Cost Optimization for Online Social Networks on Geo-Distributed Clouds

Lei Jiao
University of Göttingen
Göttingen, Germany
jiao@cs.uni-goettingen.de

Jun Li
University of Oregon
Eugene, OR, USA
lijun@cs.uoregon.edu

Tianyin Xu
UC San Diego
La Jolla, CA, USA
tixu@cs.ucsd.edu

Xiaoming Fu
University of Göttingen
Göttingen, Germany
fu@cs.uni-goettingen.de

***Abstract*—Geo-distributed IaaS (Infrastructure-as-a-Service) clouds provide an intriguing platform to deploy Online Social Network (OSN) services. To leverage the potential of clouds, a major task of OSN providers is optimizing the monetary cost spent on cloud resource utilization while providing satisfactory Quality of Service (QoS) to OSN users. We thus study the problem of cost optimization for the dynamic OSN on multiple geo-distributed clouds over consecutive time periods, with its QoS meeting the pre-defined requirement. We model the QoS as well as the cost of an OSN, formulate the problem, and design a solution named cosplay. Our experiments with a large-scale Twitter trace show that, while always ensuring the QoS as required, cosplay can achieve superior one-time cost reduction compared with the state of the art, and can also reduce the accumulative cost significantly when continuously evaluated over 48 months with dynamics comparable to real-world OSNs.**

I. Introduction

Recent years have witnessed the unprecedented popularity of Online Social Network (OSN) services. Instead of hosting all the OSN users in one central site, to provide better geographic proximity, data availability, and fault tolerance, large-scale OSN services are often deployed across multiple sites at diverse locations [1], [2]. This requirement matches seamlessly with geo-distributed clouds that provide Infrastructure-as-a-Service (IaaS), further with tremendous resource and cost efficiency advantages: "infinite" on-demand cloud resources can accommodate surges of user requests; flexible "pay-as-you-go" charging schemes can economize investments of service providers; and cloud infrastructures also free service providers from building and operating one's own data centers. Indeed, a number of OSN services are increasingly deployed on IaaS clouds, *e.g.*, Sonico, CozyCot and Lifeplat [3].

Distributing OSN data on multiple geo-distributed clouds must reconcile the needs from two different aspects. On one hand, OSN providers want to optimize the monetary cost spent in using cloud resources by a strategic data placement. 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 other users that are hosted at a different cloud. On the other hand, OSN providers hope to provide OSN users with satisfactory Quality of Service (QoS). For this purpose, they may want a user's data and those of her friends to be accessible from the cloud closest to the user, for example. Addressing both the cost and the QoS needs is further complicated by the fact that an OSN continuously experiences dynamics including the addition of new users, the departure of old users, and the variation of social relations.

In this paper, we set out to study the problem of optimizing the cost of the dynamic, multi-cloud-based OSN over consecutive time periods, while ensuring its QoS as required. We link the QoS of OSN service with users' data locations among clouds. All available clouds can be sorted for each user in terms of a certain quality metric (*e.g.*, access latency), and therefore every user has her first most preferred cloud, her second most preferred cloud, and so on. The QoS of the service is better if more users have their requested data hosted on clouds of more preference. We also model the monetary cost for cloud resource utilization over consecutive time periods. We identify different types of cost associated with OSN data placements upon clouds, while considering the real-world OSN dynamics as well as the social locality [4], [5], *i.e.*, the OSN data access pattern that a user's most activities occur between herself and her neighbors.

Having the QoS and the cost models, we formulate the cost optimization problem with a QoS requirement, and propose an algorithm named cosplay based on our observation that *swapping the roles* of a user's data replicas on different clouds can lead to possible cost reduction. We carry out extensive experiments based on distributing a geo-social dataset from Twitter with 321,505 users over 10 clouds all across US. Compared with many existing alternatives, including some straightforward solutions such as the *greedy* placement (the common practice of many online services [6], [1]), the *full* replication, the *random* placement (the *de facto* standard of data partitioning in distributed DBMS such as MySQL and Cassandra [7]), and some state-of-the-art algorithms such as SPAR [4] and METIS [8], cosplay can produce better data placements with the one-time cost reduced by up to 74% while ensuring the QoS as required. Further evaluations over 48 consecutive months with OSN dynamics comparable to real-world cases indicate that, by continuously applying cosplay, the accumulative invested cost of OSN provider can be reduced by more than 40%, compared with the *greedy* placement. Micro-benchmarks also demonstrate that our implementation

978-1-4673-2447-2/12/$31.00

of cosplay can compute the data placements with moderate memory and time overheads.

The remainder of this paper proceeds as follows. Section II describes our models of the QoS and the cost of OSN service. Section III presents the problem formulation. Section IV elaborates our cosplay algorithm. Section V demonstrates the evaluations. We contrast our work in this paper with existing work in literatures in Section VI, and we conclude this paper in Section VII.

II. Models

We model OSN QoS based on data locations among clouds, and demonstrate the usage of this model. We identify four types of cost, discuss the characteristics of real-world OSN dynamics, and establish our OSN cost model.

A. Modeling OSN Service Quality

Sorting clouds. We consider multiple disparate clouds distributed in different geographic locations. Among all clouds, one cloud can be better than another for a particular user in terms of a certain metric (*e.g.*, access latency, security risk, *etc.*). 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 M users, with cloud IDs from 1 to N and user IDs from 1 to M, clouds are sorted for user i as $\vec{c_i} = (c_{i1}, c_{i2}, ..., c_{iN}), \forall i \in [1, M]$, where $c_{ij} \in [1, N], \forall j \in [1, N]$. For any cloud c_{ih} and c_{ik} $(h < k)$, c_{ih} is more preferred than c_{ik}, *i.e.*, placing user i's requested data on the former provides better service quality for this user than placing the data on the latter. The clouds $\{c_{i1}, c_{i2}, ..., c_{ik}\}$ $(k \leq N)$ are therefore named the most preferred k clouds of user i, and the cloud c_{ik} is named the kth most preferred cloud of user i.

Defining QoS. We define QoS as $\vec{q} = (q_1, q_2, ..., q_N)$, where

$$q_k = \frac{1}{M} \sum_{j=1}^{k} \sum_{i=1}^{M} p_{jm_i}, \forall k \in [1, N]$$

m_i denotes the cloud that hosts the data requested by user i; p_{jm_i} equals to 1 if cloud m_i is the jth most preferred cloud of user i and equals to 0 otherwise. Therefore, q_k is the proportion of users whose data are placed on any of their respective most preferred k clouds. QoS is defined as a metric for the entire service, not for any individual user.

Comparing QoS. There can be different data placements upon clouds. Each may result in a different corresponding QoS. QoS' are compared to determine whether one placement outperforms another. For two QoS' $\vec{q_a}$ and $\vec{q_b}$, representing two placements respectively, the first is considered to be no better than the second, *i.e.*, $\vec{q_a} \leq \vec{q_b}$, if every element of the former is no larger than the corresponding element of the latter. If an OSN provider sets a QoS requirement $\vec{Q}$, we can seek the data placement(s) with the minimum cost and with the QoS no worse than $\vec{Q}$ according to this definition of QoS comparison.

Using this model. But how can an OSN provider express its QoS requirement using our model? We consider a latency requirement of "80% of access must be satisfied within 200 ms" as an example. In this case, for each user, the number of most preferred clouds that satisfy this requirement is calculated, *e.g.*, for any user i, it can be calculated that only by putting her requested data on any of her most preferred n_i $(1 \leq n_i \leq N)$ clouds can grant her the latency of lower than 200 ms.[1] This latency requirement can thus be expressed by setting $\vec{Q}[\min\{n_i | \forall i \in [1, M]\}] = 0.8$. Actually, the OSN provider can use our model to express any fine-grained requirement such as "95% of access must be satisfied within 500 ms, 80% be within 200 ms and 65% be within 90 ms".

B. Modeling OSN Cost

Taking social locality into consideration, we identify four types of cost for cloud resource utilization: the *storage cost* for storing users' data, the *inter-cloud traffic cost* for maintaining the consistency of data replicas across clouds, the *maintenance cost* for accommodating OSN dynamics, and the *redistribution cost* incurred by the cost optimization mechanism itself. We can therefore model the *total* cost of the data back-end of the dynamic OSN that is partitioned and placed in a multi-cloud environment over consecutive time periods.

*1) **Storage and 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 [9]. We extend the social graph model by associating three distinct quantities with every user: (1) Each user has her sorted list of clouds, as mentioned earlier; (2) Each user has her *storage cost*, which is the monetary cost for storing one replica of her data (*e.g.*, profile, statuses, *etc.*) in the cloud for one billing period; (3) Each user has her *traffic cost*, which is the monetary cost for this user's updates (*e.g.*, new statuses, *etc.*) as inter-cloud traffic to synchronize her data replica in one other cloud during one billing period.

When placing OSN data among multiple clouds, we maintain the social locality (*i.e.*, one-hop locality) by replicating the data of a user's every neighbor to the cloud that hosts the user's own data [4], [5], [10], which has been shown to be necessary for OSN services due to the OSN data access pattern that a user's most activities happen between herself and her neighbors (*e.g.*, all friends must be accessed to collect their recent tweets for a Twitter user). Maintaining social locality eliminates the inter-cloud multi-get traffic and the unpredictable response time because all the data requested by a user can be retrieved from a single cloud. Compared with the *full* replication, the storage cost is much lower because only neighbors need to be replicated across clouds.

The monetary cost for cloud resource utilization is usually charged per billing period, *e.g.*, on a per month basis [11], [12]. Given a billing period and a partitioning of the OSN data back-end, we consider the storage cost of all replicas and the inter-cloud traffic cost of the synchronization traffic from *master* replicas to *slave* replicas. We do not consider intra-cloud traffic because it is usually free of charge [11], [12].

[1] If a user cannot access any cloud in the system within 200 ms, we simply place the data requested by this user on her first most preferred cloud.

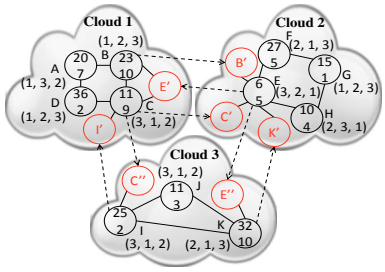


Fig. 1. Storage and inter-cloud traffic cost

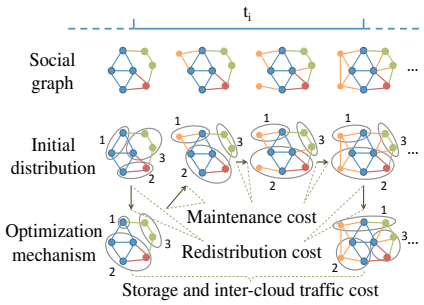


Fig. 2. Different types of cost

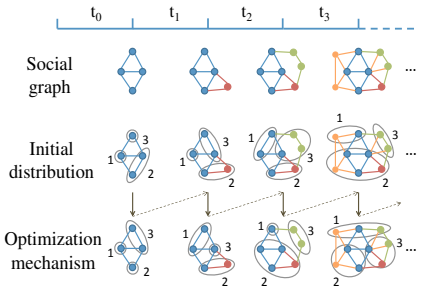


Fig. 3. Cost over consecutive time periods

Fig. 1 is a toy example demonstrating a partitioning of an OSN back-end. 11 users are hosted by 3 clouds. Black circles represent each user's only master replica, and red ones represent each user's slave replicas to ensure the social locality of neighbors' master replicas. Solid lines are social relations and dotted arrows are the synchronization traffic from a user's master to her slaves. The value in the first row within each circle is the storage cost of this user, and the value in the second row is the traffic cost. The vector associated with each circle is the sorted cloud IDs for this user. It can be calculated that the storage cost of this partitioning is 330 and the inter-cloud traffic cost is 50. Out of all the 11 users, 7 are hosted on their first most preferred cloud, 10 are on either of their most preferred two clouds, and all users are on any of their most preferred three clouds. Thus, the QoS is $\vec{q} = (0.64, 0.91, 1)$.

*2) **Maintenance Cost**:* Real-world OSNs are often characterized by *dynamics* such as new user arrivals, old user departures, and the variation of social relations. These events change the structure and the partitioning of OSN.

Let us look at the user joining case. When a new user joins the service, a cloud, *e.g.*, the one which has the lowest access latency for this user, is selected to place this user's data [1], [6]. [2] Some time later after this placement, social locality for this user and her neighbors must be maintained. To achieve this, new slave replicas are probably to be created on involved clouds. A creation is actually copying the data of a given user from the cloud that hosts her data to the cloud that does not have her data, and therefore incurs inter-cloud traffic cost. We name it as the *maintenance cost*, in order to differentiate from the inter-cloud traffic cost of synchronization as mentioned earlier. We assume an eventual maintenance for social locality, *i.e.*, necessary replicas are created at latest before the end of the current billing period. Fig. 2 illustrates the different types of cost of the OSN back-end during a single billing period. For simplicity, one-hop replication is not shown in this figure.

[2]Since no information about the new user's storage cost (except a certain reserved storage for every user, like storing a profile with pre-filled fields) and traffic cost for the current billing period is known, it is not possible to determine an optimum cloud in terms of cost for this user. The cloud to store this user's data can only be selected according to some pre-specified strategies. Aligning with the common practice of many real-world online services, in this paper, we assume that a new user's data are put on her first most preferred cloud (*e.g.*, the closest cloud in terms of access latency or geographic distance).

*3) **The Cost Model**:* Over a series of consecutive time slots, each of which is an independent billing period, the total cost is the sum of the cost of each time slot: (1) the storage and inter-cloud traffic cost, (2) the maintenance cost, and (3) the cost incurred by any cost optimization mechanism if applied.

One issue of this model is how frequently the cost optimization mechanism needs to be applied. We expect that such optimization is executed at a per billing period granularity for the following reasons. Firstly, this frequency is consistent with the billing period, usually the smallest 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 investment and expected profit, *etc.* Secondly, applying any cost optimization mechanism too frequently may fail the optimization itself. We generally envisage that an optimization mechanism to be devised optimizes the cost by moving data across clouds to optimum locations. This results in the *redistribution cost* (as shown in Fig. 2) which is essentially also the inter-cloud traffic cost. At the time of writing this paper, the real-world price of inter-cloud traffic transferring some data *once* is quite similar to that of storing the same amount of data for one month [11], [12]. As a result, moving data too frequently within one month can incur more cost which 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.

Consider a series of consecutive time slots $t_0, t_1, t_2, ...$ The social graph can vary within one time slot or at different time slots. We denote the final steady social graph within time slot t_i as $G_i = (V_i, E_i)$ $(i \geq 0)$. Let ΔG_i $(i \geq 1)$ be the difference between G_i and G_{i-1}. For t_i $(i \geq 1)$, let $\Psi_i(\cdot)$ denote the total cost, $\Phi_i(\cdot)$ denote the storage and inter-cloud traffic cost, $\Omega_i(\cdot)$ denote the maintenance cost, and $\Theta_i(\cdot)$ denote the redistribution cost, respectively. Then:

$$\begin{aligned}\Psi_i(G_i) &= \Phi_i(G_i) + \Omega_i(\Delta G_i) + \Theta_i(G_{i-1}) \\ &= \Phi_i(G_{i-1}) + \Phi_i(\Delta G_i) + \Omega_i(\Delta G_i) + \Theta_i(G_{i-1})\end{aligned}$$

For each time slot, the storage and inter-cloud traffic cost is incurred by the social graph of the previous time slot and by the social graph changes that occur in the current time slot. The maintenance cost is incurred by the social graph changes

that occur in the current time slot. The redistribution cost is incurred by the social graph of the previous time slot. Fig. 3 illustrates this cost model over consecutive time slots without showing one-hop replication.

We do not consider $\Phi_i(\Delta G_i)$ and $\Omega_i(\Delta G_i)$ because we expect $\Delta G_i \ll G_{i-1}$ when the user base reaches a certain scale so that $\Phi_i(\Delta G_i)$ and $\Omega_i(\Delta G_i)$ becomes negligible (as shown in Section V-C). Real-world OSNs are usually observed to have an S-shape growth [13], [14]. As the user population becomes larger, the increment of the total number of users or social relations is exponentially decaying [15], [16]. Let us look at some examples and focus on the *monthly growth rate* (*i.e.*, $|\Delta V_i|/|V_{i-1}|$). The user population of Facebook reached 58 million by the end of 2007. Afterwards, it grew with an average monthly rate below 13% through 2008 and 2009, with the rate below 6% through 2010, and then with the rate below 4% until the end of 2011 when it reached 845 million.[3] Twitter's average monthly growth rate was less than 8% in most months between March 2006 and September 2009 [18]. Similar rates have also been observed for YouTube and Flickr [19].

Therefore, the empirical cost model we focus on is:

$$\Psi_i(G_i) \approx \Phi_i(G_{i-1}) + \Theta_i(G_{i-1})$$

Note that calculating $\Psi_i(G_i)$ requires the storage cost and the traffic cost for t_i of each user in G_{i-1}. For any cost optimization mechanism that runs at the beginning of t_i, an estimation is required to predict each user's costs for t_i.

III. Problem

Having the QoS and the cost models, we are interested in the problem that, given an existing partitioning of social graph G of M users upon N clouds, finding out the optimum partitioning with the minimum cost, including the storage and inter-cloud traffic cost $\Phi(G)$ and the redistribution cost $\Theta(G)$ for implementing this optimum from the existing partitioning, while ensuring the QoS of the optimum to be no worse than the pre-defined QoS requirement. We assume that the storage and the traffic costs for each user can be estimated, and the list of sorted clouds for each user is known.

We introduce additional notations for the problem formulation. A partitioning is represented by binary variables r_{ij} and integer variables m_i. If either the master or a slave replica of user i is placed on cloud j, $r_{ij} = 1$; otherwise, $r_{ij} = 0$. m_i equals to the ID of the cloud that hosts the master replica of user i. r^e_{ij} and r^o_{ij} denote whether user i has a replica on cloud j in the existing partitioning and in the optimum partitioning, respectively; m^e_i and m^o_i denote the cloud hosting user i's master replica in these two partitionings. μ_i and τ_i are the estimated storage cost and traffic cost for user i. β is the coefficient converting the storage cost of a replica to the redistribution cost of moving this replica across clouds. p_{jm_i} is also binary: $p_{jm_i} = 1$ if cloud m_i is the jth most preferred cloud of user i; $p_{jm_i} = 0$ otherwise. $e_{ii'} \in E$ if user i and i' are neighbors. We formulate the problem as follows, where variables r^o_{ij} and m^o_i are the ones to be solved:

minimize:

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

where

$$\Phi(G) = \sum_{i=1}^{M} (\mu_i \times \sum_{j=1}^{N} r^o_{ij} + \tau_i \times (\sum_{j=1}^{N} r^o_{ij} - 1))$$

$$\Theta(G) = \sum_{i=1}^{M} (\beta \times \mu_i \times \sum_{j=1}^{N} (\max\{r^o_{ij} - r^e_{ij}, 0\}))$$

subject to:

$$r^e_{im^e_i} + r^o_{im^o_i} = 2, \forall i \in [1, M] \quad (1)$$

$$r^e_{i'm^e_i} + r^o_{i'm^o_i} = 2, \forall i, i' \in [1, M], e_{ii'} \in E \quad (2)$$

$$\frac{1}{M} \sum_{j=1}^{k} \sum_{i=1}^{M} p_{jm^o_i} \geq \vec{Q}[k], \forall k \in [1, N] \quad (3)$$

Constraint (1) ensures the existence of each user's master replica in the existing partitioning and in the optimum partitioning. Constraint (2) ensures the social locality for each user in these two partitionings. Constraint (3) ensures that the QoS of the optimum partitioning is no worse than the pre-defined QoS requirement. This problem can be proved to be NP-hard. We skip the formal proof in this paper due to space limitation.

IV. Algorithm

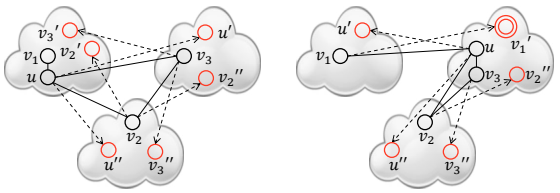
We seek heuristics that perform well in practice. We propose cosplay, a greedy optimization algorithm based on swapping the roles of replicas on different clouds to adjust the existing partitioning towards the optimum iteratively.

A. Basic Idea

We observe that swapping the roles of a user's master and her slave (*i.e.*, the master becomes a slave and the slave becomes the master) can lead to possible cost reduction. Fig. 4 is a simple example. We have a social graph with 4 users hosted by 3 clouds. Lines and circles have the same meanings as in Fig. 1. We swap the roles of replicas u and u' while keeping the social locality. Before swapping, there are 10 units of replica storage and 6 units of inter-cloud traffic; after swapping, 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 v_1 to create a new replica v'_1, marked by concentric circles). Overall, we can achieve 1 unit of cost reduction. Note that calculating cost reductions in practice must use each user's specific storage cost and traffic cost values, which usually vary for different users.

Inspired by this observation, we employ a series of role-swaps to maximize the total cost reduction which depends on two states: the existing partitioning before any role-swap is

[3]The rates presented here are calculated with the user population information announced by Facebook [17].



(a) Before swapping (b) After swapping

Fig. 4. Swapping the roles of u and u'

applied, and the final partitioning (*i.e.*, the output of cosplay) after all role-swaps are applied. A greedy approach of using role-swaps to achieve the final partitioning is ensuring that *every* applied role-swap has a positive cost reduction (while a negative cost reduction means the cost actually increases). Cosplay follows this approach and enables that summing up the cost reduction of each applied role-swap equals to the total cost reduction. The more cost reduction each role-swap has and the more role-swaps are applied, the more total cost reduction is achieved.

More importantly, it is quite likely that not every role-swap between a user's master and any of her slaves is *feasible*. We must ensure that each applied role-swap does not to make the overall QoS worse than the pre-defined QoS requirement.

Note that cosplay is an algorithm to compute a better partitioning (*i.e.*, data placement), and it does not physically move data across clouds. Once cosplay outputs its result, data can be moved to their corresponding new locations in order to implement the new partitioning.

B. Cosplay

We describe our cosplay algorithm as the following two phases. In practice, these two phases can be repeated one after the other until neither of them can be further executed.

Phase 1: Single role-swap. 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 positive cost reduction and apply it. Repeat this for a pre-specified number of iterations, or until some pre-specified cost reductions are achieved, or until no such single role-swap can be done.

Phase 2: Pair role-swap. In each iteration, select a user randomly, and pair this user and each of her neighbors whose masters are on a different cloud. For each of such pairs, if a pair of role-swaps, 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, are feasible, calculate the cost reduction of this pair. Then, choose the pair with the largest positive cost reduction and apply the two role-swaps. Repeat this for a pre-specified number of iterations, or until some pre-specified cost reductions are achieved, or until no such pair role-swap can be done.

The basic operations of cosplay, given a single role-swap or a pair role-swap, calculating its cost reduction and determining whether it is feasible, are nontrivial. The next two sections elaborate how to achieve these operations efficiently.

C. Calculating the Cost Reduction

Algorithm 1 specifies the calculation of the cost reduction of a single role-swap. Aligned with previous notations, u is the selected user; m_u is the cloud hosting u's master; s_u is the cloud hosting u's slave involved in this role-swap; μ_u and τ_u are u's storage cost and traffic cost; P_e is the existing partitioning before any role-swap is applied; $\Delta\mu$ and $\Delta\tau$ are the storage cost and the inter-cloud traffic cost that can be saved by this role-swap; ρ is the redistribution cost that is incurred by this role-swap. Based on Algorithm 1, Algorithm 2 calculates the cost reduction of a pair role-swap, where users u and v are neighbors.

Algorithm 1: ***calcCostReducSingle***(m_u, s_u)

```
begin
    Δμ ⇐ 0, Δτ ⇐ 0, ρ ⇐ 0;
    Non_m_u ⇐ ∅, Non_s_u ⇐ ∅;
    Rmv_m_u ⇐ true, Rmv_s_u ⇐ true;
    for each v ∈ u's neighbors do
        if m_v ≠ m_u then
            Non_m_u ⇐ Non_m_u ⋃ m_v;
            if u is v's only neighbor on m_u then
                Δμ ⇐ Δμ + μ_v, Δτ ⇐ Δτ + τ_v;
                if v has no replica on m_u in P_e then
                    ρ ⇐ ρ − β × μ_v;
            if m_v = s_u then
                Rmv_s_u ⇐ false;
        if m_v ≠ s_u then
            Non_s_u ⇐ Non_s_u ⋃ m_v;
            if u is v's only neighbor on s_u then
                Δμ ⇐ Δμ − μ_v, Δτ ⇐ Δτ − τ_v;
                if v has no replica on s_u in P_e then
                    ρ ⇐ ρ + β × μ_v;
            if m_v = m_u then
                Rmv_m_u ⇐ false;
    if Rmv_s_u = true then
        Δμ ⇐ Δμ − μ_u;
        if u has no replica on s_u in P_e then
            ρ ⇐ ρ + β × μ_u;
    if Rmv_m_u = true then
        Δμ ⇐ Δμ + μ_u;
        if u has no replica on m_u in P_e then
            ρ ⇐ ρ − β × μ_u;
    Δτ ⇐ Δτ + τ_u × (|Non_m_u| − |Non_s_u|);
    return Δμ + Δτ − ρ;
```

Algorithm 2: ***calcCostReducPair***(m_u, s_u, m_v, s_v)

```
begin
    ΔΨ_1 ⇐ calcCostReducSingle(m_u, s_u);
    swapRole(m_u, s_u);
    ΔΨ_2 ⇐ calcCostReducSingle(m_v, s_v);
    swapRole(s_u, m_u);
    return ΔΨ_1 + ΔΨ_2;
```

We highlight two of our insights with Algorithm 1. The first is, we recognize that accessing at most the two-hop neighborhood of a user is necessary and sufficient to calculate the storage and inter-cloud traffic cost that can be saved

by a role-swap of this user. An intuitive alternative can be calculating the difference between the total storage and inter-cloud traffic cost of the partitioning before applying this role-swap and that of the partitioning after applying this role-swap, which involves accessing every user and calculating the total cost of the partitioning twice and is therefore unnecessary and can cause considerable computation overhead given that we have a number of role-swaps on a large-scale social graph.

The second insight is the way we calculate the redistribution cost incurred by a role-swap. When removing a slave since it is no longer needed for social locality, we check whether this slave existed on its current cloud in the existing partitioning. If not, indicating that this slave was created by a previous role-swap and the incurred redistribution cost has already been counted, the cost of creating this slave is thus subtracted from the redistribution cost of the current role-swap. When creating a slave for social locality, we do the same the same check. If this slave did not exist on its current partition in the existing partitioning, the cost of creating this slave is added to the redistribution cost of the current role-swap.

D. Ensuring the QoS

Algorithm 3 determines the feasibility of a single role-swap by checking whether the current QoS would turn worse than the QoS requirement if this role-swap was applied. The selected user u's master and slave are on cloud c_{ui} and c_{uj} (*i.e.*, u's ith and jth most preferred cloud), respectively. $\vec{q}$ is the QoS of the current partitioning.

Algorithm 3: ***isSingleFeasible***(c_{ui}, c_{uj})

```
begin
    if i < j then
        for each k ∈ [i, j − 1] do
            if q⃗[k] − 1/M < Q⃗[k] then
                return false;
    return true;
```

Algorithm 4 determines the feasibility of two simultaneous role-swaps as in the pair role-swap case. 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 (if feasible) can change the current QoS, and the feasibility of the other role-swap must be determined based on this changed QoS. Algorithm 4 invokes Algorithm 5 to modify the QoS when a role-swap is applied.

E. Discussions

As a heuristic, cosplay produces partitionings with superior cost reductions over many existing approaches (*e.g.*, state-of-the-art heuristics such as SPAR and METIS) as shown in Section V-C. However, it is not yet clear to us how close these partitionings come to the optimum. Determining such the optimum is challenging because standard commercial optimization packages simply do not scale to the large size of real-world OSNs. This permits the possible existence of solutions beyond cosplay.

Algorithm 4: ***isPairFeasible***$(c_{ui}, c_{uj}, c_{vi}, c_{vj})$

```
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;
```

Algorithm 5: ***adjustQoS***(c_{ui}, c_{uj})

```
begin
    if i < j then
        for each k ∈ [i, j − 1] do
            q⃗[k] ⇐ q⃗[k] − 1/M;
    else
        for each k ∈ [j, i − 1] do
            q⃗[k] ⇐ q⃗[k] + 1/M;
```

The time complexity of cosplay is $O(M^2)$, where M is the total number of users. As an example, we consider Phase 1. The first step of random selection is $O(1)$. In the second step, checking the feasibility of one single role-swap by Algorithm 3 takes $O(N)$, where N is the total number of available clouds, and thus checking all single role-swaps of a user takes $O(N \times M)$. Then, calculating the cost reduction of one single role-swap by Algorithm 1 takes $O(M)$, and thus calculating the cost reductions of a user's all feasible role-swaps takes $O(M^2)$. The third step sorts all role-swaps with a complexity of $O(M \log M)$. With a constant pre-specified number of iterations for Phase 1, its complexity is $O(1)+O(N \times M)+O(M^2)+O(M \log M) = O(M^2)$, given that $N \ll M$; Similar analysis indicates that Phase 2 has the same complexity as Phase 1.

We also discuss and compare our current version of cosplay with other possible design options.

Why a random selection of users? One alternative can be that we always select and apply the role-swap with the globally maximum cost reduction. This approach is intriguing, but every time after a role-swap is applied, we have to update the cost reductions of affected users and then sort all role-swaps again. As mentioned earlier, the affected users are those within the two-hop neighborhood of a user. Besides, sorting all role-swaps after every role-swap causes large overheads. Actually, such a globally greedy version has a complexity of $O(M^3)$.

Why a combination of single and pair role-swap? Another alternative can be that we only run Phase 1 or Phase 2 solely. However, running both phases one after the other can help

each of them climb out of the local optimal, and can therefore achieve better results than running either of them alone. These two are the most basic forms of role-swaps. There may exist other more complicated forms of role-swaps (if not increasing the complexity), however, as shown in Section V-C, our current design of cosplay already achieves significantly good results.

V. Evaluations

A. Data Preparation

Collecting data. We crawled Twitter during March and April 2010 in a breadth-first-search manner, consistent with previous OSN crawls [9], [20]. We collected 3,117,553 users with 23,883,149 social relations. For each crawled user, we have his/her profile, tweets, and the followers list. 1,157,425 users provide location information in their profiles.

Sorting clouds for each user. We sort the clouds for each user in terms of access latency which is approximated by the real-world geographic distance between the user and the clouds [21], due to lack of the publicly available dataset on latencies between OSN users and OSN sites.

We focus on users geographically located in US. Firstly, we exploit the official database of US Board on Geographic Names [22] to filter out invalid locations (*e.g.*, "My home", "In the sea", *etc.*), and use Google Maps [23] to convert text locations to geo-coordinates (*i.e.*, [latitude, longitude]). Secondly, out of the result of the previous step, we extract the largest connected component as the input for our evaluations. Such a component we get has 321,505 users with 3,437,409 social relations. Thirdly, we select 10 cities all across US as cloud locations for our evaluations: 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 [24]. Finally, we sort all clouds for each user in terms of the respective geographic distance between the user and the clouds. Figure 5 plots the locations of all 321,505 users.

Extracting monthly OSN snapshots. To evaluate cosplay for dynamic OSN over consecutive time periods, we extract monthly graph snapshots out of our largest connected component, each representing the social graph of one month between March 2006 (when the earliest user joined) and February 2010 (when the latest user joined). 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 followers list of each user, we easily extract 48 monthly social graphs.

Obtaining and estimating costs for each user. Costs are calculated by multiplying the *unit price* with the data size. For all clouds, we use \$0.125/GB/month for storage cost and \$0.12/GB for inter-cloud traffic cost [11]. With each tweet and its time stamp, we calculate 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). Such calculated costs of each user for each of the 48 months are the ground truth in our evaluations. However, as stated in Section II-B, cosplay requires the estimation of each user's costs. We here exploit the *Exponential Weighted Moving Average* to iteratively estimate the number of a user's published posts during a month. We set the smoothing factor $\alpha = 0.9$ to capture the insight that users' online activities are bursty with heavy-tailed inter-activity times [25], [26]. Multiplying this number with the average size of all previous posts produces the traffic size of a user in a given month. The estimated storage size of a user in a month is calculated by adding the estimated traffic size to the storage size of the previous month. As a result, we also have 48 monthly graphs with estimated costs of each user for each month.

B. Experimental Settings

We run two groups of evaluations to compare the costs and the QoS'. We also micro-benchmark cosplay to show its own run-time performance.

In the first group, we compare the storage and inter-cloud traffic cost and the QoS of the partitionings produced by the *greedy* placement, the *full* replication, the *random* placement, SPAR, METIS and cosplay, respectively. The input is our largest connected component of February 2010. We ensure one-hop locality for all these approaches for fair comparison. The *greedy* placement places every user's master on her first most preferred cloud; the *full* replication maintains a replica for every user on every cloud; the *random* placement assigns a user's master to a cloud selected randomly. For SPAR, we have to implement its algorithm on our own, where we treat each social relation between two users as an edge creation event and create a random permutation of all events in order to produce a sorted edge creation trace as the input, following the method suggested in [4]. 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. Regarding cosplay, we use the *greedy* placement as the existing partitioning. Although there are 10 clouds sorted for every user, we also compare the cases when each user only selects her most preferred 2, 4, 6, and 8 clouds to place her master, respectively. For all 5 cases of different number of most preferred clouds, the first element of $\vec{Q}$ is always set to 0.5 and the last element is always 1, with a linear growth in between, *e.g.*, $\vec{Q} = (0.5, 1)$[4] when each user selects most preferred 2 clouds and $\vec{Q} = (0.5, 0.6, 0.7, 0.8, 0.9, 1)$ when each user selects most preferred 6 clouds. β is set to 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, we concentrate on the *continuous* cost reduction of cosplay. The inputs are our 48 monthly OSN snapshots with real-world costs and also the other 48 monthly snapshots with estimated costs. For each month, we run *greedy*

[4]By definition, $\vec{Q}$ has 10 elements when there are 10 clouds. We omit the consecutive 1's at the end of an QoS vector for ease of presentation.

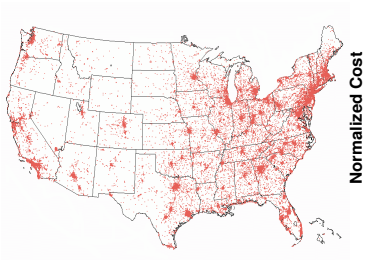


Fig. 5. User locations

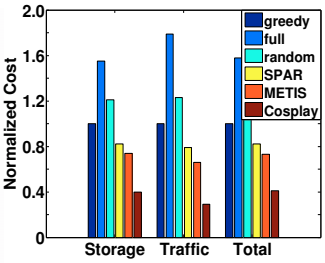


Fig. 6. Cost comparison

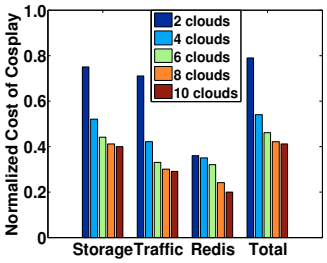


Fig. 7. Cost of Cosplay

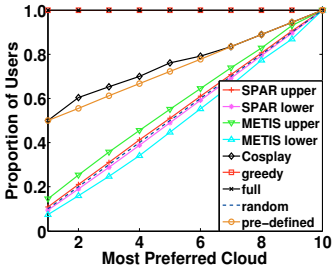


Fig. 8. QoS comparison

placement 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 for 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 for 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* placement. When only using *greedy*, the total cost for each month is the sum of the storage and inter-cloud traffic cost, plus the maintenance cost; When also running cosplay, the total cost for each month also counts the redistribution cost. We use the same $\vec{Q}$ and β settings as in our first group of evaluations and consider every user selecting her own most preferred 2, 4, 6, 8 and 10 clouds, respectively.

In the micro-benchmarks, we focus on how much RAM and time cosplay consumes in order to produce a partitioning. We implement cosplay by Java, and micro-benchmark it on a Dell PowerEdge 2950 server with 3 GHz Intel Xeon CPU and 16 GB RAM. The input is our largest graph snapshot with 321,505 users. Cosplay runs until neither of its two phases can be further executed.

C. Evaluation Results

1) ***One-time Cost Reduction****:* In this group, the cost of each partitioning method is *normalized* as the quotient of the cost of this method divided by the standard cost, where the standard cost is the cost of the *greedy* placement. 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; the total cost is normalized by the standard total cost.

Fig. 6 compares the costs of the partitionings produced by different methods on all the 10 clouds. For all methods except cosplay, the total cost is the sum of its storage and inter-cloud traffic cost; for cosplay, the total cost also accounts the redistribution cost. There is no doubt that *full* replication has the most 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 on the same nearby cloud and *random* tends to straddle local social relations across clouds. SPAR has the cost less 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 (the total cost reductions are 59%, 74%, 66%, 50% and 44%, compared with *greedy*, *full*, *random*, SPAR and METIS, respectively), indicating that the cost of the OSN back-end can be significantly reduced while meeting an appropriate pre-defined QoS requirement.

Fig. 7 demonstrates the costs of the partitionings produced by cosplay on users' most preferred 2, 4, 6, 8, and 10 clouds. As the number of involved clouds grows, the storage cost, the inter-cloud traffic cost and the total cost drops. This is because more optimization (*i.e.*, role-swaps) can be done if more clouds are available for each user. However, it is interesting to see that the redistribution cost also declines, indicating we pay less overhead to achieve more saved costs. The reason is, as the total number of role-swaps grows, the number of those with negative redistribution cost also increases, dragging down the total redistribution cost since more role-swaps put users' masters on clouds where they do not have any replica in the existing partitioning (see Algorithm 1).

Fig. 8 depicts the QoS' of the partitionings produced by all these methods on all the 10 clouds. No doubt that *greedy* and *full* replication 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', 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, respectively. As shown, SPAR and METIS are only able to produce QoS similar to *random*, while cosplay always keeps the QoS no worse than the pre-defined requirement.

2) ***Continuous Cost Reduction****:* In this group, we use our monthly OSN snapshots with real-world and estimated costs to evaluate cosplay for cost reductions over 48 months. Extracted from real-world Twitter data, our 48 snapshots with monthly growth rate around 15% since the 16th month (as shown in Fig. 9) are a good mimic of real-world OSN growth (as stated in Section II-B). We therefore mainly focus on the time periods *after* the 16th month. Note that there are two local peaks in the 26th and 37th month, which have also been reflected in our results.

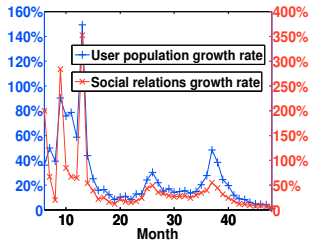


Fig. 9. Monthly growth of our dataset

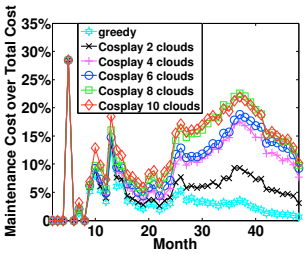


Fig. 10. Maintenance cost

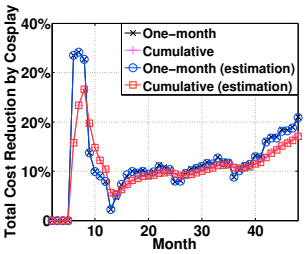


Fig. 11. Cost reduction: 2 clouds

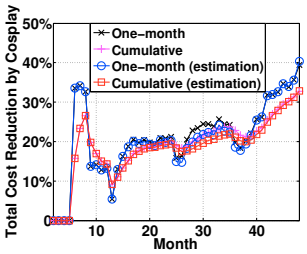


Fig. 12. Cost reduction: 4 clouds

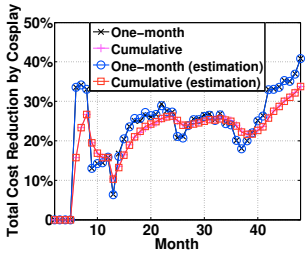


Fig. 13. Cost reduction: 6 clouds

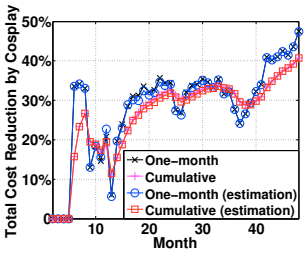


Fig. 14. Cost reduction: 8 clouds

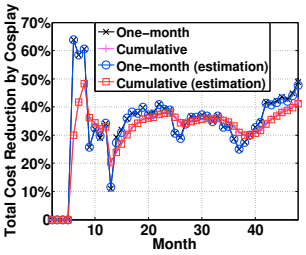


Fig. 15. Cost reduction: 10 clouds

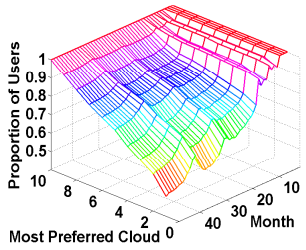


Fig. 16. QoS variation: 10 clouds

Fig. 10 verifies our cost model by showing that 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 empirical cost model. Cosplay significantly reduces the total cost for each month, causing the maintenance cost to occupy larger proportions out of the total cost.

From Fig. 11 to Fig. 15, we find the one-time cost reduction for each month and the cumulative cost reduction until each month, compared with *greedy*. We make several observations. Firstly, our estimation approach performs effectively. The one-month and the cumulative cost reductions achieved by running cosplay on estimated costs do not deviate too much from, and almost overlap with reductions achieved by running cosplay on real-world costs. Secondly, the cost reduction climbs up as time elapses, and increases as more clouds are involved for each user. In the 10-clouds case, the accumulative total cost reduction goes towards more than 40%. Thirdly, the cost reduction can be deteriorated by large monthly growth rates, as in the months where local peaks occur. However, as discussed in Section II-B, the real-world monthly growth rate is usually quite small and thus we can expect significant cost reductions. OSN provider can also determine whether to apply cost optimization to a billing period based on an estimation of the OSN growth rate. Finally, we report that, for all cases and through all months, the redistribution cost of a month always keeps below 2% of the total cost of that month.

Fig. 16 visualizes the QoS of the 10-clouds case as an example to show the QoS variation during the 48 months.

*3) **Micro-benchmarks**:* Table I and II show the RAM and time overheads of cosplay. Concerning the time consumption, *e.g.*, 0.7 minutes for 50% progress in the 2-clouds case means it takes 0.7 minutes for cosplay to achieve half of the total cost reduction that can be achieved in this case. With random selections of users, it becomes more difficult for cosplay to find role-swaps with positive cost reductions as more role-swaps are applied. One option can be that we do not wait until its completion and suspend it at a proper time when we have achieved "enough" cost reductions.

TABLE I
COSPLAY RAM CONSUMPTION

# of Most Preferred Clouds	2	4	6	8	10
RAM (GB)	2.0	2.1	2.1	2.2	2.2

TABLE II
COSPLAY TIME CONSUMPTION

# of Most Preferred Clouds		2	4	6	8	10
Time (Min)	50% Progress	0.7	1.7	2.5	3.1	3.4
	80% Progress	1.8	3.4	5.2	5.6	5.7
	90% Progress	2.4	4.5	7.0	7.2	7.3
	Completion	8.2	25.0	26.0	36.0	46.0

VI. RELATED WORK

OSN Back-end Scaling. Commercial OSN services usually adopt distributed hashing to partition the OSN data back-end among servers [7], [27], which can lead to poor performance (*e.g.*, response time). To address this problem, recent works propose replicating one-hop neighbors, *i.e.*, maintaining one-hop locality. SPAR [4] minimizes the total number of slave replicas while maintaining one-hop locality for every user and balancing the number of master replicas in each partition. S-CLONE [5] maximizes the number of users whose one-hop locality can be maintained, given a fixed number of replicas for each user. These works focus on optimization within a single site, and thus they do not have the QoS concern as in our geo-distribution case. Moreover, their cost models cannot fit the cloud scenario. We target at the storage cost and inter-cloud traffic cost while their goals are the number of replicas

or users. The redistribution cost, a first concern in our multi-cloud case, is out of their consideration because intra-data-center bandwidth is usually large and intra-cloud traffic is free of charge.

Graph (Re)Partitioning. The graph partitioning problem is dividing 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 while balancing the total weights of vertices in each partition [28]. The repartitioning problem additionally considers the existing partitioning, minimizing the migration costs while balancing the vertex weights [29]. State-of-the-art solutions for such problems include METIS [8] and Scotch [30]. 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 do not consider one-hop locality and QoS, which makes these algorithms inapplicable for 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.

Cross-site Data Placement. Volley [31] uses system logs of accesses to determine the data center for each data item based on access interdependencies, the identity and time stamp of data access, and the balance of storage capacity across data centers. PNUTS [2] proposes selective replication at a per record granularity to minimize inter-data-center replication and forwarding bandwidth, while respecting policy and latency constraints. Nomad [6] proposes storage overlay as an efficient migration mechanism to migrate e-mail data across data centers. Other related works also include placing social media files across clouds [32] and a substantial body of literatures studying data placements in Content Distribution Networks (CDNs). All such works are not for OSN and have respective models for their targeted services, neither do they capture OSN data access patterns or OSN dynamics.

VII. Conclusion

In this paper, we investigate the problem of optimizing the monetary cost spent on cloud resource utilization as deploying the OSN service among multiple geo-distributed IaaS clouds for consecutive time periods. We quantify the QoS by our novel vector-based approach, and model the monetary cost for the OSN data back-end, integrating social locality and exploring real-world characteristics of OSN dynamics. Based on these models, we present the cost optimization problem and propose cosplay as the solution. By extensive evaluations with large-scale Twitter data, cosplay is best verified by significant cost reductions over a 48-months period with the QoS guaranteed to be no worse than the pre-defined requirement.

References

[1] Y. Sovran, R. Power, M. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in *SOSP*, 2011.

[2] 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.

[3] "Case studies - amazon web services," http://aws.amazon.com/solutions/case-studies/.

[4] J. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The little engine(s) that could: Scaling online social networks," in *SIGCOMM*, 2010.

[5] D. Tran, K. Nguyen, and C. Pham, "S-clone: Socially-aware data replication for social networks," *Computer Networks*, vol. 56, no. 7, pp. 2001–2013, 2012.

[6] N. Tran, M. Aguilera, and M. Balakrishnan, "Online migration for geo-distributed storage systems," in *USENIX ATC*, 2011.

[7] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[8] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1999.

[9] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *IMC*, 2007.

[10] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing user behavior in online social networks," in *IMC*, 2009.

[11] "Pricing details: Windows azure," http://www.windowsazure.com/en-us/pricing/details/.

[12] "Amazon ec2 pricing," http://aws.amazon.com/ec2/pricing/.

[13] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: membership, growth, and evolution," in *SIGKDD*, 2006.

[14] H. Chun, H. Kwak, Y. Eom, Y. Ahn, S. Moon, and H. Jeong, "Comparison of online social relations in volume vs interaction: a case study of cyworld," in *IMC*, 2008.

[15] H. Hu and X. Wang, "Evolution of a large online social network," *Physics Letters A*, vol. 373, no. 12-13, pp. 1105–1110, 2009.

[16] Y. Yang, Q. Chen, and W. Liu, "The structural evolution of an online discussion network," *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 24, pp. 5871–5877, 2010.

[17] "Timeline - facebook newsroom," http://newsroom.fb.com/content/default.aspx?NewsAreaId=20.

[18] *State of the Twittersphere*. HubSpot, Inc., Jan. 2010.

[19] A. Mislove, "Online social networks: Measurement, analysis, and applications to distributed information systems," Ph.D. dissertation, Rice University, 2009.

[20] B. Viswanath, A. Mislove, M. Cha, and K. Gummadi, "On the evolution of user interaction in facebook," in *SIGCOMM WOSN*, 2009.

[21] P. Gao, A. Curtis, B. Wong, and S. Keshav, "It's not easy being green," in *SIGCOMM*, 2012.

[22] "U.s. board on geographic names (bgn)," http://geonames.usgs.gov/index.html.

[23] "Google maps api web services," http://code.google.com/apis/maps/documentation/webservices.

[24] "Cloud servers - data center map," http://www.datacentermap.com/cloud.html.

[25] A. Barabasi, "The origin of bursts and heavy tails in human dynamics," *Nature*, vol. 435, no. 7039, pp. 207–211, 2005.

[26] A. Vázquez, J. Oliveira, Z. Dezsö, K. Goh, I. Kondor, and A. Barabási, "Modeling bursts and heavy tails in human dynamics," *Physical Review E*, vol. 73, no. 3, p. 036127, 2006.

[27] "Twitter engineering: Introducing gizzard, a framework for creating distributed datastores," http://engineering.twitter.com/2010/04/introducing-gizzard-framework-for.html.

[28] A. Abou-Rjeili and G. Karypis, "Multilevel algorithms for partitioning power-law graphs," in *IPDPS*, 2006.

[29] 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*, vol. 12, no. 5, pp. 451–466, 2001.

[30] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," *High-Performance Computing and Networking*, pp. 493–498, 1996.

[31] 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.

[32] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. Lau, "Scaling social media applications into geo-distributed clouds," in *INFOCOM*, 2012.