cost during $t \in [1\ldots T]$
1 Initialize: $\forall d \in DC,\ P(d, 0) \leftarrow 0 \wedge \forall\, t \in [1\ldots T],\ x_d(t) \leftarrow 0$
2 for $t \leftarrow 1$ **to** T **do**
3 /*DCs(i. e., d)are a temporal DC and a backup DC. */
4 forall $x_d(t)$ do
5 forall $x_d(t-1)$ do
6 Calculate $P(d, t)$ based on Equation (5).
7 **end**
8 **end**
9 end
10 Select $\min_d P(d, T)$ as the optimized overall cost (i.e., Eq.(4)).
11 Take the optimized overall cost, i.e., $\min_d P(d, T)$, in time T and set its corresponding $x_d(T)$ to 1 (i.e., $x_d^*(T)$). Then, the value of $x_d(T-1)$ is set to 1 if its corresponding cost value in time slot $T-1$ leads to the optimized overall cost in time slot T. In the same way, find the value of all $x_d(t)s$ from $T-2$ to 1.
12 All $x_d(t)$s with value of 1 are $x_d^*(t)$ s.
13 Return $x_d^*(t)$ and the optimized overall cost.

The time complexity of the algorithm is dominated by three nested "for" loops in which the recursive function (Eq. (5)) is calculated. For each paired DCs (lines 5–8), we compute the value of $P(d, t)$ for each time slot $t \in [1\ldots T]$. Thus, this calculation takes $O(2T)$ since we have two DCs in the dual cloud-based storage architecture. To find the location of the object in each time slot, we

need to backtrack the obtained results (line 12), which takes $O(T)$. Since this process (lines 2–13) is repeated for all pairs of DCs (i.e., $\binom{n}{2}$, not shown in the algorithm), the total time complexity of the algorithm yields $O(n^2T)$, where n is the number of DCs.

4.2. Near-Optimal Object Placement (NOOP) algorithm

We propose a novel heuristic algorithm that finds a competitive solution for the cost optimization problem, as compared to that of OOP. Intuitively, if the object migrations do not happen at the right time, then besides migrations cost, the residential and delay costs increase as compared to these costs in OOP. We refer to the difference between these costs in OOP and NOOP as the overhead cost. This overhead cost should be minimized by providing a strategy that leads to the object migration in time(s) t_m so that it should be near to the optimal migration time(s) obtained by OOP. To achieve this aim and minimize the overhead cost, we make a trade-off between the migration cost and the summation of residential and delay costs (for summary, denoted by C_{RD}) in the absence of the future workload knowledge. The idea behind this trade-off is that the object is migrated to a new DC when (i) the object migration leads to save monetary cost at the new DC, and (ii) the sum of the lost cost savings from the last migration time up to the current time slot gets more than or equal to the migration cost of the object between DCs. This strategy avoids migrating the object too early or too late. The "trick" is to move the object "lazily", i.e., when sum of overall cost savings that could be done by any earlier migrations from the last migration time t_m is as large as the cost of migration in the current time slot.

Now we formally define the discussed trade-off. Let $C_M(t_{m-1}, t_m)$ be the migration cost between two consecutive migration times, where t_m is the last time the object is migrated. For each time slot $v \in [t_m, t)$, we calculate the summation of the residential and delay costs of the object (i) in the DC hosting the object in the previous time slot $t-1$, and (ii) in the new DC in the current time slot as if the object is migrated to it. Now, for each time slot v, we calculate the summation of the difference between two C_{RD}s in the two previous cases from time $v = t_m$ to $v = t-1$. This summation is equal to $\sum_{v=t_m}^{t-1}[C_{RD}(x_d(v-1), v) - C_{RD}(x_d(v), v)]$. Note that if the migration of the object happens in time slot $v=t$, we assign $t_{m-1} = t_m$ and $t_m = t$ in Eqs. (6) and (7). Based on the summation of the residential and delay costs (i.e., C_{RD}) and the migration cost (i.e., $C_M(t_{m-1}, t_m)$) in the current time slot, the algorithm decides whether the object should be migrated or not. As discussed before, to avoid the object being migrated too early or too late, the object migration happens only if both the following conditions are satisfied:

First, the object has the potential to be migrated to a new DC if

$$C_M(t_{m-1}, t_m) \le \sum_{v=t_m}^{t-1} [C_{RD}(x_d(v-1), v) - C_{RD}(x_d(v), v)]. \quad (6)$$

Otherwise, the object is kept in the previous DC as determined in time slot $t-1$. This condition prevents the object being migrated too late. Second, to avoid early object migration, we enforce the following condition:

$$C_M(t_{m-1}, t_m) + C_{RD}(x_d(t), t) \le C_{RD}(x_d(t-1), t). \quad (7)$$

This constraint implies that the overall cost of the object, including residential, delay, and migration costs, in the new DC should be less or equal to the summation of residential and delay costs of the object if it stays in the determined DC in time slot $t-1$. Algorithm 2 represents the pseudo code for NOOP.

Algorithm 2: Near-Optimal Object Placement (NOOP) algorithm.

Input: DCs and objects specifications
Output: $\hat{x}_d(t)$ and the overall cost during $t \in [1...T]$ as denoted by C_{ove}.
1 $C_{ove} \leftarrow 0$, $\forall\, t \in [1...T]$, $x_d(t) \leftarrow 0$
2 $C_{ove} \leftarrow$ Select either backup or temporal DC so that cost C_{RD} is minimized in time slot t, and set Corresponding $x_d(t=1)$ to 1.
3 $t_m \leftarrow 1$
4 for $t \leftarrow 2$ **to** T **do**
/*DCs (i. e.,d) are a temporal DC and a backup DC. */
5 forall $x_d(t)$ do
6
7 $C_{RD}(.) \leftarrow$ Determine $x_d(t)$ by minimizing $C_{RD}(x_d, t)$
8 $C_{ove} \leftarrow C_{ove} + C_{RD}(.)$
9 **end**
10 **if** (*Equations* (6), (7) *and* $x_d(t-1)!=x_d(t)$) **then**
11
12 $t_{m-1} \leftarrow t_m$, $t_m \leftarrow t$, $C_M \leftarrow$ calculate $C_M(t_{m-1}, t_m)$
13 $x_d(t) = 1$, $C_{ove} \leftarrow C_{ove} + C_M$
14 **else**
15 $x_d(t) \leftarrow x_d(t-1)$
end
16 end
17 x_ds with value of 1 are $\hat{x}_d(t)$ s.
18 Return $\hat{x}_d(t)$ and C_{ove}.

The time complexity of the algorithm is as follows. We need to calculate the total cost (lines 4–16) for each paired DCs for each time slot $t \in [1...T]$. This calculation takes $O(T)$ (lines 6–15). Since we repeat this computation for all pairs of DCs (i.e., $\binom{n}{2}$, not shown in the algorithm), the total time complexity of the algorithm is $O(n^2T)$, where n is the number of DCs.

5. Performance evaluation

In this section, we first discuss the experimental settings in terms of workload characteristics, DCs specifications, and assignment of users to DCs. Then, we study the performance of the proposed algorithms in terms of cost saving and investigate the effect of the various parameters on the cost saving.

5.1. Experimental settings

We evaluated the performance of algorithms via extensive experiments using a dataset from Twitter (Li et al., 2012). In the dataset, each user has her own profile, tweets, and a user friendship graph over a 5-year period. We focus on tweet objects posted by the users and their friends on the timeline, and obtain the number of tweets (i.e., number of *Puts*) from the dataset. Since the dataset does not contain information of accessing the tweets (i.e., number of *Gets*), we set a Get/Put ratio of 30:1, where the pattern of *Gets* on the tweet follows Longtail distribution (Atikoglu et al., 2012). This pattern mimics the transition status of the object from hot- to cold-spot status. The size of each tweet varies from 1 KB to 100 KB in the dataset.

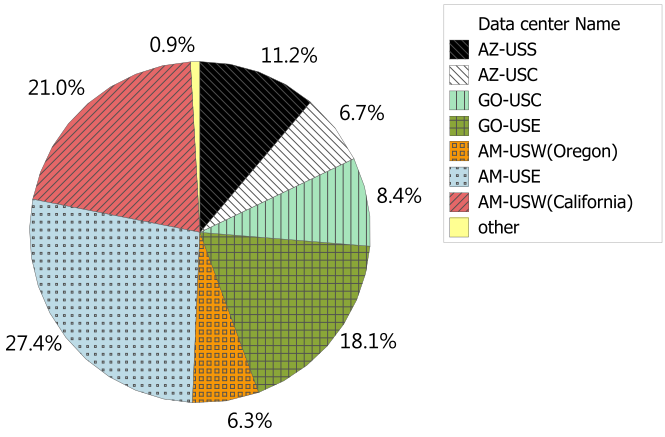

Fig. 2. Allocated users to data centers (%).

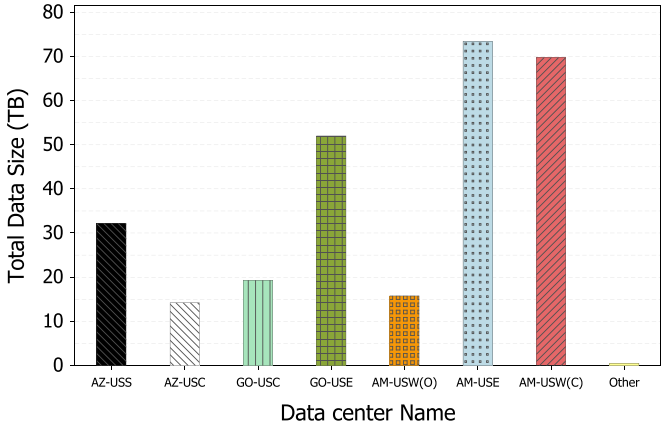

Fig. 3. Total data size in data centers.

We model 22 DCs in CloudSim Toolkit (Calheiros et al., 2011), and among these DCs, 9 are owned by Microsoft Azure, 4 by Google, and 9 by Amazon. Each DC is referred by a name that consists of (i) provider name: Microsoft Azure (AZ), Google (GO) and Amazon (AM); (ii) location: USA (US), Europe (EU), Asia (AS), Australia (AU), Japan (JA), and Brazil (BR); and (iii) the specific part of the location: south (S), north (N), west (W), east (E), and center (C). Since we have two different Amazon DCs in US-West region, i.e., Oregon and California, we add letter "O" for Oregon and "C" for California. For example, based on this naming, the DC with name `AZ-USS` refers to the DC that is part of Microsoft Azure in the USA South. Every DC offers two classes of storage services: (i) SS and (ii) RRS, that is, Locally Redundant Storage (LRS) for Azure, Nearline for Google, and IAS for Amazon. The latter storage class, i.e., RRS, is used for the object when it transits to *cold-spot* status. We set the storage and network prices of each DC as of September 2015.[2]

We measured inter-DCs latency (22*22 pairs) over several hours using instances deployed on all 22 DCs. We run `Ping` operation for this purpose, and used the medium latency values as the input for our experiments. The default value of l (as the latency cost weight) to convert delay to cost is 10. As a result, delay cost constitutes 7–10% of the total cost in the system.

With the help of Google Maps Geocoding API (The google maps geocoding api), we convert users' text locations to geo-coordinates (i.e., latitude and longitude) according to the users' profiles. Then, according to the coordination of users and DCs, we assigned users to the nearest DC based on their locations. In the case of two (or more) DCs with the same distance from the user, one of these DCs is randomly selected as the home DC for the user. One-month (December 2010) of Tweeter dataset with more than 46 K users, posting tweet on their timeline, is utilized for our experiments. As shown in Fig. 2 , around 99.1% of users are assigned to DCs in the USA while the remaining users are designated to DCs in Europe, Asia, Australia and Brazil.[3] This is because most of the users of the dataset come from the USA region. Therefore, we focus on the cost saving for DCs in the USA including: two Azure DCs (`AZ-USS` and `AZ-USC`), two Google DCs (`GO-USC` and `GO-USE`), and three Amazon DCs (`AM-USW(O)`, `AM-USE` and `AM-USW(C)`). The total size of data in each DC, as depicted in Fig. 3 , is dependent on the number of users allocated to each DC and the number of tweets posted by users.

Table 1
Summary of simulation parameters.

Parameters setting	Data size factor	Latency cost weight	Read to write ratio	Access pattern on objects
Default	1	10	30	Longtail
Range	0.2–1	1–30	1 – 30	Normal and Random

Table 2
Evaluation settings for figures and tables.

Figs./table	Data size factor	Latency cost weight	Read to write ratio	Access pattern on objects
4–6	0.2,1	10	30	Longtail
7	0.2–1	10	30	Longtail
8	1	1–30	30	Longtail
9	1	10	1–30	Longtail
Table 3	1	10	30	Normal and Random

5.2. Results

We compare the cost savings gained by the proposed algorithms with the following policy benchmark. It is important to mention that (1) the obtained results are valid for the current prices offered by three well-known cloud providers investigated in this paper, and these prices may change quickly in the current competitive market, and (2) in this work, cloud provider selection is only determined based on the monetary cost, while data placement decision can be made based on other criteria such as availability, durability, scalability, and even the reputation of the cloud provider from application owner's perspective. Thus, with these results, we do not intend to advertise or harm the reputation of an individual cloud provider.

5.2.1. Benchmark policy

In the benchmark policy, we permanently store the user's objects in the home DC (i.e., closest DC), and thus the object is not allowed to be migrated to another DC. This is because that application providers often deploy their data in data centers close to their user base. In all experiments, we normalize the incurred cost of the algorithms to the cost of the benchmark policy by varying the following parameters: home DCs, data size factor, latency cost

[2] https://www.aws.amazon.com/s3/pricing/
https://cloud.google.com/storage/pricing
https://azure.microsoft.com/en-us/pricing/details/storage/
https://azure.microsoft.com/en-us/pricing/details/data-transfers/

[3] We also used the same policy to assign friends of the user to a DC. The user's friends are derived from the friendship graph of dataset.

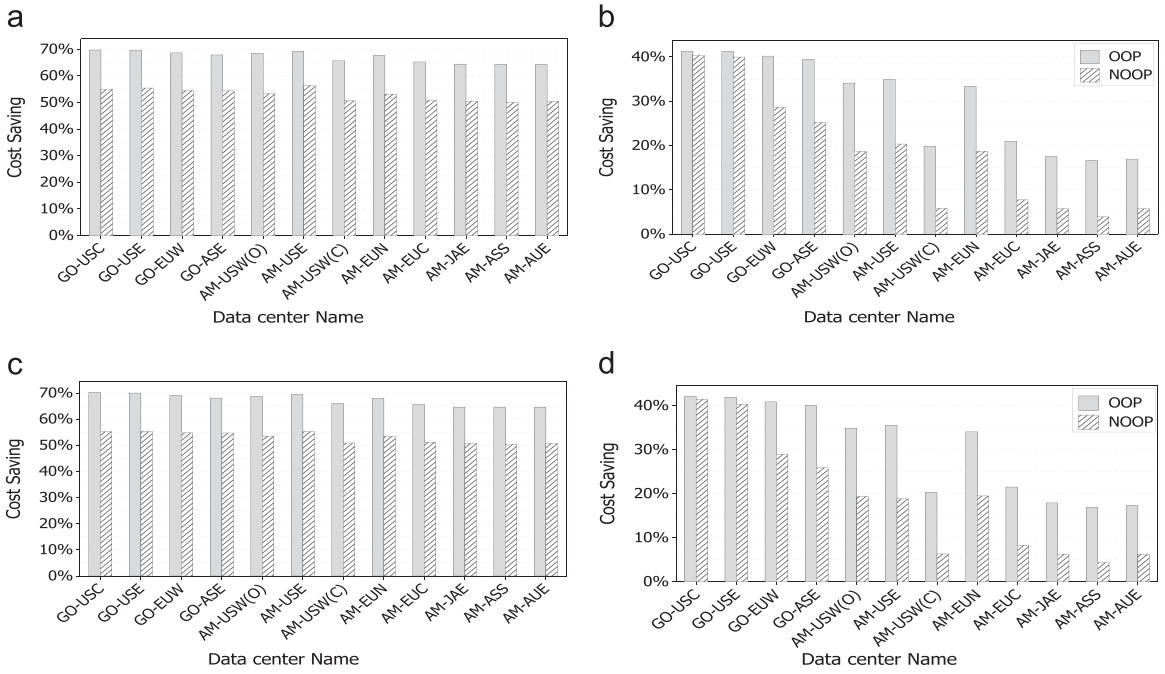

Fig. 4. Cost saving of OOP and NOOP algorithms for two Azure DCs: AZ-USS and AZ-USC as home DCs with data size factor 0.2 and 1. (a) Data center AZ-USS, data size factor=0.2. (b) Data center AZ-USS, data size factor=1. (c) Data center AZ-USC, data size factor=0.2. (d) Data center AZ-USC, data size factor=1.

weight, read to write ratio, and access pattern of read/write on the objects. Each parameter has a default value and a range of values as summarized in Table 1. This range is used for studying the impact of the parameter variations on the cost performance of the proposed algorithms. For clarity, Table 2 summarizes the specific settings in terms of parameters corresponding to each figure. In the following section, we discuss the cost saving of OOP and NOOP.

5.2.2. Cost performance

In this section, we study the cost savings of the proposed algorithms for the most populated DCs (i.e., 6 home DCs) with factor size 0.2 and 1. Note that a DC with "data size factor *x*" means that it only stores *x* percent of the generated total data size, as shown in Fig. 3 for each home DC. A DC stores the total data size when data size factor is 1. For example, based on Fig. 3 , AZ-USS with *data size factor* 0.2 stores 20% of 30 TB. It is important to note that we report result for each pairing between the home DC and each of 21 DCs in the experiments when cost can be saved.

Fig. 4 shows the cost savings of OOP and NOOP for AZ-USS and AZ-USC when each of these home DCs are paired with 21 DCs. As

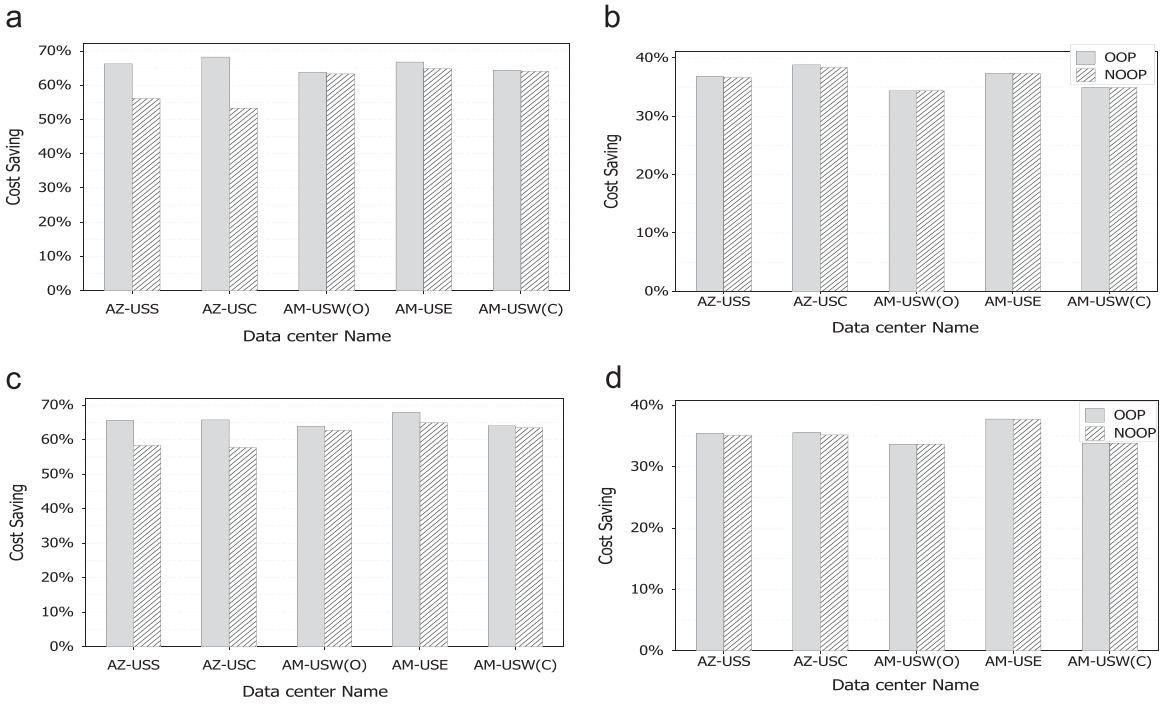

Fig. 5. Cost saving of OOP and NOOP algorithms for two Google DCs: GO-USC and GO-USE as home DCs with data size factor 0.2 and 1. (a) Data center GO-USC, data size factor=0.2. (b) Data center GO-USC, data size factor=1. (c) Data center GO-USE, data size factor=0.2. (d) Data center GO-USE, data size factor=1.

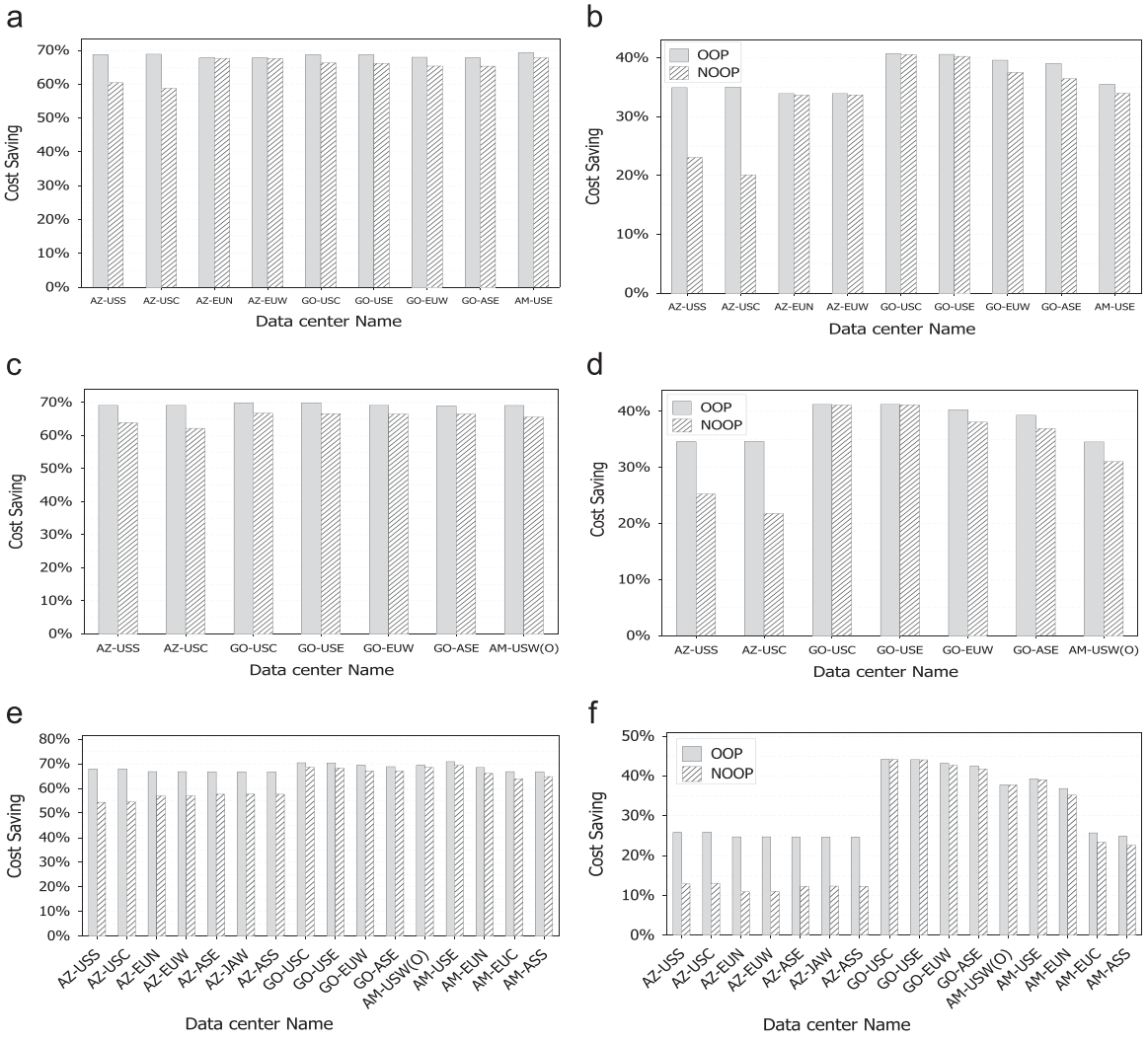

Fig. 6. Cost saving of OOP and NOOP algorithms for three Amazon data centers: AM-USW(O), AM-USE, and AM-USW(C) as home data centers with data size factor 0.2 and 1. (a) Data center AM-USW (O), data size factor=0.2. (b) Data center AM-USW (O), data size factor=1. (c) Data center AM-USE, data size factor=0.2. (d) Data center AM-USW, data size factor=1. (e) Data center AM-USW (C), data size factor=0.2. (f) Data center AM-USW (C), data size factor=1.

expected, the cost cannot be saved when AZ-USS and AZ-USC are paired with Azure DCs, as Azure DCs have more (or the same) cost in the network and storage services than these considered home DCs. In contrast, Google DCs are more suitable to pair with aforementioned home DCs compared to the DCs that belong to Amazon. The reason is that Google DCs have the cheapest cost in the storage service (i.e., Nearline) for hosting the objects in their cold-spot status. For pairing home DCs with Google DCs, OOP can save cost about 40% and 70% respectively when the data size factor is equal to 1 and 0.2. However, NOOP can reduce cost by 40% (resp. 25–28%) if the home DCs are paired with GO-USC and GO-USE (resp. GO-EUW and GO-ASE) when the data size factor is equal to 1. For data size factor=0.2, NOOP cuts the cost by around 55% (see Figs. 4a and 4c) when the considered home DCs are paired with each of Google DCs. For both algorithms, when data size factor=0.2, the different component costs (i.e., residential, delay, and migration costs) remain constant for all Google DCs; while for data size factor=1, pairing the home DCs with GO-EUW and GO-ASE incurs more delay cost compared to the case that these home DCs are paired with other Google DCs (i.e., GO-USC and GO-USE). Therefore, the latter pairing set (i.e., pairing the home DCs with GO-USC and GO-USE) gains more cost savings.

Fig. 4 also suggests that, when AZ-USS and AZ-USC are home DCs, Amazon DCs can be another suitable set of DCs to pair with, except AM-BRS as this DC is more expensive than home DCs in both network and storage services. OOP and NOOP gain cost savings about 32–35% and 18–20% respectively when paired with AM-USW(O), AM-USE, and AM-EUC for data size factor=1, and likewise 67–68% and 52–56% for data size factor=0.2. In contrast, when home DCs are paired with other Amazon DCs (i.e., AM-JAE, AM-ASA and AM-AUE), both algorithms attain lower cost savings due to two reasons: (i) these Amazon DC are more expensive in storage for backup objects and network as compared to the former subset of Amazon DCs (i.e., AM-USW(O), AM-USE, and AM-EUC), and (ii) they also impose more delay cost owing to their longer distance to home DCs.

Fig. 5 depicts the cost savings of OOP and NOOP when home DCs are GO-USC and GO-USE. The results show that the cost is reduced when these home DCs are paired with AZ-USS and AZ-USC that offer the same price in the network and storage. This cost reduction is 65–67% for OOP and 53–55% for NOOP when data size factor=0.2, while for data size factor=1, both algorithms approach the same cost saving (about 35–37%—see Figs. 5b and 5d). The reason behind this result for data size factor=1 is that both algorithms determine to migrate a low proportion of objects (about 25%) at roughly the same time. Moreover, the results show that OOP can save more cost by (2–3%) when GO-USC is paired with AZ-USC rather than AZ-USS (Figs. 5a and 5b) for both data size

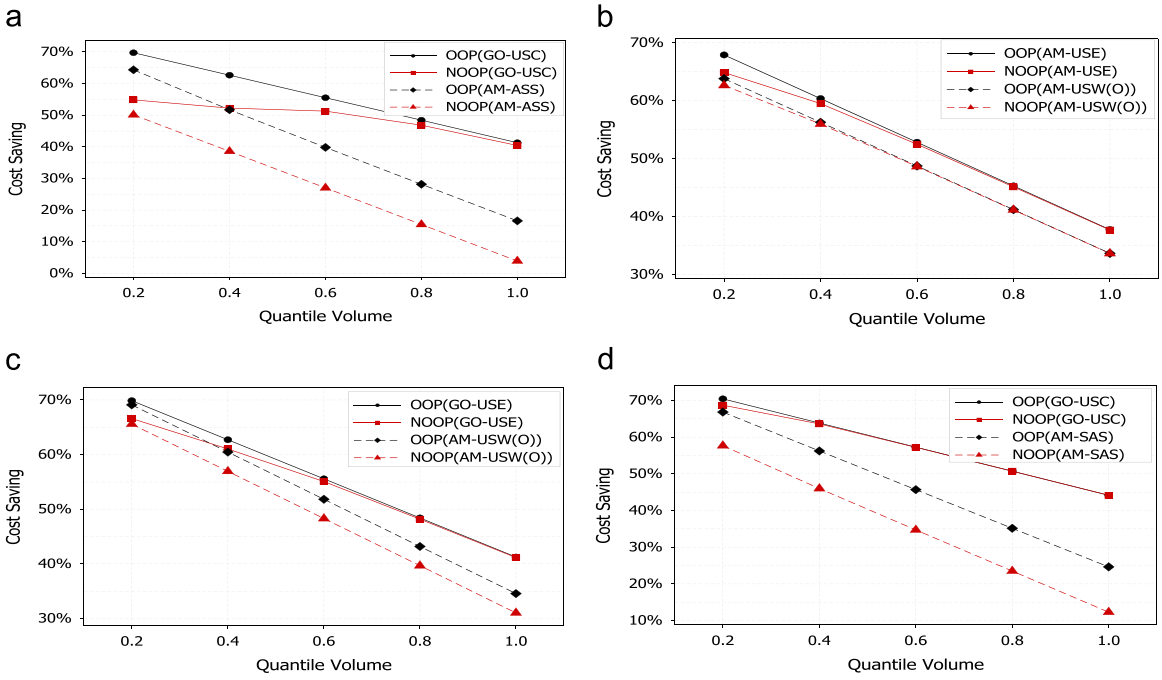

Fig. 7. Cost saving of OOP and NOOP algorithms for the home DC of Azure, Google, and Amazon when the data size factor is varied. The first (resp. last) two legends indicate DC with maximum (resp. minimum) cost saving when they are paired with the home DC. (a) Data center AZ-USS. (b) Data center GO-USE. (c) Data center AM-USE. (d) Data center AM-USW(C).

factor values. This implies that users in GO-USC and their friends (in other DCs) incur less delay cost when their read/write requests are sent to AZ-USC.

Fig. 5 also demonstrates that the home DCs can benefit from pairing with three Amazon DCs, but the benefit is less than pairing with the specified Azure DCs. This is because of AM-USW(O) and AM-USE are more expensive than Azure DCs in terms of network cost, while AM-USW(C) is more expensive in both storage and network costs. However, the results (Figs. 5c and 5d) depict an exception in which pairing GO-USE with AM-USE can save more cost than pairing GO-USE with Azure DCs. This happens because both GO-USE and AM-USE are in eastern USA, and thus read/write requests (mainly coming from this region) incur less delay cost.

Fig. 6 depicts the obtained cost savings from pairing each DC when home DCs are: AM-USW(O), AM-USE, and AM-USW(C).

Figs. 6a–6d show that AM-USW(O) and AM-USE can benefit from pairing with at most three Azure DCs, all Google DCs, and one Amazon DC. Pairing with Google DCs can bring more cost savings than with Amazon DC[4] which in turn, save more cost than with Azure DCs (i.e., AZ-USS and AZ-USS) especially for NOOP when data size factor = 1. The reason behind this is: (i) objects tend to migrate to Google DCs with the cheapest storage cost for backup objects, and (ii) Amazon DCs offer discount in network cost if the objects are migrated between two Amazon DCs. For data size factor = 1, NOOP can save around 32–35% cost when AM-USW(O) is paired with AM-USE (Fig. 6b) or vice versa (Fig. 6d) by utilizing this discount, and around 20–25% when both home DCs are paired with AZ-USS and AZ-USC. The difference in cost savings of NOOP between the two pairing settings (i.e., pairing the home DCs with Amazon DC and with Azure DCs) is at most 15% (Fig. 6a) for the data size factor = 1, and likewise at most 5% when data size factor is 0.2 (Fig. 6c).

Figs. 6e and 6f show that AM-USW(C), as a home DC, can be paired with more DCs to save cost because its storage and network costs are substantially higher than the cost in storage and network services offered by other DCs. As it can be easily seen in the figures, the most and least profitable DCs for pairing respectively are Google and Azure DCs for both algorithms and for both values of the data size factor.

As shown in Fig. 6e. for OOP and NOOP, pairing the home DC AM-USW(C) with three Amazon DCs (i.e., AM-USW(O), AM-USE, and AM-EUN) gains roughly the same cost savings than when it is paired with Google DCs. For OOP (resp. NOOP), the pairing with the remaining Amazon DCs (i.e., AM-EUC and AM-ASS) cuts the same cost (resp. more cost) as it is paired with Azure DCs. In fact, for both algorithms and for both values of the data size factor, paring with AM-USW(O), AM-USE, and AM-EUN can offer more cost savings than AM-EUC and AM-ASS.

As depicted in Fig. 6f, for both algorithms, pairing the home DC AM-USW(C) with Google DCs outperforms pairing the home DC with AM-USW(O), AM-USE, and AM-EUN at most by 10% in cost saving. For two pairing sets, pairing the home DC with AM-EUC and AM-ASS, and with Azure DCs, OOP gains the same cost saving in both sets, while NOOP achieves a twice cost savings in the latter pairing set in comparison to the former pairing set.

The results can be justified as the aforementioned three Amazon DCs (i.e., AM-USW(O), AM-USE, and AM-EUN) benefit from the discount in network price for moving data across Amazon DCs. The amount of discount is about 3/4 of the network price for moving data out to the Internet. AM-EUC takes the advantage of the same amount of discount but its low profitability happens due to the small difference in the price of both storage classes (5% less for IAS and 8% more for SS) compared to these prices of the home DC AM-USW(C). For AM-ASS, the proposed algorithms achieve a lower cost saving because the amount of discount is about 1/4 of the network price.

5.2.3. *The impact of data size factor value*

We evaluate the effect of the data size factor value by varying it from 0.2 to 1 with the step size of 0.2. The read and write requests for all data size factor values are fixed based on the default value as

[4] AM-USW(O) as a home DC is paired with AM-USE (Figs. 6a and 6b), and AM-USE as a home DC is paired with AM-USW(O) (Figs. 6c and 6d).

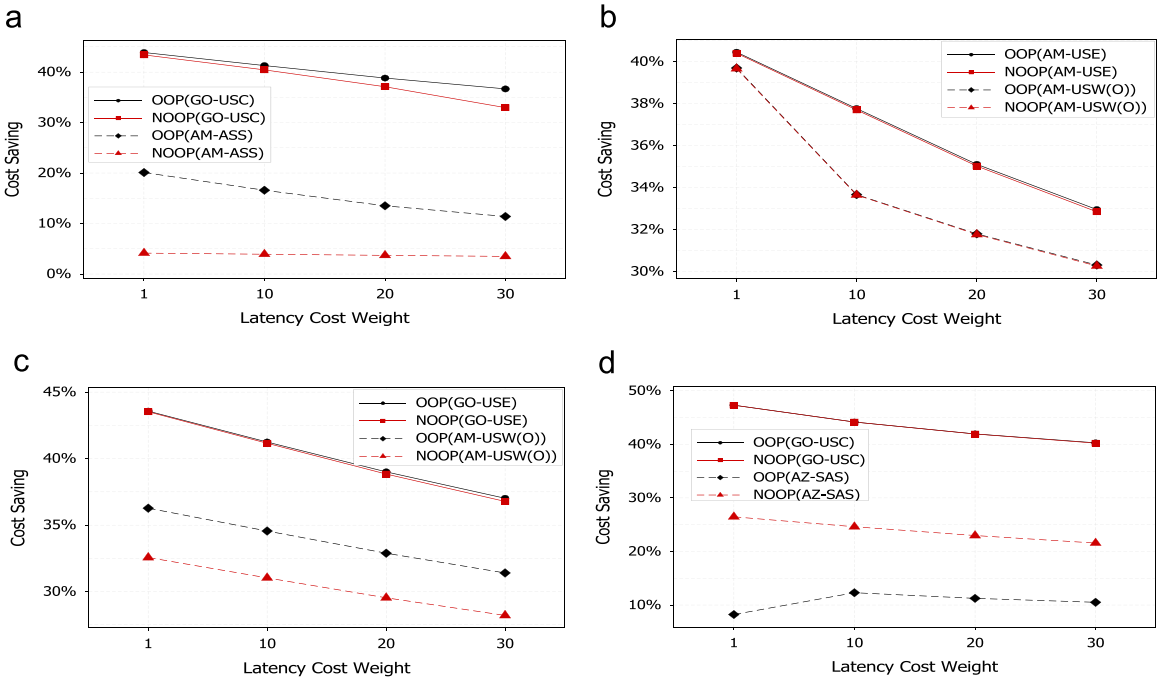

Fig. 8. Cost saving of OOP and NOOP algorithms for the home DC of Azure, Google, and Amazon when the latency cost weight is varied. The first (resp. last) two legends indicate DC with the maximum (resp. minimum) cost saving when they are paired with the home DC. (a) Data center AZ-USS. (b) Data center GO-USE. (c) Data center AM-USE. (d) Data center AM-USW(C).

in Table 1. This setting implies that as the data size factor value is smaller, the data is more read- and write-intensive.

For the sake of brevity, from hereafter (excluding Section 5.2.6), we report the results only for the most populated Azure DC (AZ-USS), Google DC (GO-USE), and the two most populated Amazon DCs (AM-USE and AM-USW(C)), as home DCs. We also consider the pairing of these DCs with two DCs: the ones with maximum and minimum cost savings, where the value of data size factor is 1. Note that these DCs can be easily recognized in Figs. 4b, 5d, 6d, and 6f. For example, Fig. 4b depicts AZ-USS, as a home DC, can achieve maximum (resp. minimum) cost saving when it is paired with GO-USC (resp. AM-ASS).

As shown in Fig. 7, as the data size factor value increases, the cost saving decreases. This is because when the data size factor value is small, the network cost dominates the total cost, and the proposed algorithm exploits more difference between network prices offered by the paired DCs. On the contrary, as the value of data size factor increases, the storage cost becomes more

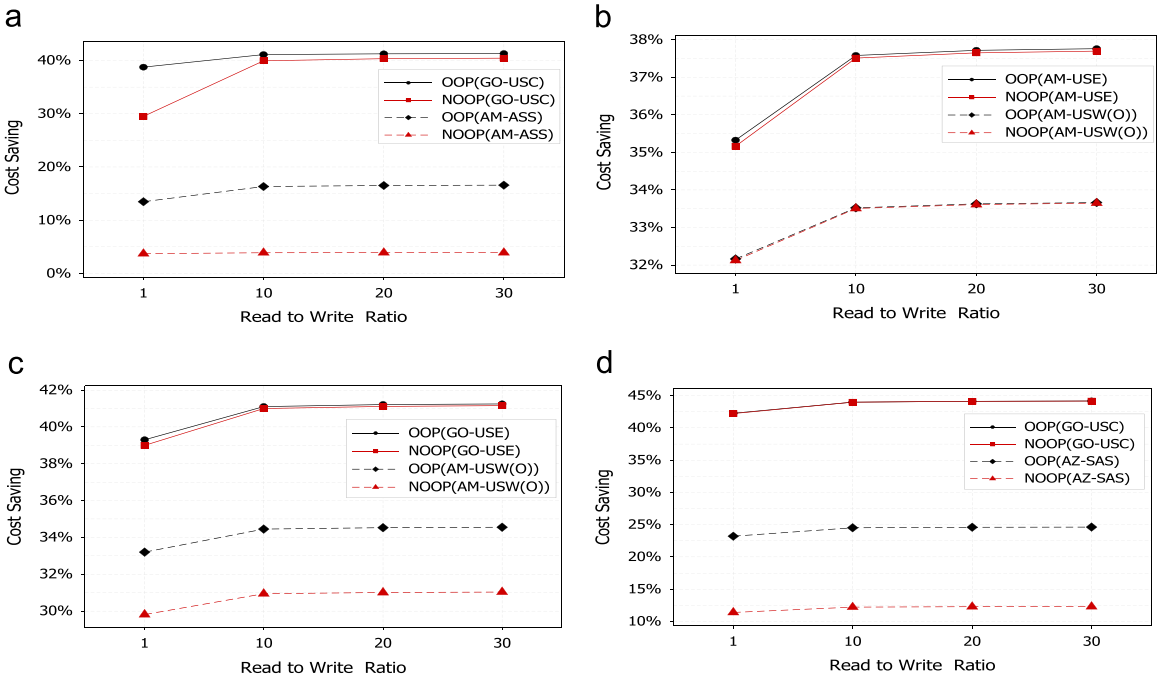

Fig. 9. Cost saving of OOP and NOOP algorithms for the home DC of Azure, Google, and Amazon when the write to read ratio is varied. The first (resp. last) two legends indicate DC with the maximum (resp. minimum) cost saving when they are paired with the home DC. (a) Data center AZ-USS. (b) Data center GO-USE. (c) Data center AM-USE. (d) Data center AM-USW(C).

Table 3
Cost saving of OOP and NOOP (shown in bracket), and the potential DCs pairing with four home DCs when the access patterns on the objects are Normal and Random.

Home DC	Access pattern	Azure	Google	Amazon
AZ-USS	Normal	–	2–3%[−(11–18)%]	2−4% [−(3–7)%]
	Random	–	2–3%[−(1–13)%]	2–3% [−(3–5)%]
	–	–	All DCs	USW(O,C), USE
GO-USE	Normal	4%[−(7)%]	–	4–5% [(2–4)%]
	Random	4%[−(2)%]	–	3–4% [(1–2)%]
	–	USS, USC	–	USW(O,C), USE
AM-USE	Normal	4% [0%]	4% [1–3%]	4% [2%]
	Random	4% [−5%]	3–4% [−(0–4)%]	3% [1%]
	–	USS, USC	All DCs	USW(O)
AM-USW(C)	Normal	7–8% [−3%]	4–10% [3–9%]	6–9% [6–7%]
	Random	7% [1%]	3–9% [3–6%]	6–8% [5–6%]
	–	USS, USC	All DCs	USW(O), USE

important and thus the difference between storage prices offered by the paired DCs comes into play for optimization. We can also see that both algorithms approach the same cost saving value (referred as convergence point) in the case of maximum cost savings obtained from pairing DCs. The convergence point for each home DC is: data size factor=0.6 for GO-USE and AM-USE, and data size factor=0.8 for AM-USW(C). This implies that both algorithms decide to migrate the objects at roughly the same time, and consequently they almost achieve the same cost saving.

5.2.4. The impact of latency cost weight

We study the effects of *latency cost weight* by varying it from 1 to 30 on the cost performance of the algorithms on the pairing DCs as already discussed in Section 5.2.3. As shown in Fig. 8, when the value of *latency cost weight* increases, the cost savings gradually decrease (1–10%) for both algorithms. This is due to the fact that as the *latency cost weight* value grows, the delay cost in dual cloud-based storage increases and the impact of other parts of the total cost (i.e., residential and migration costs) diminishes. In fact, the growth in the latency cost weight reduces the potential for exploiting the difference cost between storage and network, which leads in the cost saving reduction. In summary, for both algorithms, as the *latency cost weight* value increases, the delay cost comes as a vital factor in the cost saving, while the impact of difference between storage and network costs on the cost saving decrease. As a result, a dual cloud-based storage prefers to store more objects locally.

5.2.5. The impact of read to write ratio

We explore the effects of *read to write ratio* by varying it from 1 to 30 on the cost performance of the proposed algorithms on pairing with DCs discussed in the previous section. Fig. 9 depicts that, as the ratio of read to write increases, the cost savings gradually increase at most 3% for OOP and 10% for NOOP. This indicates more exploitation of pricing differences in the case of network costs between the two paired DCs as the ratio grows. Also, as it can be seen, for the paired DCs with minimum cost saving, the OOP algorithm gains 3–10% more cost saving than NOOP, except for the home DC GO-USE. On the contrary, for the paired DCs with maximum cost saving, both algorithms approach the same cost saving. This is because the total cost is dominated by the storage cost, where in all paired DCs, Google DCs offer the same storage cost. For the data size factor values between 0.2 and 0.6 in the case of the paired DCs with maximum cost saving (not shown in results), OOP outperforms NOOP in the cost savings. The reason is that the total cost is dominated by the network cost and home DCs are paired with different DCs (except Google DCs) in terms of network cost.

5.2.6. The impact of the access pattern of reads/writes on objects

We finally investigate the cost performance of the proposed algorithms when the access pattern of reads/writes on the objects follows different distributions, i.e., Normal and Random (Recall that the access pattern of reads/writes on objects follows a Long-tail distribution (Atikoglu et al., 2012), as the default). Although Normal and Random access patterns are not compatible with hot- and cold-spot status definition for objects, we investigate the impact of these patterns separately to find out whether the algorithms can still cut cost. And if so, to what extent? We conduct the experiment for four home DCs: AZ-USS, GO-USE, AM-USE, and AM-USW(C). We use the default value of read to write ratio, latency cost weight, and data size factor, as shown in Table 1. Table 3 gives the cost savings for OOP and NOOP (shown in brackets).

As shown in Table 3, OOP can save cost for all home DCs under both access patterns. For AZ-USS, the algorithm saves more cost if it is paired with Amazon DCs rather than Google DCs under Normal access pattern by incurring less migration cost. For GO-USE, OOP cuts the cost by 4% if it is paired with Azure DCs and cuts slightly more costs with Amazon DCs under Normal access pattern. In contrast to the two discussed home DCs, AM-USE and AM-USW(C), as home DCs, can achieve cost reduction by pairing with DCs of all cloud providers (i.e., Microsoft Azure, Google, and Amazon). The cost saving obtained from these home DCs with Amazon DCs (i.e., AM-USW(O) and AM-USE, see rows 3 and 4 under column "Amazon" in Table 3) approaches the one achieved through pairing with Google (i.e., all Google DCs) and Azure DCs (i.e., AZ-USS and AZ-USC). This is because home DCs exploit the discount on the network cost when data is moved across two Amazon DCs.

From the results of the OOP algorithm, we observe that the cost saving obtained from pairing potential DCs (see their name in rows 3 and 4 of Table 3) with home DC AM-USW(C) is approximately two times more than that achieved by pairing these DCs with home DC AM-USE. The reason is that the difference between storage and network prices offered by AM-USW(C) and its paired DCs is substantially high, while for AM-USE is comparatively low. The result for AM-USW(C) shows that the cost savings are still considerable under both Normal and Random access patterns when a home DC offers services with prices that are significantly different from those prices provided by other DCs. We can find similar DCs (i.e., a significant difference between two DCs in terms of price) in Asia-Pacific and Brazil regions as well. The companies in these regions can leverage pairing of their DCs with DCs in other regions to reduce cost not only for objects with hot- and cold-spot status, but also for objects accessed under Normal and Random access patterns.

Contrary to OOP, NOOP achieves less cost savings and in some circumstances this algorithm is not even cost-efficient. Under both access patterns, NOOP is not profitable when AZ-USS is paired with DCs of Google and Amazon. NOOP triggers more migrations that increase the total cost. Moreover, the results indicate that pairing GO-USE with Azure DCs is not cost-efficient (−7% for Normal and −2% for Random access pattern), while pairing of GO-USE with Amazon DCs can still cut cost by 4% for Normal and 2% for Random access pattern. NOOP brings more cost saving for pairing both of the Amazon home DCs with the other potential DCs, as compared to the pairing of home DCs such as AZ-USS and GO-USE with the other DCs. For instance, AM-USW(C) achieves cost savings under all circumstances except pairing with Azure DCs under Normal access pattern. As already mentioned, this is

because AM-USW(C) offers more expensive storage and network as compared to its paired DC, and object migration between these paired DCs under both access patterns is cost-effective.

In summary, according to the experimental results, one can conclude that OOP is cost efficient under both access patterns. NOOP is not profitable for pairing Azure DCs with Google and Amazon DCs, while it can be cost-effective in other pairing situations, such as Amazon DCs with Azure, Google, and Amazon DCs, as well as pairing Google DC with Amazon DC. We realized that NOOP can be profitable especially when the paired DCs can exploit the discounted price in the network cost for moving data across DCs belonging to the same provider. This discount is currently offered by Amazon, and it is likely that Google and Azure would offer their customers the same discount in the near future.

6. Conclusions and future work

Choosing storage options across CSPs for time-varying workload is critical for optimizing data management cost. In particular, issues such as when should an object be migrated and in which storage class it should be stored need to be addressed. We consider a fine-grained architecture and propose two algorithms that determine optimal (resp. near optimal) placement of the object with (resp. without) the knowledge of the future workload. Such a fine-grained architecture provides evidence that one can achieve cost savings in Geo-replicated system where a home DC can be paired with different DCs during the lifetime of the object.

In the future, we plan to apply the cost model in Geo-replicated data stores so that the cost saving is maximized, while the weak and strong consistency models are both guaranteed. Furthermore, we are interested to investigate the effect of users' mobility on the cost optimization, and to determine when it is necessary to change the home DC (in the proposed model we assumed the closest DC as user's home DC) of the users when they move across the globe.

Acknowledgments

We thank Rodrigo N. Calheiros, Adel Nadjaran Toosi, Sareh Fotuhi Piraghaj, Chenhao Qu, and Yali Zhao for their valuable comments in improving the quality of the paper. This work is supported by the Australian Research Council Future Fellowship and Discovery Project Grant.

References

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

Atikoglu, B., Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M., 2012. Workload analysis of a large-scale key-value store. In: Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'12), ACM, New York, NY, USA, 2012, pp. 53–64.

Beloglazov, A., Abawajy, J., Buyya, R., 2012. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener. Comput. Syst. 28 (5), 755–768 (Special Section: Energy Efficiency in Large-Scale Distributed Systems).

Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P., 2011. Depsky: dependable and secure storage in a cloud-of-clouds. In: Proceedings of the Sixth European Conference on Computer Systems (EuroSys'11), ACM, New York, NY, USA, pp. 31–46.

Broberg, J., Buyya, R., Tari, Z., 2009. Metacdn: harnessing storage clouds for high performance content delivery. J. Netw. Comput. Appl. 32 (5), 1012–1022.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R., 2011. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. 41 (1), 23–50.

Chen, F., Guo, K., Lin, J., La Porta, T., 2012. Intra-cloud lightning: Building cdns in the cloud. In: INFOCOM, 2012 Proceedings, IEEE, Orlando, Florida, USA, pp. 433–441.

Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., Woodford, D., 2013. Spanner: google's globally distributed database. ACM Trans. Comput. Syst. 31 (3), 8:1–8:22.

Elmore, A.J., Das, S., Agrawal, D., El Abbadi, A., 2011. Zephyr: Live migration in shared nothing databases for elastic cloud platforms. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, ACM, New York, NY, USA, pp. 301–312.

Fang, Y., Wang, F., Ge, J., 2010. A task scheduling algorithm based on load balancing in cloud computing. In: Proceedings of the 2010 International Conference on Web Information Systems and Mining, WISM'10, Springer-Verlag, Berlin, Heidelberg, pp. 271–277.

Gao, P.X., Curtis, A.R., Wong, B., Keshav, S., 2012. It's not easy being green, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, ACM, New York, NY, USA, pp. 211–222.

Hajjat, M., Sun, X., Sung, Y.-W.E., Maltz, D., Rao, S., Sripanidkulchai, K., Tawarmalani, M., 2010. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. SIGCOMM Comput. Commun. Rev. 40 (4), 243–254.

Jiao, L., Li, J., Xu, T., Du, W., Fu, X., 2014. Optimizing cost for online social networks on geo-distributed clouds. Netw. IEEE/ACM Trans. 99, 1.

Kotla, R., Alvisi, L., Dahlin, M., 2007. Safestore: a durable and practical storage system. In: Proceedings of the USENIX Annual Technical Conference (ATC'07), USENIX Association, Berkeley, CA, USA, pp. 10:1–10:14.

Latency-it Costs You. ⟨http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it⟩.

Li, R., Wang, S., Deng, H., Wang, R., Chang, K.C., 2012. Towards social user profiling: unified and discriminative influence model for inferring home locations, in: The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'12, Beijing, China, August 12–16, pp. 1023–1031.

Lin, W., Liang, C., Wang, J.Z., Buyya, R., 2014. Bandwidth-aware divisible task scheduling for cloud computing. Softw.: Pract. Exp. 44 (2), 163–174.

Liu, J., Pacitti, E., Valduriez, P., de Oliveira, D., Mattoso, M., 2016. Multi-objective scheduling of scientific workflows in multisite clouds. Future Gener. Comput. Syst. 63, 76–95 (Modeling and Management for Big Data Analytics and Visualization).

Liu, Z., Chen, Y., Bash, C., Wierman, A., Gmach, D., Wang, Z., Marwah, M., Hyser, C., 2012. Renewable and cooling aware workload management for sustainable data centers. In: Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS'12, ACM, New York, NY, USA, pp. 175–186.

Liu, Z., Lin, M., Wierman, A., Low, S.H., Andrew, L.L., 2011. Greening geographical load balancing. In: Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS'11, ACM, New York, NY, USA, pp. 233–244.

Luo, J., Rao, L., Liu, X., 2014. Temporal load balancing with service delay guarantees for data center energy cost optimization. IEEE Trans. Parallel Distrib. Syst. 25 (3), 775–784.

Luo, J., Rao, L., Liu, X., 2015. Spatio-temporal load balancing for energy cost optimization in distributed internet data centers. IEEE Trans. Cloud Comput. 3 (3), 387–397.

Papagianni, C., Leivadeas, A., Papavassiliou, S., 2013. A cloud-oriented content delivery network paradigm: modeling and assessment. IEEE Trans. Depend. Secur. Comput. 10 (5), 287–300.

Puttaswamy, K.P., Nandagopal, T., Kodialam, M., 2012. Frugal storage for cloud file systems. In: Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12), ACM, New York, NY, USA, pp. 71–84.

Qiu, X., Li, H., Wu, C., Li, Z., Lau, F.C.M., 2015. Cost-minimizing dynamic migration of content distribution services into hybrid clouds. IEEE Trans. Parallel Distrib. Syst. 26 (12), 3330–3345.

Rao, L., Liu, X., Xie, L., Liu, W., 2010. Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In: Proceedings of the 29th Conference on Information Communications, INFOCOM'10, IEEE Press, Piscataway, NJ, USA, pp. 1145–1153.

Rodriguez, M.A., Buyya, R., 2015. A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds. In: 2015 44th International Conference on Parallel Processing (ICPP), pp. 839–848.

Salahuddin, M.A., Elbiaze, H., Ajib, W., Glitho, R.H., 2015. Social network analysis inspired content placement with qos in cloud based content delivery networks. In: 2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6–10, 2015, pp. 1–6.

Tak, B.C., Urgaonkar, B., Sivasubramaniam, A., 2011. To move or not to move: The economics of cloud computing. In: Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing. HotCloud'11, USENIX Association, Berkeley, CA, USA, pp. 5–5.

The Google Maps Geocoding API. ⟨https://developers.google.com/maps/documentation/geocoding/intro⟩.

Tran, N., Aguilera, M.K., Balakrishnan, M., 2011. Online migration for geo-distributed storage systems. In: Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'11, USENIX Association, Berkeley, CA, USA, pp. 15–15.

Van Den Bossche, R., Vanmechelen, K., Broeckhove, J., 2013. Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. Future Gener. Comput. Syst. 29 (4), 973–985.

Wikipedia. Hierarchical Storage Management (hsm). ⟨http://en.wikipedia.org/wiki/

Hierarchicalstoragemanagement⟩ (accessed 14.01.13).
Wood, T., Cecchet, E., Ramakrishnan, K.K., Shenoy, P., van der Merwe, J., Venkataramani, A., 2010. Disaster recovery as a cloud service: Economic benefits & deployment challenges. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, Berkeley, CA, USA, pp. 8–8.
Wu, Z., Butkiewicz, M., Perkins, D., Katz-Bassett, E., Madhyastha, H.V., 2013. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13), ACM, New York, NY, USA, pp. 292–308.
Yu, S., Lin, X., Misic, J., Shen, X., 2015. Networking for Big Data, Chapman & Hall/CRC Big Data Series. CRC Press, Taylor & Francis Group, Boca Raton, FL.
Zhang, L., Wu, C., Li, Z., Guo, C., Chen, M., Lau, F., 2013. Moving big data to the cloud: an online cost-minimizing approach. IEEE J. Sel. Areas Commun. 31 (12), 2710–2721.