Optimizing Cost for Online Social Networks on Geo-Distributed Clouds

Lei Jiao, Jun Li, *Senior Member, IEEE,* Tianyin Xu, Wei Du, and Xiaoming Fu, *Senior Member, IEEE*

***Abstract*—Geo-distributed clouds provide an intriguing platform to deploy online social network (OSN) services. To leverage the potential of clouds, a major concern of OSN providers is optimizing the monetary cost spent in using cloud resources while considering other important requirements, including providing satisfactory quality of service (QoS) and data availability to OSN users. In this paper, we study the problem of cost optimization for the dynamic OSN on multiple geo-distributed clouds over consecutive time periods while meeting pre-defined QoS and data availability requirements. We model the cost, the QoS, as well as the data availability of the OSN, formulate the problem, and design an algorithm named cosplay. We carry out extensive experiments with a large-scale real-world Twitter trace over 10 geo-distributed clouds all across the US. Our results show that, while always ensuring the QoS and the data availability as required, cosplay can reduce much more one-time cost than the state-of-the-art methods, and it can also significantly reduce the accumulative cost when continuously evaluated over 48 months, with OSN dynamics comparable to real-world cases.**

***Index Terms*—Online social network, Cloud computing, Optimization models and methods, Performance analysis and evaluation**

I. Introduction

INTERNET services today are experiencing two remarkable changes. One is the unprecedented popularity of online social networks (OSNs), where users build social relationships, and create and share contents with one another. The other is the rise of clouds. Often spanning multiple geographic locations, clouds provide an important platform for deploying distributed online services. Interestingly, these two changes tend to be combined. While OSN services often have a very large user base and need to scale to meet demands of users worldwide, geo-distributed clouds that provide Infrastructure-as-a-Service can match this need seamlessly, as well as tremendous resource and cost efficiency advantages. Infinite on-demand cloud resources can accommodate the surges of user requests; flexible pay-as-you-go charging schemes can save the investments of service providers; and cloud infrastructures also free service providers from building and operating ones' own data centers. Indeed, a number of OSN services are increasingly deployed on clouds, *e.g.*, Sonico, CozyCot, and Lifeplat [2].

Migrating OSN services towards geographically distributed clouds must reconcile the needs from several different aspects. First, OSN providers want to optimize the monetary cost spent in using cloud resources. For instance, they may wish to minimize the storage cost when replicating users' data at more than one cloud, or minimize the inter-cloud communication cost when users at one cloud have to request the data of others that are hosted at a different cloud. Moreover, OSN providers hope to provide OSN users with satisfactory quality of service (QoS). To this end, they may want a user's data and those of her friends to be accessible from the cloud closest to the user, for example. Last but not least, OSN providers may also be concerned of data availability, *e.g.*, ensuring the number of users' data replicas to be no fewer than a specified threshold across clouds. Addressing all such needs of cost, QoS, and data availability is further complicated by the fact that an OSN continuously experiences dynamics, *e.g.*, new users join, old users leave, and the social relations also vary.

Existing work on OSN service provisioning either pursues least cost in a single site without the QoS concern as in the geo-distribution case [29], [33], or aims for least inter-data-center traffic in the case of multiple data centers without considering other dimensions of the service [25], *e.g.*, data availability. More importantly, the models in all such work do not capture the monetary cost of resource usage and thus cannot fit the cloud scenario. There are some work on cloud-based social video [37], [38], focusing on leveraging online social relationships to improve video distribution, which is only one of the many facets of OSN services; most optimization research on multi-cloud and multi-data-center services are not for OSN [23], [39], [30], [10]. They fail to capture the OSN features such as social relationships and user interactions, and thus their models are not applicable to OSN services.

In this paper, we study the problem of optimizing the monetary cost of the dynamic, multi-cloud-based OSN, while ensuring its QoS and data availability.

We first model the cost, the QoS, and the data availability of the OSN service upon clouds. Our cost model identifies different types of costs associated with multi-cloud OSN while capturing social locality [29], [33], an important feature of the OSN service that most activities of a user occur between herself and her neighbors. Guided by existing research on OSN growth and our analysis of real-world OSN dynamics, our

Manuscript received December 27, 2013; revised July 9, 2014; accepted September 6, 2014; approved by IEEE/ACM Transactions on Networking Editor L. Ying. This work was supported in part by the EU under Projects FP7 IRSES MobileCloud (grant no. 612212) and FP7 ITN CleanSky (grant no. 607584), and by the Simulation Science Center, sponsored by the Volkswagen Foundation and the State of Lower Saxony, Germany. A preliminary version of this work appeared in IEEE ICNP 2012 [20].

Lei Jiao and Xiaoming Fu are with the Institute of Computer Science, University of Göttingen, 37077 Göttingen, Germany (e-mail: jiao@cs.uni-goettingen.de; fu@cs.uni-goettingen.de).

Jun Li is with the Department of Computer and Information Science, University of Oregon, Eugene, OR 97403, USA (e-mail: lijun@cs.uoregon.edu).

Tianyin Xu is with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093, USA (e-mail: tixu@cs.ucsd.edu).

Wei Du is with the Department of Electrical Engineering and Computer Science, University of Liège, 4000 Liège, Belgium (e-mail: wei.du@ulg.ac.be).

model approximates the total cost of OSN over consecutive time periods when the OSN is large in user population but moderate in growth, enabling us to achieve the optimization of the total cost by independently optimizing the cost of each period. Our QoS model links the QoS with OSN users' data locations among clouds. For every user, all clouds available are sorted in terms of a certain quality metric (*e.g.*, access latency); therefore every user can have the most preferred cloud, the second most preferred cloud, *etc*. The QoS of the OSN service is better if more users have their data hosted on clouds of a higher preference. Our data availability model relates with the minimum number of replicas maintained by each OSN user.

We then base on these models to formulate the cost optimization problem which considers QoS and data availability requirements. We prove the NP-hardness of our problem. We propose an algorithm named cosplay based on our observations that *swapping the roles* (*i.e.*, master or slave) of a user's data replicas on different clouds can not only lead to possible cost reduction, but also serve as an elegant approach to ensuring QoS and maintaining data availability. Compared with existing approaches, cosplay reduces cost significantly and finds a substantially good solution of the cost optimization problem, while guaranteeing all requirements are satisfied. Furthermore, not only can cosplay reduce the one-time cost for a cloud-based OSN service, it can also solve a series of instances of the cost optimization problem and thus minimize the aggregated cost over time by estimating the heavy-tailed OSN activities [13], [35] during runtime.

As part of our extensive research, we distribute a real-world geo-social Twitter dataset of 321,505 users with 3,437,409 social relations over 10 clouds all across the US in a variety of settings. Compared with existing alternatives, including some straightforward methods such as the *greedy* placement (the common practice of many online services [34], [32]), the *random* placement (the *de facto* standard of data placement in distributed DBMS such as MySQL and Cassandra [24]), and some state-of-the-art algorithms such as SPAR [29] and METIS [22], cosplay produces better data placements. While meeting all requirements, it can reduce the one-time cost by up to about 70%. Further, over 48 consecutive months with OSN dynamics comparable to real-world cases, compared with the *greedy* placement, continuously applying cosplay can reduce the accumulative cost by more than 40%. Our evaluations also demonstrate quantitatively that the trade-off among cost, QoS, and data availability is complex; an OSN provider may have to incorporate cosplay to all three dimensions. For instance, according to our results, the benefits of cost reduction decline when the requirement for data availability is higher, whereas the QoS requirement does not always influence the amount of cost that can be saved.

The remainder of this paper is structured as follows. Section II describes our models of the cost, QoS, and data availability of the OSN service over multiple clouds. Section III formulates the cost optimization problem. Section IV elaborates our cosplay algorithm, as well as our considerations and insights. Section V demonstrates and interprets our evaluations. Section VI discusses some related issues such as complexity and optimality. We contrast our work with related work in Section VII and conclude this paper in Section VIII.

II. Models

Targeting the OSN service over multiple clouds, we begin with identifying the types of costs related to cloud resource utilization: the storage cost for storing users' data, the inter-cloud traffic cost for synchronizing data replicas across clouds, the redistribution cost incurred by the cost optimization mechanism itself, and some underlying maintenance cost for accommodating OSN dynamics. We discuss and approximate the total cost of the multi-cloud OSN over time. Afterwards, we propose a vector model to capture the QoS of the OSN service, show the features of this model, and demonstrate its usage. Finally, we model the OSN data availability by linking it with the number of each user's data replicas.

A. System Settings

Clouds and OSN users are all geographically distributed. Without loss of generality, we consider the single-master-multi-slave paradigm [32], [12]: each user has only one master replica and several slave replicas of her data, where each replica is hosted at a different cloud. When signing in to the OSN service, a user always connects to her master cloud, *i.e.*, the cloud that hosts her master replica, and every read or write operation conducted by a user goes to her master cloud first.

We assume the placement of OSN users' replicas follows the social locality scheme [29], [33]. Observing that most activities of an OSN user happen between the user and her neighbors (*e.g.*, friends on Facebook or followees on Twitter), this scheme requires that a user's master cloud host a replica (either the master or a slave) of every neighbor of the user. This way, every user can read the data of her friends and her own from a single cloud, and the inter-cloud traffic only involves the write traffic for maintaining the consistency among a user's replicas at different clouds. Social locality has multi-fold advantages: given that there are often many more reads than writes in an OSN service [14], it can thus save a large proportion of the inter-cloud traffic; this scheme also incurs a much lower storage consumption than full replication in that the full replication requires every cloud to maintain a data replica for every user. Note that for a user with one master and r slaves, a write on this user's data always incurs r corresponding inter-cloud writes to maintain consistency. We consider eventual consistency in our work, and assume issues such as write conflicts are tackled by existing techniques.

B. Modeling the Storage and the Inter-cloud Traffic Cost

OSN is commonly abstracted as a social graph, where each vertex represents a user and each edge represents a social relation between two users [26]. We extend this model by associating three distinct quantities with every user. (1) A user has a *storage cost*, which is the monetary cost for storing one replica of her data (*e.g.*, profile, statuses) in the cloud for one billing period. (2) Similarly, a user has a *traffic cost*, which is the monetary cost during a billing period because of the inter-cloud traffic. As mentioned earlier, due to social locality,

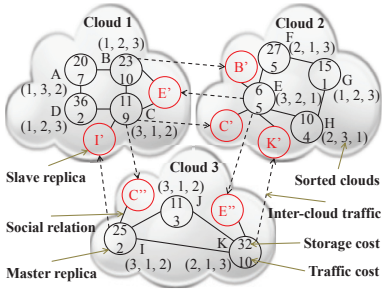


Fig. 1: Storage and inter-cloud traffic cost

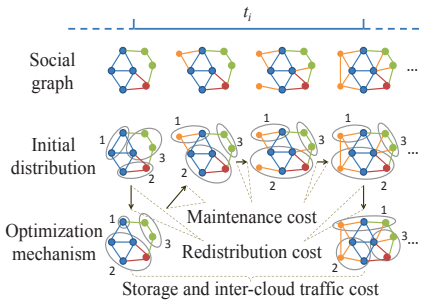


Fig. 2: Different types of costs

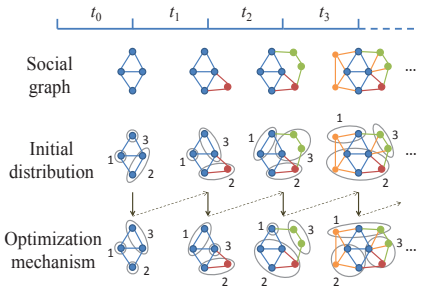


Fig. 3: Cost over consecutive time periods

in our settings the inter-cloud traffic only involves writes (*e.g.*, posting tweets, leaving comments). We do not consider *intra*-cloud traffic, no matter read or write, as it is free of charge [4], [1]. (3) A user has a sorted list of clouds for the purpose of QoS, as will be described in Section II-E.

Fig. 1 is an example where 11 users are hosted by 3 clouds. Black circles represent each user's master replica, and red ones represent the slave replicas of neighbors to ensure social locality. Solid lines are social relations and dotted arrows are the synchronization traffic. Within each black circle, the value on the top is the storage cost of a user, and the value at the bottom is the traffic cost. For Fig. 1, the total storage cost is 330 and the total inter-cloud traffic cost is 50.

Besides the cost described above, note that the read/write operations themselves are charged based on the number of operations performed [1]. As we require the social locality for every user, the number of read operations performed by a user on all replicas of hers and her friends depends neither on the number of the replicas nor on the placement of the replicas. The charging for read operations is thus out of the scope of our optimization of replica placement. In contrast, the number of the write operations performed by a user on all replicas of hers and her friends depends on the number and the placement of the replicas. Fortunately, its charging can be included just as part of a user's traffic cost. For example, let $\tau_u = w_u T$ denote user u's traffic cost, where w_u is the number of writes performed on u's data and T is the average traffic cost incurred by a single write. Then, one can include the cost charged for a single write into T so that optimizing the total inter-cloud traffic cost by our model can actually optimize the sum of the traffic and the read/write operations cost.

We make further assumptions. When calculating the costs, we assume that all clouds have the same billing prices. In reality, resource usage of clouds from different providers or at different locations may be charged at different prices. Such cases can be easily addressed by associating a proper weight with each cloud in our model, and our proposed algorithm, as shown later, can also straightforwardly adapt to these cases. We also assume that each cloud can provide "infinite" resources on demand to an OSN service provider, a guarantee often provided by a cloud provider to its customers.

C. Modeling the Redistribution Cost

An important part of our cost model is the cost incurred by the optimization mechanism itself, which we call the *redistribution* cost. We generally envisage that an optimization mechanism is devised to optimize the cost by moving data across clouds to optimum locations, thus incurring such cost. The redistribution cost is essentially the inter-cloud traffic cost, but in this paper we use the term inter-cloud traffic to specifically refer to the inter-cloud write traffic for maintaining replica consistency, and treat the redistribution cost separately.

We expect that the optimization is executed at a per-billing-period granularity (*e.g.*, per-month) for the following reasons. First, this frequency is consistent with the usual charging unit for a continuously running and long-term online service. The OSN provider should be enabled to decide whether to optimize the cost for each billing period, according to her monetary budget and expected profit, *etc.* Also, applying any cost optimization mechanism too frequently may fail the optimization itself. At the time of writing this paper, the real-world price of inter-cloud traffic for transferring some data *once* is quite similar to that of storing the same amount of data for an entire billing period [4], [1]. As a result, moving data too frequently can incur more redistribution cost that can hardly be compensated by the saved storage and inter-cloud traffic cost. Without loss of generality, we assume that the optimization mechanism is applied only *once* at the *beginning* of each billing period, *i.e.*, the redistribution cost only occurs at the beginning of every billing period.

D. Approximating the Total Cost

Consider the social graph in a billing period. As it may vary within the period, we denote the final steady snapshot of the social graph in this period as $G' = (V', E')$, and the initial snapshot of the social graph at the beginning of this period as $G = (V, E)$. Thus, the graph G experiences various changes—collectively called ΔG—to become G', where $\Delta G = (\Delta V, \Delta E)$, $\Delta V = V' - V$, and $\Delta E = E' - E$.

Now consider the total cost incurred during a billing period. Denoting the total cost, the storage plus the inter-cloud traffic cost, the maintenance cost, and the redistribution cost during a period as Ψ, $\Phi(\cdot)$, $\Omega(\cdot)$, and $\Theta(\cdot)$, respectively, we have

$$\Psi = \Phi(G) + \Phi(\Delta G) + \Omega(\Delta G) + \Theta(G).$$

The storage cost in $\Phi(G) + \Phi(\Delta G)$ is for storing users' data replicas, including the data replicas of existing users and of those who just join the service in this period. The inter-cloud traffic cost in $\Phi(G) + \Phi(\Delta G)$ is for propagating all users' writes to maintain replica consistency. The redistribution cost $\Theta(G)$ is the cost of moving data across clouds for optimization; it is only incurred at the beginning of a period, following

our previous assumption. There is also some underlying cost $\Omega(\Delta G)$ for maintenance, described as follows.

The maintenance cost $\Omega(\Delta G)$ is used to capture the cost spent on handling OSN changes. When a new user joins the OSN service, the service selects a cloud and places this user's data there. Some time later after this initial placement and no later than the end of the current billing period, the OSN service must maintain social locality for this user and her neighbors, including creating new slave replicas on involved clouds as needed, incurring maintenance cost. However, in reality, when the OSN user base reaches a certain scale, $\Omega(\Delta G)$ and $\Phi(\Delta G)$ become negligible as the size of ΔG (*i.e.*, $|\Delta V|$) becomes much smaller than that of G (*i.e.*, $|V|$). Existing research observes that real-world OSNs usually have an S-shape growth [11], [16]. As the user population becomes larger, the increment of the total number of users or social relations will decay exponentially [18], [40]. Let us look at the *monthly growth rate* (*i.e.*, $|\Delta V|/|V|$) in some real examples. According to Facebook [5], after its user population reached 58 million by the end of 2007, it grew with an average monthly rate below 13% through 2008 and 2009, a rate below 6% through 2010, and then a rate below 4% until the end of 2011 when it reached 845 million. For Twitter, its average monthly growth rate was less than 8% in most months between March 2006 and September 2009 [8]. Similar rates were also observed for YouTube and Flickr [27].

Therefore, we derive an approximated cost model as

$$\Psi \approx \Phi(G) + \Theta(G)$$

which we will focus on throughout the rest of this paper. Note that calculating Ψ requires the storage cost and the traffic cost of each user in G. For any cost optimization mechanism that runs at the beginning of a billing period, an estimation is required to predict each user's costs during this billing period. Let's for now deem that the costs can be predicted and known. We defer the discussion on cost prediction to Section V-A.

Fig. 2 and 3 illustrate different types of costs during a single billing period and consecutive billing periods. The numbers in the figures are the cloud IDs. Slave replicas are not drawn for the ease of presentation.

Note that, for the initial data placement, the OSN service may use various pre-specified strategies to choose a cloud, such as choosing the one with the lowest access latency for the user [34], [32]. At this point the OSN cannot determine an optimum cloud in terms of cost for a new user, as it knows neither the user's storage cost (except for a certain reserved storage such as storing a profile with pre-filled fields) nor her traffic cost for the current billing period. We assume that an OSN places a new user's data on her most preferred cloud.

E. Modeling QoS and Data Availability

Sorting clouds. Among all clouds, one cloud can be better than another for a particular user in terms of certain metric(s) (*e.g.*, access latency, security risk). For instance, concerning access latency, the best cloud to host the data requested by a user is likely the geographically closest cloud to that user. Given N clouds and $|V|$ users, with cloud IDs $\{1, \ldots, N\}$ (denoted as $[N]$ hereafter) and user IDs $\{1, \ldots, |V|\}$ (denoted as $[|V|]$ hereafter), clouds can be sorted for user u as $\vec{c_u} = (c_{u1}, c_{u2}, ..., c_{uN})$, where $c_{ui} \in [N]$, $\forall i \in [N]$. For any cloud c_{ui}, c_{uj}, $i < j$, we deem that c_{ui} is more preferred than c_{uj}; in other words, placing user u's data on the former provides better service quality to this user than the latter. The clouds $\{c_{u1}, c_{u2}, ..., c_{uj}\}$, $\forall j \in [N]$ are thus the j most preferred clouds of user u, and the cloud c_{uj} is the jth most preferred cloud of user u. This sorting approach provides a unified QoS abstraction for every user while making the underlying metric transparent to the rest of the QoS model.

Defining QoS. We define the QoS of the entire OSN service as a vector $\vec{q} = (\vec{q}[1], \vec{q}[2], ..., \vec{q}[N])$, with

$$\vec{q}[k] = \frac{1}{|V|} \sum_{u=1}^{|V|} \sum_{j=1}^{k} f_u(m_u, j), \forall k \in [N],$$

where m_u denotes the ID of the cloud that hosts the master data replica of user u, $f_u(i, j)$ is a binary function that equals to 1 if cloud i is user u's jth most preferred cloud but 0 otherwise. Therefore, $\vec{q}[k]$ is the ratio of users whose master data are placed on *any* of their respective k most preferred clouds over the entire user population. This CDF-style vector allows OSN providers to describe QoS at a finer granularity.

Let us refer back to Fig. 1 as an example, where the vector associated with each circle represents the sorted cloud IDs for the corresponding user. We see that out of all the 11 users, 7 are hosted on their first most preferred cloud, 10 on either of their two most preferred clouds, and all users on any of their three most preferred clouds. Thus, the QoS is $\vec{q} = (\frac{7}{11}, \frac{10}{11}, 1)$.

Comparing QoS. There can be different data placements upon clouds. Each may result in a different corresponding QoS vector. For two QoS vectors $\vec{q}_a$ and $\vec{q}_b$ representing two placements respectively, we deem that the former placement provides QoS no better than the latter, *i.e.*, $\vec{q}_a \leq \vec{q}_b$, if every element of the former vector is no larger than the corresponding element of the latter, *i.e.*, $\vec{q}_a[k] \leq \vec{q}_b[k]$, $\forall k \in [N]$.

QoS requirement. We model the QoS requirement as two vectors $\vec{Q}_l$ and $\vec{Q}_u$, $\vec{Q}_l \leq \vec{Q}_u$ that serve as a lower bound and an upper bound, respectively. In order to meet the QoS requirement, a data placement must have a QoS $\vec{q}$ that meets $\vec{Q}_l \leq \vec{q} \leq \vec{Q}_u$. Specified by the OSN provider, $\vec{Q}_l$ captures the worst QoS that can be tolerated and $\vec{Q}_u$ captures the best QoS that can be provided. Note that we do not require $\vec{Q}_u$ to represent the placement of every user's data on her first most preferred cloud. $\vec{Q}_u$ can be set as any valid QoS vector, subject to the OSN provider's customized policies and considerations.

As an example, let us see how $\vec{Q}_l$ can express "80% of all users must access data in no more than 200 ms." In this case, clouds are sorted according to access latency for every user. For any user u, we can calculate that only putting her master data replica on any of her n_u, $n_u \in [N]$ most preferred clouds can grant her the latency of no more than 200 ms. By denoting $n_{min} = \min\{n_u | \forall u \in [|V|]\}$, this requirement can thus be expressed by setting $\vec{Q}_l[n_{min}] = 0.8$. If $n_{min} \neq 1$, then $\vec{Q}_l[k]$, $\forall k \in \{1, \ldots, n_{min} - 1\}$ can be set as any value as long as $0 \leq \vec{Q}_l[k_1] \leq \vec{Q}_l[k_2] \leq 0.8$, $1 \leq k_1 < k_2 < n_{min}$. In fact, $\vec{Q}_l$ can express any fine-grained requirement such as

"95% of users' access must be satisfied within 500 ms, 80% be satisfied within 200 ms and 65% be satisfied within 90 ms."

Data availability requirement. An OSN provider specifies the data availability requirement by indicating the minimum number of every user's slave replicas. We denote it using a number R, $R \in \{0, \ldots, N-1\}$, where N is the number of clouds. In order to meet the data availability requirement, each user must maintain slave replicas no fewer than R. If the number of a user's slave replicas to maintain social locality is no smaller than R, the data availability requirement for this user has already been met and this user does not have to own more slaves; in contrast, besides the slaves to maintain social locality, if a user does not have enough slaves to meet the data availability requirement, then this user must have more slaves to ensure that the total number of her slaves is equal to R.

III. Problem

With the models defined in Section II, we are interested in the following problem: given an existing data placement upon N clouds of OSN $G(V, E)$ with $|V|$ users, find out the optimal data placement with the minimal total cost—*i.e.*, the sum of the storage and inter-cloud traffic cost $\Phi(G)$ and the redistribution cost $\Theta(G)$ for implementing this optimal placement from the existing placement—while ensuring QoS and data availability meet pre-defined requirements.

We introduce the following notations in order to formulate the problem. m_{ui} and s_{ui} are binary decision variables. The former equals to 1 if in the optimal placement user u's *master* replica is placed on cloud i, and 0 otherwise. The latter equals to 1 if in the optimal placement u has a *slave* replica placed on cloud i, and 0 otherwise. m'_{ui} and s'_{ui} are also binary, and are counterparts of m_{ui} and s_{ui} respectively in the *existing* placement. μ_u is the storage cost for storing one master or slave replica of user u. τ_u is the traffic cost for synchronizing one slave replica of user u. β is the coefficient for converting the storage cost of a replica to the redistribution cost of moving this replica across clouds. $e_{uv} \in E$ if user u and user v are neighbors. $f_u(i, j)$ is a binary function indicating whether cloud i is user u's jth most preferred cloud (as introduced in Section II-E). The QoS requirement is given by two vectors $\vec{Q}_l$ and $\vec{Q}_u$, and the data availability requirement is given by a number R. We formulate the problem as follows.

minimize:

$$\Phi(G) + \Theta(G)$$

where

$$\Phi(G) = \sum_{u=1}^{|V|} (\mu_u \sum_{i=1}^{N} (m_{ui} + s_{ui}) + \tau_u \sum_{i=1}^{N} s_{ui})$$

$$\Theta(G) = \sum_{u=1}^{|V|} (\beta \mu_u \sum_{i=1}^{N} (\max\{(m_{ui} + s_{ui}) - (m'_{ui} + s'_{ui}), 0\}))$$

subject to:

$$\sum_{i=1}^{N} m_{ui} = 1, \forall u \in [|V|] \quad (1)$$

$$m_{ui} + s_{ui} \leq 1, \forall u \in [|V|], \forall i \in [N] \quad (2)$$

$$m_{vj} + s_{vj} = 1, j = \sum_{i=1}^{N} (i m_{ui}), \text{if } e_{uv} \in E, \forall u, v \in [|V|] \quad (3)$$

$$\sum_{i=1}^{N} s_{ui} \geq R, \forall u \in [|V|] \quad (4)$$

$$\vec{Q}_l[k] \leq \frac{1}{|V|} \sum_{u=1}^{|V|} \sum_{j=1}^{k} f_u(\sum_{i=1}^{N} (i \cdot m_{ui}), j) \leq \vec{Q}_u[k], \forall k \in [N] \quad (5)$$

Constraint (1) ensures that every user has a single master replica. Constraint (2) ensures that no master and slave replicas of the same user are co-located on a common cloud. Constraint (3) ensures the social locality. Constraint (4) ensures the data availability. Constraint (5) ensures that the QoS of the data placement meets the QoS requirement. All constraints apply to both the existing data placement and the optimal placement. Here we do not write the existing case for the ease of presentation. Our cost optimization problem is NP-hard. We provide the proof in the appendix of this paper.

IV. Algorithm

Our cost optimization problem is an Integer Programming (IP) problem. The huge user population of real-world OSN services translates into a huge number of decision variables, and the NP-hardness of our problem makes it impossible to be efficiently solved by existing general-purpose IP solvers. We thus seek practical heuristics. We propose cosplay, an optimization algorithm that iteratively swaps the roles of master and slave replicas on different clouds to reach the optimal placement.

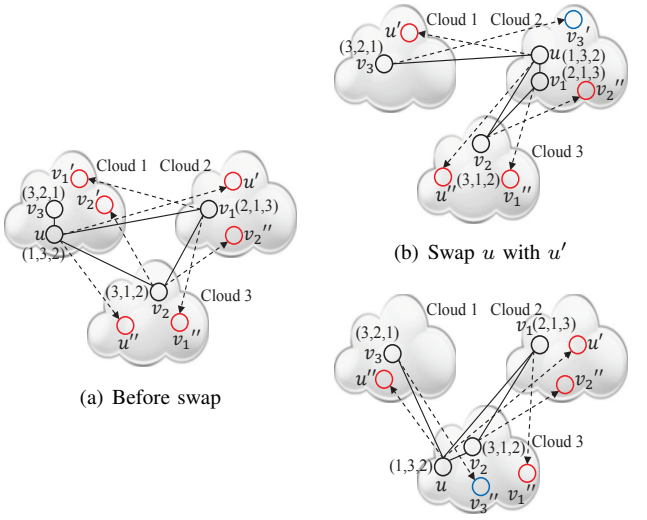
A. Observations

Our algorithm is inspired by three observations below when swapping a master replica and a slave replica of a user. In this what we call a role-swap process, the master replica becomes a slave replica and the slave becomes the master. We use Fig. 4 and Fig. 5 to illustrate our observations, where lines and circles in these figures have the same meanings as in Fig. 1, and each user has 1 unit of storage cost and 1 unit of traffic cost. Note that while symbols like u, v are supposed to denote users throughout this paper, we also use them to denote the master replicas of the corresponding users in the figures here.

Observation 1: *Role-swap can lead to possible cost reduction.* Fig. 4 is a simple example with 4 users hosted by 3 clouds. For user u, we may choose to swap the roles of replica u with replica u' (as in Fig. 4(b)), or swap the roles of replica u with replica u'' (as in Fig. 4(c)), while maintaining the social locality. Before the swap in Fig. 4(a), there are 10 units of replica storage and 6 units of inter-cloud traffic. After the swap, as in both Fig. 4(b) and Fig. 4(c), there are 9 units of replica storage and 5 units of inter-cloud traffic. We thus

save 1 unit of replica storage and 1 unit of inter-cloud traffic by paying 1 unit of redistribution cost (caused by copying the replica v_3 to create a new replica v_3' in Fig. 4(b), or v_3'' in Fig. 4(c)). Overall, we can achieve 1 unit of cost reduction.

Observation 2: *Because of the QoS requirement, not every role-swap is feasible, although it may reduce cost.* When multiple role-swaps for a user are available, we must choose the one(s) meeting QoS requirements. The two different role-swap choices taken in Fig. 4(b) and in Fig. 4(c) result in the same amount of cost reduction. Let us suppose every cloud has an ID and every user has a sorted list of preferred clouds as shown in the figure. Before the swap, $\vec{q} = (0.75, 0.75, 1)$. If the QoS requirement is given by $\vec{Q}_u = (1, 1, 1)$ and $\vec{Q}_l = (0.5, 0.75, 1)$, then we should choose Fig. 4(c) instead of Fig. 4(b), because the QoS of the former is $\vec{q} = (0.5, 0.75, 1)$, which still meets the QoS requirement, and the QoS of the latter is $\vec{q} = (0.5, 0.5, 1)$, which violates the QoS requirement.

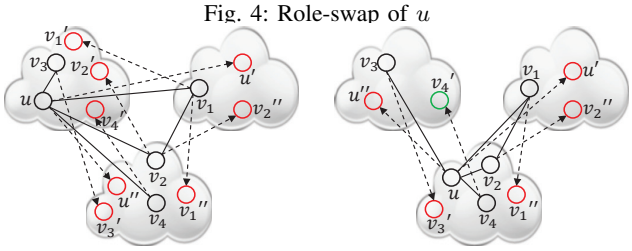


(a) Before swap

(b) Swap u with u'

(c) Swap u with u''

Fig. 4: Role-swap of u



(a) Before swap

(b) Swap u with u'': v_4' is kept

Fig. 5: Role-swap of u

Observation 3: *Every user needs to have slave replicas no fewer than a given number, which can be maintained when performing role-swaps.* We use Fig. 5 for illustration, where 5 users are hosted on 3 clouds, and for simplicity, we do not show the sorted lists of cloud IDs. Suppose in this example, the required minimum number of slaves of every user is 1. After maintaining social locality, users u, v_1, v_2, and v_4 already meet this requirement, but user v_3 still needs a slave replica for data availability, *i.e.*, v_3' in this figure. Now we swap the roles of replicas u and u'' for cost reduction. After this swap, note that the slave replica v_4' is not needed for maintaining the social locality of u, but it is still needed to satisfy the data availability of user v_4. Therefore, in this case, the cost reduction is 4 units, instead of 6 units if we remove v_4'.

B. Our Algorithm: Cosplay

Inspired by the above three observations, we employ a series of role-swaps to maximize the total cost reduction while maintaining data availability and ensuring QoS requirements. Our algorithm follows a greedy approach in using role-swaps and requiring that *every* applied role-swap reduce cost. The more cost reduction each role-swap has and the more role-swaps are applied, the more total cost reduction we can achieve. Note that our algorithm computes a better placement, and it does not physically manipulate data. When our algorithm terminates, data are role-swapped or moved (in the case of redistribution) from existing locations to new locations in order to implement the new placement output by our algorithm.

We describe our cosplay algorithm as follows: starting with an existing placement, the algorithm runs and repeats the two procedures of single role-swaps (made up of Algorithms 1, 3, 5) and double role-swaps (made up of Algorithms 2, 4, 5) one after the other, and it terminates when neither of them can be further executed to reduce the cost or when a specified number of iterations are executed.

Single role-swaps. In each iteration, select a user randomly. For each feasible role-swap between this user's master and one of her slaves, calculate the cost reduction. Then, choose the role-swap with the largest cost reduction and apply it. Repeat this until no further cost can be reduced.

Double role-swaps. In each iteration, select a user randomly, and pair this user with each of her neighbors whose master is on a different cloud. For each such pairs, first check if the following pair of role-swaps is feasible: one between the selected user's master and her slave on the neighbor's cloud and the other between the neighbor's master and her slave on the selected user's cloud. If feasible, calculate the cost reduction of these two role-swaps. Then, choose the pair with the largest cost reduction and apply the two role-swaps. Repeat this until no further cost can be reduced.

Whether a single role-swap or a double role-swap, three basic but non-trivial operations of cosplay are needed: determining whether it is feasible, calculating its cost reduction, and swapping the roles of involved replicas. We elaborate how to efficiently achieve these operations below.

1) Determining Feasibility: Algorithms 1 and 2 determine the feasibilities of a single role-swap and a double role-swap, respectively. Algorithm 1 checks whether applying a role-swap would make the current QoS out of the range specified by the QoS lower bound and upper bound. In Algorithm 1, user u's master and slave replicas are on cloud c_{ui} and c_{uj} respectively. Algorithm 2 invokes Algorithm 1, where users u and v are selected, with their masters on cloud c_{ui} and c_{vi} and slaves on c_{uj} and c_{vj}, respectively. Note that applying one role-swap can change the current QoS, and the feasibility of the next role-swap must be considered based on the new QoS. We do not show the function ***adjustQoS***(c_{ui}, c_{uj}) as it is very simple, adjusting $\vec{q}$ in a way similar to Algorithm 1.

Algorithm 1: ***isSingleFeasible***(c_{ui}, c_{uj})

Data: c_{ui}, c_{uj}: u's ith and jth most preferred cloud
$\vec{Q}_l$, $\vec{Q}_u$: the QoS lower and upper bounds
$\vec{q}$: the current QoS of the placement
begin
  if $i < j$ **then** // c_{ui} is more preferred than c_{uj}
    for *each* $k \in [i, j-1]$ **do**
      if $\vec{q}[k] - \frac{1}{|V|} < \vec{Q}_l[k]$ **then**
        return $false$;
  else
    for *each* $k \in [j, i-1]$ **do**
      if $\vec{q}[k] + \frac{1}{|V|} > \vec{Q}_u[k]$ **then**
        return $false$;
  return $true$;

Algorithm 2: ***isDoubleFeasible***$(c_{ui}, c_{uj}, c_{vi}, c_{vj})$

Data: c_{ui}, c_{uj}: u's ith and jth most preferred cloud
c_{vi}, c_{vj}: v's ith and jth most preferred cloud
begin
  if ***isSingleFeasible***(c_{ui}, c_{uj}) **then**
    adjustQoS(c_{ui}, c_{uj});
    if ***isSingleFeasible***(c_{vi}, c_{vj}) **then**
      adjustQoS(c_{uj}, c_{ui});
      return $ture$;
    else
      adjustQoS(c_{uj}, c_{ui});
  if ***isSingleFeasible***(c_{vi}, c_{vj}) **then**
    adjustQoS(c_{vi}, c_{vj});
    if ***isSingleFeasible***(c_{ui}, c_{uj}) **then**
      adjustQoS(c_{vj}, c_{vi});
      return $ture$;
    else
      adjustQoS(c_{vj}, c_{vi});
  return $false$;

2) Calculating Cost Reduction: Algorithms 3 and 4 specify the calculations of the cost reduction of a single role-swap and a double role-swap, respectively. Here we highlight three of our insights about Algorithm 3 as follows.

Local computation. To calculate the cost reduction for a role-swap between user u's master and her slave, an intuitive option would be calculating the difference between the total cost of the old placement (*i.e.*, the one before applying the role-swap) and that of the new placement (*i.e.*, the one after applying the role-swap). However, doing so involves accessing *every* user and calculating the total cost twice, which can cause considerable computation overhead given a large social graph. In fact, we observe that the cost reduction can be calculated by accessing only local information. The cost reduction only depends on the storage and traffic cost of user u and her neighbors, and the locations of their replicas in the old and new placements. If on user u's master cloud we store a slave replica of her neighbor v to maintain the social locality for u, *and* if v has no other neighbors of her own on this cloud, a role-swap between user u's master and her slave will make this slave replica of v useless, and this replica is thus a candidate for elimination (whether it can be eliminated further depends on data availability as described below). In one word, a local computation is sufficient to calculate the reduced cost.

Algorithm 3: ***calcCostReducSingle***(m_u, s_u)

Data: m_u: the cloud hosting u's master replica
s_u: the cloud hosting u's slave replica
μ_u, τ_u: u's storage cost and traffic cost
P_e: the existing placement of all users' replicas
Δ: the cost that can be reduced
δ_u: the number of u's slaves that can be reduced
ρ: the number of slaves that incur the redistribution cost
Rmv_m_u: boolean: true if removing u's replica on m_u, false if not
Rmv_s_u: boolean: true if removing u's replica on s_u, false if not
begin
  $\Delta \Leftarrow 0, \delta_u \Leftarrow 0, \delta_v \Leftarrow 0, \rho \Leftarrow 0$;
  $Rmv\_m_u \Leftarrow true$, $Rmv\_s_u \Leftarrow true$;
  /* calculate the reduced cost incurred by replicas of u's neighbors */
  for *each* $v \in$ *u's neighbors* **do**
    $\delta_v \Leftarrow 0$, $\rho \Leftarrow 0$;
    if $m_v \neq m_u$ **then**
      if u *is* v*'s only neighbor on* m_u **then**
        $\delta_v \Leftarrow \delta_v + 1$;
        if v *has no replica on* m_u *in* P_e **then**
          $\rho \Leftarrow \rho - 1$;
      if $m_v = s_u$ **then**
        $Rmv\_s_u \Leftarrow false$;
    if $m_v \neq s_u$ **then**
      if v *has no slave replica on* s_u **then**
        $\delta_v \Leftarrow \delta_v - 1$;
        if v *has no replica on* s_u *in* P_e **then**
          $\rho \Leftarrow \rho + 1$;
      if $m_v = m_u$ **then**
        $Rmv\_m_u \Leftarrow false$;
    if $\neg$ *(*v *has* R *slave replicas and* $\delta_v > 0$*)* **then**
      $\Delta \Leftarrow \Delta + (\mu_v + \tau_v)\delta_v - \beta\mu_v\rho$;
  /* calculate the reduced cost incurred by replicas of u's own */
  $\rho \Leftarrow 0$;
  if $Rmv\_s_u = true$ **then**
    $\delta_u \Leftarrow \delta_u - 1$;
    if u *has no replica on* s_u *in* P_e **then**
      $\rho \Leftarrow \rho + 1$;
  if $Rmv\_m_u = true$ **then**
    $\delta_u \Leftarrow \delta_u + 1$;
    if u *has no replica on* m_u *in* P_e **then**
      $\rho \Leftarrow \rho - 1$;
  if $\neg$ *(*u *has* R *slave replicas and* $\delta_u > 0$*)* **then**
    $\Delta \Leftarrow \Delta + (\mu_u + \tau_u)\delta_u - \beta\mu_u\rho$;
  return Δ;

Algorithm 4: ***calcCostReducDouble***(m_u, s_u, m_v, s_v)

begin
  $\Delta\Psi_1 \Leftarrow$ ***calcCostReducSingle***(m_u, s_u);
  swapRole(m_u, s_u);
  $\Delta\Psi_2 \Leftarrow$ ***calcCostReducSingle***(m_v, s_v);
  swapRole(s_u, m_u);
  return $\Delta\Psi_1 + \Delta\Psi_2$;

Redistribution cost. The cost of redistribution incurred by a role-swap depends on the new placement where this role-swap is applied, and the existing placement which is the input to our cosplay algorithm. In the new placement, when a slave needs to be created on a cloud for social locality, we check to see if it did not exist on the cloud in the existing placement. If so, the cost of creating this slave is added to the redistribution cost incurred by this role-swap. Similarly, when a slave is to

Algorithm 5: ***swapRole***(m_u, s_u)

Data: m_u: the cloud hosting u's master replica
s_u: the cloud hosting u's slave replica
δ_u: the number of u's slaves that can be reduced
Rmv_m_u: boolean: true if removing u's replica on m_u, false if not
begin
  $\delta_u \Leftarrow 0, \delta_v \Leftarrow 0, Rmv\_m_u \Leftarrow true$;
  for *each* $v \in u$*'s neighbors* **do**
    $\delta_v \Leftarrow 0$, $\rho \Leftarrow 0$;
    if $m_v \neq m_u$ **then**
      if u *is* v*'s only neighbor on* m_u **then**
        $\delta_v \Leftarrow \delta_v + 1$;
    if $m_v \neq s_u$ **then**
      if v *has no slave replica on* s_u **then**
        $\delta_v \Leftarrow \delta_v - 1$;
      if $m_v = m_u$ **then**
        $Rmv\_m_u \Leftarrow false$;
    if $\neg$ *(*v *has* R *slave replicas and* $\delta_v > 0$*)* **then**
      if $m_v \neq m_u$ **then**
        if u *is* v*'s only neighbor on* m_u **then**
          Remove v's slave at m_u;
      if $m_v \neq s_u$ **then**
        if v *has no slave replica on* s_u **then**
          Create v's slave at s_u;
  // Do the role-swap
  if $Rmv\_m_u = true$ **then**
    $\delta_u \Leftarrow \delta_u + 1$;
  u's master at m_u becomes a slave;
  u's slave at s_u becomes the master;
  if $\neg$ *(*u *has* R *slave replicas and* $\delta_u > 0$*)* **then**
    if u *has a slave replica on* m_u *and* $Rmv\_m_u = true$ **then**
      Remove u's slave at m_u;

be removed as it is no longer needed, we check whether this slave existed on its current cloud in the existing placement. If not, this slave was created by a previous role-swap and the incurred redistribution cost of creating this slave has already been counted. We thus subtract this cost from the redistribution cost of the role-swap.

Data availability. Whether to remove a slave or not does not only depend on social locality, but also on the data availability requirement. Creating slaves is always fine because it never violates the data availability requirement. We must ensure that if we remove a slave replica, the number of slaves of this user is still no fewer than the pre-specified number, thus maintaining the data availability for this user. If we cannot remove a slave due to the data availability reason, this user should not be considered when calculating cost reduction of a role-swap that involves this user, and the slave is also not touched when performing the role-swap.

3) Swapping Roles of Replicas: Algorithm 5 describes the operation of swapping the roles of a user u's master on cloud m_u and her slave on cloud s_u. Swapping the roles does not simply involve u's replicas alone; instead, it may also involve removing or creating her neighbors's slave replicas due to social locality and data availability. The flow of this algorithm shares some similarities with Algorithm 1, specifically calculating the cost reduction of a role-swap before it is performed is actually simulating how the cost would be affected if the role-swap occurred.

V. Evaluations

We carry out extensive evaluations by placing real-world Twitter data over 10 clouds all across the US. We demonstrate significant one-time and accumulated cost reductions with cosplay compared to existing approaches, while always ensuring QoS and data availability requirements. By varying the experimental settings, we also investigate the complex trade-off among cost, QoS, and data availability.

A. Data Preparation

Collecting data. We crawled Twitter during March and April 2010 in a breadth-first manner, consistent with previous OSN crawls [26], [36]. We collected 3,117,553 users with 23,883,149 social relations. For each user, we have her profile, tweets, and her list of followers. Among all the users, 1,157,425 users provide location information in their profiles.

Sorting clouds for each user. Due to the lack of publicly available dataset on latencies between OSN users and OSN sites, we cannot sort clouds for users in terms of latency. However, the geographic distance is widely used as an important QoS metric in previous work [17], [15]. With our Twitter dataset, we thus sort clouds for each user in terms of the real-world geographic distance between a user and the clouds.

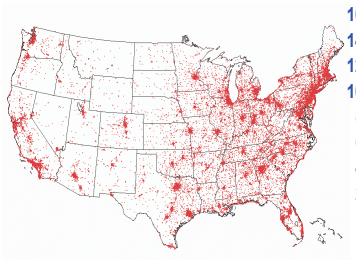


Fig. 6: User locations

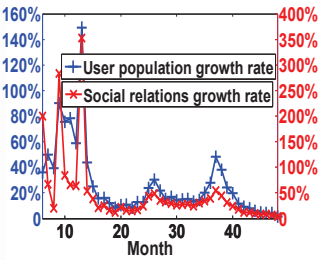


Fig. 7: Monthly growth rate

We focus on users who are geographically located in the US. With the help of the database of US Board on Geographic Names [7] and Google Maps, we convert users' text locations to geo-coordinates (*i.e.*, [latitude, longitude]). Out of all these users, we extract the largest connected component of 321,505 users with 3,437,409 social relations as the input to our evaluations. We then select 10 cities all across the US as cloud locations: Seattle (WA), Palo Alto (CA), Orem (UT), Chicago (IL), San Antonio (TX), Lansing (MI), Alexandria (LA), Atlanta (GA), Ashburn (VA), and New York (NY). All these locations have real-world cloud data centers [3]. Afterwards, we sort all clouds for each user in terms of the respective geographic distance between a user and the clouds. Fig. 6 plots the locations of all the 321,505 users.

Extracting monthly OSN snapshots. To evaluate cosplay for OSN with dynamics, we extract monthly graph snapshots out of our largest connected component. Under the assumption that each user publishes her first tweet and forms her social relations immediately after she joins the service, and with the time stamp of each tweet and the follower list of each user, we find that the earliest user in our largest connected component joined Twitter in March 2006 and the latest user joined Twitter in February 2010. We thus can extract 48 monthly social graphs. Fig. 7 shows the monthly growth rates of our 48

graphs. The user base growing at about 15% since the 16th month is a good approximation of the real-world OSN growth, as discussed in Section II.

Obtaining monthly costs for each user. We calculate costs by multiplying the *unit price* with the data size, and use such calculated costs of each user for each of the 48 months as the ground truth. We use \$0.125/GB/month for storage cost and \$0.12/GB for inter-cloud traffic cost [4]. With each tweet and its time stamp, we can easily find how much data a user publishes in a given month (*i.e.*, the traffic size of that month), and how much data this user has accumulated by the end of a given month (*i.e.*, the storage size of that month).

In the real world, when executing cosplay in consecutive billing periods, it needs an estimate of each user's costs at the beginning of each period. Existing research tells that users' online activities are bursty and the inter-activity time is heavy-tailed [13], [35], making the estimation of future activities and data size difficult. However, our goal here is not to pursue estimation accuracy, instead, our interest is in the trending size of a user's produced data so that we can make the correct role-swap. We exploit the *Exponentially Weighted Moving Average*, a common approach for similar purposes, to estimate the number of activities in a future time period with the smoothing factor $\alpha = 0.9$ to capture the burstiness of user activities. By multiplying it with the average data size of all previous activities, we obtain the estimation of the data size in a future time period. As a result, we obtain 48 monthly graphs with the estimated costs of each user for each month.

B. Experimental Settings

We run two groups of evaluations. In the first group, with our largest February 2010 social graph as input, we compare the costs and the QoS' of the data placements produced by the *greedy* method, the *random* method, SPAR, METIS, and cosplay. We also investigate how the costs are influenced by the data availability requirement and by the QoS requirement. We ensure social locality for all approaches for fair comparison. The *greedy* method places every user's master on her first most preferred cloud. The *random* method assigns a user's master to a cloud randomly. For SPAR, we implement it ourselves, and we treat each social relation between two users as an edge creation event and create a random permutation of all events to produce the edge creation trace as input, following the method suggested in [29]. For METIS, there is an open-source implementation from its authors. We use its option of minimizing the inter-partition communication. We use each user's storage cost plus her traffic cost as the vertex size (in METIS' terminology) to create its input. For cosplay, we use the *greedy* method to produce an existing placement.

We vary the number of most preferred clouds that users use to place masters, and we also vary the QoS and the data availability requirements. We have 10 clouds sorted for every user. Besides the 10-clouds case, we also compare the cases when each user uses her 2, 4, 6, and 8 most preferred clouds for master placement. We vary the QoS requirement by varying the lower bound while keeping the upper bound fixed at $\vec{Q}_u = (1)$. Note that, when there are 10 clouds a QoS vector should have 10 elements, but we omit the consecutive 1's at the end of a QoS vector for the ease of presentation. $\vec{Q}_u = (1)$ corresponds to the *greedy* placement and is the best QoS that can be provided. We vary $\vec{Q}_l$ by the following rule. Given the value of the first element (hereafter we call it the "first value" for brevity) of the $\vec{Q}_l$ vector and given the number of most preferred clouds that users use, we set the values of all other elements of $\vec{Q}_l$ by building a linear growth from its first value to 1. For example, if the first value of $\vec{Q}_l$ is 0.5 and users use 2 most preferred clouds, then $\vec{Q}_l = (0.5, 1)$. With the same first value, if users use 6 most preferred clouds, then $\vec{Q}_l = (0.5, 0.6, 0.7, 0.8, 0.9, 1)$. We vary the data availability requirement by iterating R from 0 to 9. We set $\beta = 1$, reflecting the fact that the cost of moving some data across clouds once is similar to that of storing the same data in the cloud for one month.

In the second group of evaluations, with the inputs of our 48 monthly OSN snapshots with real-world costs and the other 48 monthly snapshots with estimated costs, we focus on the *continuous* cost reduction that can be achieved by cosplay, compared with the *greedy* method. For each month, we run *greedy* on the former, representing the real-world common practice of placing user's data on the closest cloud for lowest access latency. We run cosplay on the former to show the "ideal" cost reduction, assuming we know the exact costs of each user for each month at the beginning of every month. We also run cosplay on the latter, where replica locations are adjusted according to the estimated costs of each user, to show the effectiveness of our estimation approach. Note that cosplay runs only *once* at the beginning of every month. When new users join the system during a month, each user is still placed by the *greedy* method. When only using *greedy*, the total cost for each month is the sum of the storage and the inter-cloud traffic cost, plus the maintenance cost. When running cosplay, the total cost for each month additionally includes the redistribution cost. We use the same $\vec{Q}_u$ and β settings as in our first group of evaluations and only consider the case where every user uses all 10 most preferred clouds. We set $R = 0$ and the first value of $\vec{Q}_l$ to be 0.5.

C. Evaluation Results

In the figures, the cost of every placement is normalized as the quotient of the placement divided by the standard cost, where the standard cost is the cost of the *greedy* placement with $R = 0$. The storage cost is normalized by the standard storage cost, the inter-cloud traffic cost and the redistribution cost is normalized by the standard inter-cloud traffic cost, and the total cost is normalized by the standard total cost.

1) One-time Cost Reduction: We note that, throughout Fig. 8 to 11 and in Fig. 14 and 15, the first value of $\vec{Q}_l$ is always set as 0.5, and we vary $\vec{Q}_l$ in Fig. 12 and 13.

Fig. 8 compares the costs of the placements produced by different methods over all the 10 clouds with $R = 0$. For all methods except cosplay, the total cost is the sum of its storage and inter-cloud traffic cost. For cosplay, the

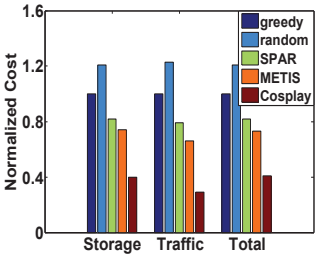


Fig. 8: Cost comparison (I)

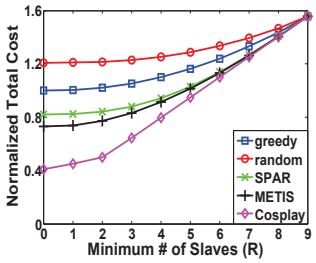


Fig. 9: Cost comparison (II)

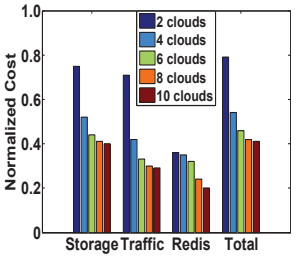


Fig. 10: Cost of cosplay (I)

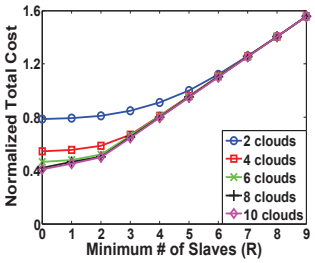


Fig. 11: Cost of cosplay (II)

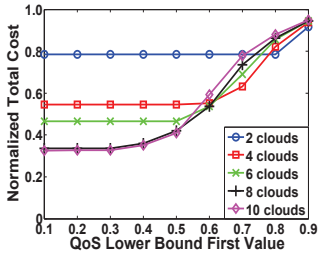


Fig. 12: Cost of cosplay (III)

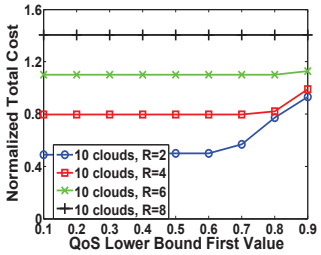


Fig. 13: Cost of cosplay (IV)

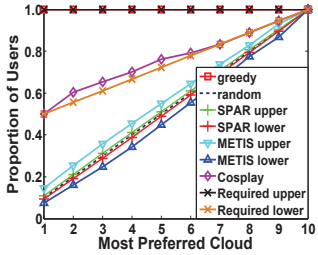


Fig. 14: QoS comparison

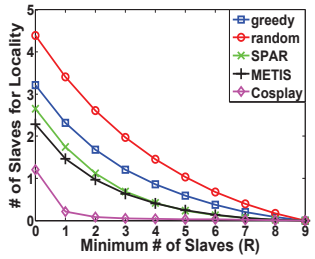


Fig. 15: Slave numbers

total cost additionally includes the redistribution cost. The *greedy* placement has moderate cost compared with *random*. Users who are geographically close to one another tend to have similar sorted lists of clouds. Thus, *greedy* can assign local users to the same nearby cloud and *random* tends to straddle local social relations across clouds. SPAR has less cost than *greedy* and *random* but more than METIS, indicating that minimizing the *number* of replicas cannot necessarily minimize the actual cost. Cosplay outperforms all others with total cost reductions of 59%, 66%, 50% and 44%, compared to *greedy*, *random*, SPAR and METIS, respectively.

Fig. 9 depicts the total cost of each method over 10 clouds as R, the minimum number of slave replicas required for every user to ensure data availability, varies. When R is small, cosplay achieves more cost reduction via role-swaps and eliminates those slaves that are no longer needed for social locality. When it becomes larger, cosplay's advantages are decreasing, as it cannot eliminate some slaves as they are needed for data availability even if they are not needed for social locality. The room for optimization also becomes less. All methods turn to full replication when $R = 9$.

Fig. 10 dissects the costs of the placements produced by cosplay on users' 2, 4, 6, 8, and 10 most preferred clouds in the case of $R = 0$. As the number of involved clouds grows, the storage cost, the inter-cloud traffic cost and the total cost drops, since more optimization can be done if more clouds are available for each user. However, the redistribution cost also declines, indicating we pay less overhead to achieve more saved costs. The reason is, as the total number of feasible role-swaps grows, the number of those with negative redistribution cost also increases, dragging down the total redistribution cost as more role-swaps put users' masters on clouds where they do not have a replica in the existing placement.

Fig. 11 demonstrates the total cost of each cosplay placement as R increases. No matter how R changes, we observe that when users consider placing their masters on more than 4 of their most preferred clouds, the advantages of cosplay are not obviously influenced by the number of most preferred clouds. In contrast, when every user only considers using her 2 most preferred clouds to host her master, cosplay achieves much less cost reduction. This phenomenon implies that 2 most preferred clouds of each user do not cover the social relations among users, while using 4 most preferred clouds of each user results in clouds of friends overlapping and masters of friends co-located, and thus fewer slaves are needed for social locality and the total cost can be significantly less than the 2-clouds case. However, this advantage is gradually compensated as the minimum number of slaves required by every user increases.

Fig. 12 shows the total costs cosplay achieves when the QoS requirement varies with $R = 0$. As the first value of $\vec{Q}_l$ increases, the number of users that are allowed to be role-swapped decreases and thus less room is left for optimization. When the first value is small, it does not affect the amount of cost that can be saved. Although more users are allowed to be role-swapped, the number of role-swaps that can lead to actual cost reduction is limited—allowing more users to be role-swapped does not necessarily indicate more cost reduction. When the first value becomes large enough, operating on fewer most preferred clouds achieves more cost reduction, which aligns with our intuition that placing together a small number of users instead of straddling them across clouds could save the cost. As cosplay operates on a larger number of most preferred clouds, it is easier for the cost reduction to be affected by the first value as this value grows. This is natural as operating on more clouds indicates that more role-swaps can be done to save cost. Hence, it is easier to be affected when the number of permitted users decreases.

Fig. 13 provides the total cost cosplay achieves as both the QoS requirement and R vary when operating on all the 10 most preferred clouds. The case with a larger R tends to be less affected by $\vec{Q}_l$ than the case with a smaller R. This is because when R is larger, the room for optimization becomes smaller because the number of role-swaps that leads to cost reduction becomes small. It can be small enough that even when $R = 8$ the QoS requirement does not have any effect on

the cost. With any given QoS requirement, a larger R always comes with a lower cost reduction, consistent with Fig. 11.

Fig. 14 visualizes the QoS vectors of the placements produced by all the methods over the 10 clouds. No doubt that *greedy* has the best QoS. *Random* has the linear QoS as expected. SPAR or METIS partitions a graph into a given number of partitions. With 10 clouds, there exist 10! = 3,628,800 different ways of placing the 10 partitions upon the 10 clouds. Each placement has its own QoS. Out of all 10! QoS vectors, we obtain the largest value and the smallest value for each of the 10 dimensions of the QoS vector. We can therefore draw the upper and lower QoS bounds for SPAR and METIS. SPAR and METIS are only able to produce QoS similar to *random*, while cosplay can always keep the QoS within any pre-defined upper and lower bounds.

Fig. 15 investigates how many more slaves we need to ensure the social locality for every user, except the minimum number of slaves that are maintained for data availability (some of them may also serve social locality). This figure draws the average number of additional slaves needed for social locality in the placements produced by different methods. We see that, while meeting the data availability requirement, cosplay always needs the fewest number of additional slaves for social locality of all users. Cosplay not only minimizes the cost, it also reduces the number of replicas. What is interesting is that, SPAR, an algorithm of minimizing the replica number, is beat by METIS. This is because SPAR runs as a procedure responding to a series of edge creation events and only guarantees the minimal number of replicas of involved users in a local sense, while METIS takes the whole social graph as input and thus achieves better results in a global sense.

2) Continuous Cost Reduction: We note that, as stated in Section V-A, we mainly focus on the time periods after the 16th month. The two peaks of user population growth in the 26th and 37th month are also reflected in our results.

Fig. 16 and 17 report the ratios of the maintenance cost and the redistribution cost over the total cost in each month, respectively. Fig. 16 verifies our cost model as the maintenance cost of *greedy* occupies less than 5% of the total cost, which is the reason why we can neglect the maintenance cost incurred by newly-joined users in our approximated cost model. Cosplay significantly reduces the total cost for each month, causing the maintenance cost to occupy larger proportions out of the total cost. Fig. 17 shows that the redistribution cost always keeps below 2% of the total cost of a month.

Fig. 18 depicts the one-time cost reduction for each month and the cumulative cost reduction until each month, compared with *greedy*. We observe that the one-month and the cumulative cost reductions achieved by running cosplay on estimated costs do not deviate much from, and almost overlap with reductions achieved by running cosplay on real-world costs. We check the prediction accuracy for all 48 months. On predicting the traffic cost of each user in each of the 48 months, in 4 months the prediction error averaged over all users is less than ± 1 time, and in 39 months it is less than ± 5 times. On predicting the storage cost of each user in each of the 48 months, in 35 months the prediction error averaged over all users is less than $\pm 1\%$, and in 45 months it is less than $\pm 2\%$. The small prediction error in the storage cost tends to be the cause of the small deviation of the cost reduction based on predicted cost. This is because as time elapses a user's stored data accumulates and becomes much bigger than her traffic amount, making the storage cost dominate the cost reduction. In this figure we also observe that the cost reduction climbs up as time elapses, and the accumulative total cost reduction goes towards more than 40%. The cost reduction can be deteriorated by large monthly growth rates, as in the months where user growth peaks occur. However, as discussed previously, the real-world monthly growth rate is usually quite small and thus we can expect significant cost reductions.

VI. Discussions

Having demonstrated the evaluation results of our algorithm, now we selectively discuss some related issues.

Complexity. Cosplay has a time complexity of $O(N^2\delta)$, where N is the number of clouds in the system and δ is the number of iterations executed, given that real-world OSN services often enforce a constant limit of the number of friends a user can have; without such a limit, in the worst case where every user was a friend of every other user, the complexity would be $O(|V|\delta)$ where $|V| \gg N$ is the number of users in the system. As an example, let's consider one iteration in the single role-swaps procedure, assuming each user has up to C friends. The first step of the random selection of a user is $O(1)$. In the second step, checking the feasibility of a single role-swap by Algorithm 1 takes $O(N)$ and thus checking all single role-swaps of a user takes $O(N^2)$. Then, calculating the cost reduction of a single role-swap by Algorithm 3 takes $O(C)$ and thus calculating the cost reductions of a user's all feasible role-swaps takes $O(NC)$. The third step picks up the role-swap with the largest cost reduction with a complexity of $O(N)$. The complexity of one iteration is thus $O(1) + O(N^2) + O(NC) + O(N) = O(N^2)$, and for δ iterations it is $O(N^2\delta)$. In our evaluations, we pre-specify that each user, on average, is allowed 100 times of being selected for role-swaps, *i.e.*, $\delta = 100|V|$; however, through all evaluations with various settings, our algorithm runs at most a few more than $10|V|$ iterations before no role-swap can be done to reduce the cost.

Optimality. Although cosplay only finds a local optimal solution to our cost optimization problem, it performs empirically much better than other placement approaches in Section V-C. Figuring out the optimality gap is challenging as finding the global optimal solution is NP-hard. Nevertheless, using the small-scale example in Fig. 1, we can have a rough sense about how much the optimality gap would be. We have 11 users and 3 clouds, so there are $3^{11} = 177147$ possible placements of masters; slaves are placed to ensure social locality and data availability. We consider the cases of $R = 0$, where slave replicas only serve the purpose of ensuring every user's social locality, and $R = 1$, where every user has at least 1 slave replica no matter it is for social locality or data availability. For a given QoS requirement and a given data availability requirement, we can obtain the placement with the

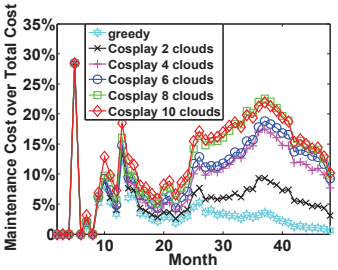


Fig. 16: Maintenance cost

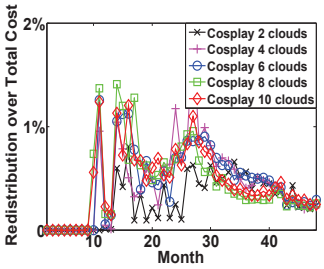


Fig. 17: Redistribution cost

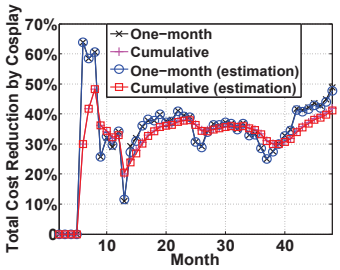


Fig. 18: Cost reduction

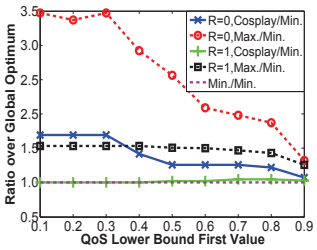


Fig. 19: Optimality gap

minimal (optimal) cost and the placement with the maximum (worst) cost by enumerating all the feasible solutions; we also run cosplay. Fig. 19 is the result. For example, when $R = 0$ and the QoS first value is 0.5, cosplay finds the solution whose cost is 1.26 times of the optimal cost while in this case the maximum cost is 2.56 times of the optimal cost; when $R = 1$, cosplay almost always finds an optimal solution. Compared with the former, there is less room for optimization in the latter case; one may imagine that in the extreme case of $R = 2$ no optimization can be done, as the total cost is fixed no matter how masters and slaves are placed. We deem that an approximation ratio like this is reasonably good.

Role-swap vs. master-migrations. It appears that role-swaps limit the solution to the initial placement, *i.e.*, a role swap does not seem to be able to migrate a user's data to a cloud that has none of her replicas in the initial placement. We will demonstrate and explain in the following that (1) allowing such master migrations does not help much in reducing the cost, and (2) only role-swaps *can* move a considerable amount of users to the clouds that do not host their replicas in the initial placement. Although not included in this paper, Algorithm 3 and 5 have already been adapted to allow master migrations. A master migration refers to moving a master replica from one cloud to another cloud which does *not* have a slave replica of the same user. As in role-swaps, social locality also needs to be ensured by creating slaves of neighbors if necessary. After adaption to master migrations, in each iteration, our algorithm can perform the operation, either a role-swap or a master migration, whose cost reduction is the maximal out of all feasible role-swaps and master migrations. We thus run additional evaluations as in Fig. 20 and 21. Fig. 20 indicates that even allowing master migrations, the number of master migrations performed only occupies a small portion of the total number of all the operations performed; in fact, the placement obtained has almost the same cost as in the only role-swaps case (which is not shown in the figure). The reason that role-swaps tend to suffice and master migrations may not be important is that a master migration barely reduces the total cost, unlike a role-swap: firstly, a master migration incurs redistribution cost itself by moving the master replica to the destination cloud, while a role-swap does not have this cost; secondly, it may also incur redistribution cost by creating slaves of neighbors at the destination cloud. In a role-swap of a user, as the user has a slave serving social locality at the destination cloud, it is highly likely that neighbors already have masters there, which does not hold for a master migration. Fig. 21 indicates that, when no master migrations are allowed and only role-swaps are performed, a considerable portion of users have ever had their masters swapped to clouds other than where their slaves are placed in the initial placement. For example, when the QoS first value is 0.1, for users who have 2 replicas (including master) in the initial placement, 45% of them have ever been beyond the initial clouds while cosplay runs. In this figure we set $R = 0$. As a user's slaves are created to maintain the social locality of a neighbor's master, the new slaves will cause the user's master to be swapped to the clouds where they are created. Therefore, a user's master is not restricted to those clouds hosting her slaves in the initial placement.

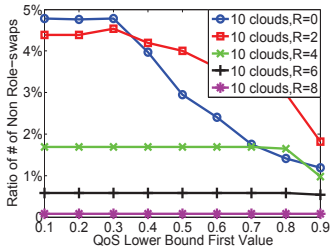


Fig. 20: Ratio of # of master migrations

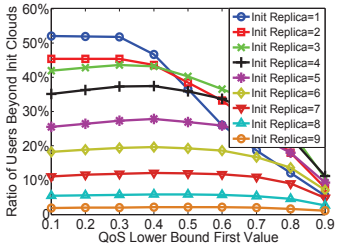


Fig. 21: Ratio of users beyond initial locations

Requirement variation. As the OSN evolves, the OSN provider's data availability requirement and QoS requirement may change. The change of the former can be easily handled. If a user needs more slaves to improve the data availability, one can choose some clouds and create the needed slaves there. Note that such slaves only accept propagated writes for consistency, as social locality is already ensured by existing slaves. The cost associated with these new slaves created for data availability does not rely on which clouds they are placed. On the other hand, if fewer slaves are expected as the current level of data availability is unnecessarily high, one can then remove slaves which do not serve the social locality. The QoS requirement can also be changed by the OSN provider so that an existing placement that satisfies the old requirement may not necessarily meet the new requirement; also, when users move their locations and their preferences for clouds change accordingly, the QoS of a placement may vary and not satisfy the QoS requirement any more. In this case, how do we minimize its cost while making such a placement satisfy the new QoS requirement and the data availability requirement? One approach can be to move data to make the placement meet the QoS requirement, which is always feasible, and afterwards running our algorithm as we do to minimize its cost. Here one may want to achieve this first step at the

minimum redistribution cost by composing an algorithm based on our adapted version of cosplay.

VII. Related Work

We contrast our work in this paper with existing work in the following three categories.

Optimizing OSN services. For OSN at a single site, using distributed hash to partition the data across servers [24], [6] potentially leads to poor performance. Recent work proposes maintaining social locality to address this issue: SPAR [29] minimizes the total number of slave replicas while maintaining social locality for every user; S-CLONE [33] maximizes the number of users whose social locality can be maintained, given a fixed number of replicas per user. For OSN across multiple sites, some propose selective replication of data across data centers to reduce the total inter-data-center traffic [25], and others propose a framework that captures and optimizes multiple dimensions of the OSN system objectives simultaneously [19]. The work in [29] and [33] does not have the concern of QoS as in our geo-distribution case. Besides, the cost models in all the aforementioned existing work, except [19], do not capture the monetary expense and cannot fit the cloud scenario, while [25] and [19] do not explore social locality to optimize the multi-data-center OSN service.

Graph (re)partitioning. The graph partitioning problem divides a weighted graph into a given number of partitions in order to minimize either the weights of edges that straddle partitions or the inter-partition communication volume while balancing the weights of vertices in each partition [9]. The repartitioning problem additionally considers the existing partitioning, minimizing the migration costs while balancing vertex weights [31]. State-of-the-art solutions for such problems include METIS [22] and Scotch [28]. Although similar in the sense of partitioning, the problem studied in this paper has fundamental difference from the classic graph (re)partitioning problems. Firstly, classic problems have no notion of social locality, QoS, and data availability, which makes these algorithms inapplicable to geo-distributed OSNs. Secondly, classic problems generally define a balance constraint, which is not necessary in the multi-cloud scenario because each cloud is supposed to provide "infinite" resources on demand.

Optimizing multi-cloud services. The work most related to OSN services may be those on social media [37], [38] that leverage online social relationships to improve media delivery. Volley [10] finds out the best data center for each data item based on access interdependencies, the identity and time stamp of data access, while balancing storage capacity across data centers; PNUTS [21] proposes selective replication at a per record granularity to minimize replication overhead and forwarding bandwidth, while respecting policy constraints. A substantial body of literature studies cloud resource pricing [30] and allocation [23], request mapping and content routing [39] in the multi-cloud or multi-data-center scenario. Although our work also focuses on multi-cloud services, OSN is unique in data access patterns (*i.e.*, social locality), making this group of existing work inapplicable to our scenario.

VIII. Conclusion

In this paper, we study the problem of optimizing the monetary cost spent on cloud resources when deploying an online social network (OSN) service over multiple geo-distributed clouds. We model the cost of OSN data placement, quantify the OSN quality of service (QoS) with our vector approach, and address OSN data availability by ensuring a minimum number of replicas for each user. Based on these models, we present the optimization problem of minimizing the total cost while ensuring the QoS and the data availability. We propose cosplay as our algorithm. By extensive evaluations with large-scale Twitter data, cosplay is verified to incur substantial cost reductions over existing, state-of-the-art approaches. It is also characterized by significant one-time and accumulated cost reductions over 48 months such that the QoS and the data availability always meets pre-defined requirements.

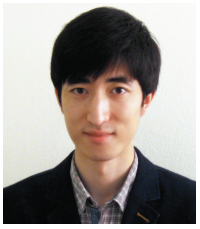
References

[1] Amazon EC2 Pricing. http://aws.amazon.com/ec2/pricing/.
[2] Case Studies. http://aws.amazon.com/solutions/case-studies/.
[3] Data Center Map. http://www.datacentermap.com/cloud.html.
[4] Pricing Overview: Microsoft Azure. http://www.windowsazure.com/en-us/pricing/details/.
[5] Facebook Newsroom. http://newsroom.fb.com.
[6] Twitter/Gizzard. http://github.com/twitter/gizzard.
[7] U.S. Board on Geographic Names (BGN). http://geonames.usgs.gov.
[8] *State of the Twittersphere*. HubSpot, Inc., Jan. 2010.
[9] A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *IPDPS*, 2006.
[10] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *NSDI*, 2010.
[11] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *SIGKDD*, 2006.
[12] J. Baker, C. Bond, J.C. Corbett, JJ Furman, A. Khorlin, J. Larson, J.M. Léon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *CIDR*, 2011.
[13] A.L. Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.
[14] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *IMC*, 2009.
[15] Fangfei Chen, Katherine Guo, John Lin, and Thomas La Porta. Intra-cloud lightning: Building cdns in the cloud. In *INFOCOM*, 2012.
[16] H. Chun, H. Kwak, Y.H. Eom, Y.Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: a case study of cyworld. In *IMC*, 2008.
[17] P.X. Gao, A.R. Curtis, B. Wong, and S. Keshav. It's not easy being green. In *SIGCOMM*, 2012.
[18] H. Hu and X. Wang. Evolution of a large online social network. *Physics Letters A*, 373(12-13):1105–1110, 2009.
[19] L. Jiao, J. Li, W. Du, and X. Fu. Multi-objective data placement for multi-cloud socially aware services. In *INFOCOM*, 2014.
[20] L. Jiao, J. Li, T. Xu, and X. Fu. Cost optimization for online social networks on geo-distributed clouds. In *ICNP*, 2012.
[21] S. Kadambi, J. Chen, B. Cooper, D. Lomax, R. Ramakrishnan, A. Silberstein, E. Tam, and H. Garcia-Molina. Where in the world is my data? In *VLDB*, 2011.
[22] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.
[23] Ali Khanafer, Murali Kodialam, and Krishna PN Puttaswamy. The constrained ski-rental problem and its application to online cloud cost optimization. In *INFOCOM*, 2013.
[24] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
[25] G. Liu, H. Shen, and H. Chandler. Selective data replication for online social networks with distributed datacenters. In *ICNP*, 2013.
[26] A. Mislove, M. Marcon, K.P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.

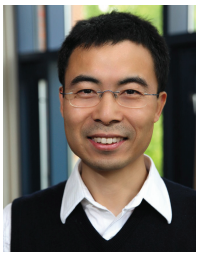
[27] A.E. Mislove. *Online Social Networks: Measurement, Analysis, and Applications to Distributed Information Systems*. PhD thesis, Rice University, 2009.
[28] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *HPCN Europe*, 1996.
[29] J.M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine(s) that could: Scaling online social networks. *IEEE/ACM Transactions on Networking*, 20(4):1162–1175, 2012.
[30] Heejun Roh, Cheoulhoon Jung, Wonjun Lee, and Ding-Zhu Du. Resource pricing game in geo-distributed clouds. In *INFOCOM*, 2013.
[31] K. Schloegel, G. Karypis, and V. Kumar. Wavefront diffusion and lmsr: Algorithms for dynamic repartitioning of adaptive meshes. *IEEE Transactions on Parallel and Distributed Systems*, 12(5):451–466, 2001.
[32] Y. Sovran, R. Power, M.K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, 2011.
[33] D.A. Tran, K. Nguyen, and C. Pham. S-clone: Socially-aware data replication for social networks. *Computer Networks*, 56(7):2001–2013, 2012.
[34] N. Tran, M.K. Aguilera, and M. Balakrishnan. Online migration for geo-distributed storage systems. In *USENIX ATC*, 2011.
[35] A. Vázquez, J.G. Oliveira, Z. Dezsö, K.I. Goh, I. Kondor, and A.L. Barabási. Modeling bursts and heavy tails in human dynamics. *Physical Review E*, 73(3):036127, 2006.
[36] B. Viswanath, A. Mislove, M. Cha, and K.P. Gummadi. On the evolution of user interaction in facebook. In *SIGCOMM WOSN*, 2009.
[37] Z. Wang, L. Sun, X. Chen, W. Zhu, J. Liu, M. Chen, and S. Yang. Propagation-based social-aware replication for social video contents. In *ACM Multimedia*, 2012.
[38] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F.C.M. Lau. Scaling social media applications into geo-distributed clouds. In *INFOCOM*, 2012.
[39] H. Xu and B. Li. Joint request mapping and response routing for geo-distributed cloud services. In *INFOCOM*, 2013.
[40] Y. Yang, Q. Chen, and W. Liu. The structural evolution of an online discussion network. *Physica A: Statistical Mechanics and its Applications*, 389(24):5871–5877, 2010.

APPENDIX
NP-HARDNESS PROOF

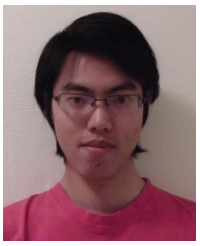
We prove the NP-hardness of our cost optimization problem formulated in Section III by restriction. Specifically, we show that the MIN_REPLICA problem, which has been proved NP-hard [29], is contained by our problem as a special case. Firstly, let $\beta = 0$ and let $\mu_u = 0, \tau_u = 1$, $\forall u \in [1, M]$. This makes the objective of our cost optimization problem the same as that of MIN_REPLICA, both minimizing the total number of slave replicas. Secondly, let the list of the sorted clouds of every user be identical, *i.e.*, $f_u(i,j) = f_v(i,j)$, $\forall u, v \in [1, M]$, $\forall i, j \in [1, N]$, and let $\vec{Q}_l$ and $\vec{Q}_u$ satisfy $\vec{Q}_l[k] = \vec{Q}_u[k] = \frac{k}{N}$, $\forall k \in [1, N]$. This makes Constraint (5) of our problem equivalent to the *load balance* constraint of MIN_REPLICA, both maintaining an equal number of master replicas across partitions. Thirdly, note that all other constraints of our problem are the same as their counterparts of MIN_REPLICA. Hence, MIN_REPLICA is a case of our problem under the above settings. Due to the NP-hardness of MIN_REPLICA, our cost optimization problem is NP-hard.■



Lei Jiao is a Ph.D. candidate in the Institute of Computer Science, University of Göttingen, Germany. Prior to Ph.D. study, he was a researcher at IBM Research - China. He received his M.Sc. and B.Sc. in Computer Science from Northwestern Polytechnical University, China in 2010 and 2007, respectively. His research interests span computer networks and distributed systems, with a recent focus on performance modeling, analysis, optimization and evaluation.

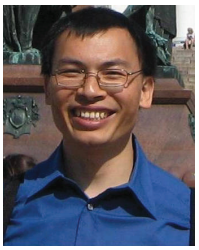


Dr. Jun Li is an associate professor in the Department of Computer and Information Science at the University of Oregon, USA, and directs the Network & Security Research Laboratory there. He received his Ph.D. from UCLA in 2002 (with honors), M.E. from Chinese Academy of Sciences in 1995 (with a Presidential Scholarship), and B.S. from Peking University in 1992, all in computer science. Specializing in computer networks, distributed systems, and their security, Dr. Jun Li is currently researching Internet monitoring and forensics, Internet architecture, social networking, cloud computing, and various network security topics. He has also served on several US National Science Foundation research panels and on more than 60 international technical program committees. He is currently an editor of Computer Networks and chair of several workshops and symposiums. Dr. Li is a 2007 recipient of the NSF CAREER award, a senior member of ACM, and a senior member of IEEE.



Tianyin Xu is a Ph.D. student in Department of Computer Science and Engineering, University of California San Diego. Before joining UCSD, he worked as a research staff in Institute of Computer Science, University of Göttingen, Germany. He received his B.Sc. and M.Eng. in Computer Science from Nanjing University, China in 2007 and 2010, respectively. His research interests lie broadly in computer systems and networks, with the goal of making today's systems more manageable, reliable, and available.

Dr. Wei Du is a postdoctoral researcher at the Research Unit in Networking, University of Liège, Belgium. He received his B.S. in 1997 from Tianjin University, China, and PhD in 2002 from Institute of Computing Technology, Chinese Academy of Sciences, China. Since graduation, he has been working as a postdoctoral researcher at INRIA, France, Hamburg University, Germany, University of Innsbruck, Austria, as well as University of Göttingen, Germany. His main research interests are computer networking and machine learning.



Dr. Xiaoming Fu received his PhD in computer science from Tsinghua University, China in 2000. He was 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. Dr. Fu's research interests include network architectures, protocols, and applications; in these fields he has published over 120 papers in journals and international conference proceedings. He serves on the editorial boards of IEEE Communications Magazine, IEEE Transactions on Network and Service Management, Elsevier Computer Networks, and Computer Communications, as well as program or organization committees of several main networking conferences such as ACM MOBICOM, MOBIHOC, IEEE INFOCOM, ICNP, ICDCS, IWQoS, CCW, and more recently as general co-chair of IEEE ICNP'13 and program co-chair of IEEE CloudNet'13.