Networking 2013 1569696579

Socially Aware Data Partitioning for Distributed Storage of Social Data

Duc A. Tran Ting Zhang
Department of Computer Science
University of Massachusetts - Boston
Email: {duc.tran@umb.edu, ting.zhang001@umb.edu}

***Abstract*—Online social networking has become ubiquitous. For a social storage system to keep pace with increasing amounts of user data and activities, a natural solution is to deploy more servers. An important design problem then is how to partition the data across the servers so that server efficiency and load balancing can both be maximized. Although data partitioning is well-studied in the literature of distributed data systems, social data storage presents a unique challenge because of the social locality in data access: we need to factor in not only how actively users read and write their own data but also how often socially connected users read the data of one another. We investigate the socially aware data partitioning problem by modeling it as a multi-objective optimization problem and exploring the applicability of evolutionary algorithms in order to achieve highly-efficient and well-balanced data partitions. Especially, we propose a solution framework that is closer to being optimal than existing techniques are, which is substantiated in our evaluation study.**

I. Introduction

To cope with rapid growth, a typical way to scale an online social network (OSN) is adding more servers to expand storage capacity and mitigate server bottleneck. Subsequently, a key problem is how to best partition the data across the servers. While this problem is well-addressed in the literature of distributed data systems [1]–[7], OSNs represent a novel class of data systems. In an OSN, where a read query for a user often requires fetching small data records of its neighbors in the social graph (e.g., friends' status messages in Facebook or connections' updates in LinkedIn), it is desirable to assign the data of socially-connected users to the same servers so that the number of servers required to process a query is kept small, which, in turn, should lead to faster response time [8]–[11]. Therefore, social locality is an important factor a partitioning design should take into account. This subject, however, was not brought up until just recently [11].

A great challenge for sharding social data across a number of servers while preserving social locality is due to the competing requirements on keeping the server load small (for good response time) while providing a good load balance (for bottleneck avoidance). The server load could be minimized by storing the entire data on a single server (so social locality is maximally preserved), but doing so incurs the worst load imbalance. On the other hand, excellent load balancing could be achieved if using DHT to assign user data on random servers, as in the partitioning implementation of today's most popular OSNs [12]–[14], but social locality would be broken due to the randomness nature of DHT. Because it is impossible to achieve both objectives, minimization of server load and maximization of load balancing, existing socially aware partitioning techniques [8], [9], [11] have to settle for a trade-off, emphasizing on one objective at the cost of the other.

Our motivating question is, while a trade-off is unavoidable, can we obtain the best trade-off or something that is close? To answer this question, we propose to model socially aware partitioning as a multi-objective optimization problem in which server efficiency and load balancing are optimized taking into account not only the social connectivity of the users but also their social activity, namely how often a user reads or write data of its own and how often connected users want to read one another's data. We then investigate the applicability of evolutionary algorithms (EA) to find a set of partition assignments that offer the best trade-offs between the two optimization objectives. Our contributions are below:

- Although EA is widely used to solve multi-objective optimization problems, our effort is the first to apply EA to the socially aware partitioning problem. This is not a trivial application of EA, though, due to the large size of the social graph. Indeed, we show that EA is not effective in converging to good partitioning solutions if EA starts with a typical population consisting of random candidate solutions (such as random partitions produced by DHT).
- To preserve social locality, previous socially aware solutions commonly rely on classic k-way graph partitioning algorithms to divide the social data graph into equally-sized components with minimum edge-cut [8], [9]. Although better than DHT, we show that this approach is far from being optimal. This is explainable because OSNs exhibit a strong community structure with power-law community size distribution, unseen in traditional random graphs for which graph partitioning algorithms are most suitable.
- We propose S-PUT, an effective EA-based solution framework that substantially outperforms both DHT- and graph-based approaches. Unlike conventional EA, the first generation in S-PUT is populated with a set of solutions to a classic graph partitioning problem that is transformed from the optimization problem. During the EA process, a final set of partition assignments can quickly be reached, offering excellent trade-off between

server efficiency and load balancing. This is substantiated in an evaluation study we have conducted with Facebook and Gowalla datasets.

The remainder of this paper is structured as follows. We discuss the related work in Section II. We define the socially aware partitioning problem formally in Section III. The proposed solution framework is described in Section IV. The results of our evaluation study are reported in Section V. The paper is concluded in Section VI with pointers to our future work.

II. Related Work

Horizontal scaling has been a de facto standard when it comes to managing data at massive scale for most OSNs. Instead of adding more hardware resources to the existing servers, the system, with horizontal scaling, is scaled "out" by adding commodity servers and partitioning the workload across these servers.

On top of a distributed infrastructure of commodity storage servers, popular OSNs today adopt a hashing-based mechanism for data partitioning, e.g., range-based hashing used in Gizzard [12] of Twitter or consistent hashing in Cassandra [13] of Facebook and Dynamo [14] of Amazon. The common drawback of these schemes is that hashing data to random servers does not preserve social locality. It has recently been shown [11] that network I/O can substantially be improved at the server side by keeping all of the relevant data of each query local to the same server. Even on a disk, these data should be stored closely together to improve disk response time [15].

Aimed to improve system performance and scalability, socially aware data partitioning and replication schemes have been proposed. SPAR [11] is aimed to preserve social locality perfectly, i.e., every two neighbor users must have their data colocated on the same servers. This is impossible if each user has only one copy of its data and so replicas are introduced and placed appropriately. Unlike SPAR which sets no limit on the maximal number of replicas for each user, S-CLONE [10], designed solely for replication, is aimed to preserve social locality as much as possible under a fixed space budget for replication. SCHISM [8] is a workload-driven scheme that partitions the data based on transaction patterns such as how often different data are retrieved together. While the queries targeted by SCHISM should be static and frequently repeated, OSNs often exhibit time-dependent queries (e.g., status messages of Facebook are frequently refreshed with more recent ones). This observation motivates the partitioning technique in [9] which takes as input an activity graph that is changing over time to better represent social locality.

The last two aforementioned, [8] and [9], are the only socially aware partitioning techniques that take into account the heterogeneity in how often a user reads/writes its own data and how often socially-connected users want to see the data of one another. They both rely on METIS, a classic graph partitioning algorithm highly effective for very large graphs, to partition the social data graph. We will show that the data partitions resulted from this approach are far from optimal. Much better partitions can be obtained by a simple yet effective evolutionary-based approach to be presented in this paper.

III. Problem Formulation

Consider a storage system with M servers to store data for a social graph of N users. We assume that each user's data is of small size and so it is desirable to minimize the number of servers required to access a given number of data records. We assume a three-tier system architecture, *User-Manager-Server*, in which the users do not communicate with the servers directly. Instead, the *Manager*, providing API and directory services, serves as the interface between the users (frond-end) and the servers (back-end). The API is used for the users to issue read and write requests. Assisted by the directory service, each request for a user is always directed to its primary server who is responsible to fulfill the request on the user's behalf. The directory service can be implemented either as a global map or as a DHT. The connectivity information of the social graph is assumed to be available at the Manager.

The system is characterized by the following parameters:

- Partition assignment P: a $N \times M$ binary matrix representing the primary assignment of user data across the servers. In this matrix, each entry p_{is} has value 1 if and only if user i is assigned to server s. The mapping from a server to a user is surjective, i.e., $\sum_{s=1}^{M} p_{is} = 1 \ \forall i \in [1, N]$.
- Write rate W: a N-dimensional vector representing user write request rates. Each element w_i is a positive real number quantifying the rate at which user i issues a write request. A write request for a user is always sent to its server.
- Read rate R: a N-dimensional vector representing user read request rates. Each element r_i is a positive real number quantifying the rate at which user i issues a read request. A read request for a user is always sent first to its server, requesting to retrieve its data and *possibly* the data of its neighbors. Whether a neighbor's data is also retrieved is determined by social bond strength, which is defined below.
- Social relationship E: a $N \times N$ real matrix representing the social relationships in the social graph. In this matrix, each entry e_{ij} is a value in the range [0, 1] quantifying the social bond between user i and user j. A stronger social bond indicates stronger probability (tendency) to read each other's data. The value 1 means the strongest and 0 means no relationship. It is noted that although i and j are socially connected, the values of e_{ij} and e_{ji} are not necessarily identical because the likelihood that user i wants to read its neighbor j's data may be different from the likelihood that user j wants to read user i's data.

We assume that the values for parameters W, R, and E can be obtained based on monitoring and analysis of actual workload. In practice, these values may vary over time and so we can divide the time into periods and update the values periodically. In the scope of this paper, we focus on one such period.

A. Server Load

Minimizing the server load and maximizing load balancing are among the most important objectives of any distributed storage system. The server load is categorized into three types: read load, write load, and storage load. We formulate these objectives below.

1) Read Load: To understand the server read load , consider a read request for user i. This request needs to be directed to its server, say server s, which will provide the data for i. Suppose that user i is also interested in the data of user j, one of its neighbors. To retrieve this data, there are two cases:

- User j's data is located on server s: The data of j can be provided by server s, requiring no additional server request.
- User j's data is not located on server s: The data of j needs to be retrieved from the server of j, requiring one additional read request sent to this server.

The amount of data returned to user i is the same in both cases, but the number of read requests that need to be processed at the server side is different, and, especially, worse if i and j do not colocate. More read requests result in more traffic and CPU processing at the server side. Therefore, an important objective is to minimize the server load due to read requests, which we refer to as *read load*, given the social relationships between the users and the rate at which they initiate read requests.

Given a server s, its read load is computed as

$$\begin{aligned}\lambda_s^{read} &= \sum_{i=1}^{N} r_i p_{is} + \sum_{i=1}^{N} r_i (1-p_{is}) \sum_{j=1}^{N} p_{js} e_{ij} \\ &= \sum_{i=1}^{N} r_i \left(p_{is} + (1-p_{is}) \sum_{j=1}^{N} p_{js} e_{ij} \right).\end{aligned}$$

This load consists of read requests that are initiated by (1) users i assigned to s and (2) users i not assigned to s that have neighbors j assigned to s. The number of read requests belonging to the latter group depends on the social strength matrix E which determines whether a neighbor's data needs also to be retrieved.

The total read load of all the servers is

$$\Lambda^{read} = \sum_{s=1}^{M} \lambda_s^{read} = \sum_{s=1}^{M} \sum_{i=1}^{N} r_i \left(p_{is} + (1-p_{is}) \sum_{j=1}^{N} p_{js} e_{ij} \right)$$

which can be re-written as

$$\Lambda^{read} = \sum_{i=1}^{N} r_i \left(1 + \sum_{j=1}^{N} e_{ij} \sum_{s=1}^{M} (1-p_{is}) p_{js} \right) \quad (1)$$

(derived using the equality $\sum_{s=1}^{M} p_{is} = 1 \ \forall i \in [1, N]$).

As visible in the above formulas, to minimize the total read load and/or balance the read load across the servers, we have to take into account how the data is partitioned (P) and the social relationships among the users (E).

2) Write Load: We refer to the number of write requests a server has to process as its *write load*, which depends on the number of users for whom the server stores data and the rate at which they initiate write requests. The write load of a server s is

$$\lambda_s^{write} = \sum_{i=1}^{N} w_i p_{is} \quad (2)$$

and the total write load of all the servers is

$$\Lambda^{write} = \sum_{s=1}^{M} \lambda_s^{write} = \sum_{s=1}^{M} \sum_{i=1}^{N} w_i p_{is} = \sum_{i=1}^{N} w_i. \quad (3)$$

While individual write loads depend on how the data are partitioned, the total write load is fixed regardlessly.

3) Storage Load: We compute the storage load of a server as the number of users whose data is stored at this server. The storage load of a server s is $\lambda_s^{store} = \sum_{i=1}^{N} p_{is}$ and the total storage load of all the servers is

$$\Lambda^{store} = \sum_{s=1}^{M} \lambda_s^{store} = \sum_{s=1}^{M} \sum_{i=1}^{N} p_{is} = N.$$

Similar to write load, the total storage load is fixed regardless of the data partition scheme, whereas individual storage loads are affected by how the data is partitioned.

B. Multi-Objective Optimization

Ideally, we want to simultaneously minimize the total read load, total write load, and total storage load of all the servers while balancing individual loads across the servers. These objectives, however, are conflicting with each other. For example, we could place all the data on the same server to minimize the read load, resulting in a read load of $\sum_{i=1}^{N} r_i$, but this would incur severe imbalance of the storage load and of the write load. Also, the write load and storage load cannot be balanced at the same time, nor can they be minimized, because of the write rate vector W. Thus, trade-offs are inevitable and we should be specific about which objectives are given more priority before we design the system.

Since OSNs exhibit high read/write rates, in this paper, we address the case where our priority is to minimize the total read load and balance write load (note that the total write load is a constant regardless of any partition). A solution to this case can easily be adapted to work for systems that want low read load and balanced storage load (instead of balanced write load) because the formulas for storage load are just a special case of the formulas for write load. Indeed, by setting $W = \{1, 1, ..., 1\}$, λ_s^{write} and Λ^{write} will exactly equal λ_s^{store} and Λ^{store}, respectively.

To represent the degree of load balancing across the servers, a variety of measures of statistical dispersion can be used, such as coefficient of variation, standard deviation, mean different, and Gini coefficient. Our framework does not enforce any specific measure. For the purpose of illustration, in this paper, we formulate the problem based on the Gini coefficient. Gini coefficient can be used to compare load balancing of different

distributions independent of their size, scale, and absolute values. This measure naturally captures the fairness of the load distribution, with a value of 0 expressing total equality and a value of 1 maximal inequality.

Assuming that servers are ranked in the increasing order of write load, the formula for the Gini coefficient is

$$\Gamma^{write} = \frac{2}{(M-1)\Lambda^{write}} \sum_{s=1}^{M} s\lambda_s^{write} - \frac{M+1}{M-1}.$$

Replacing λ_s^{write} and Λ^{write} with Eq. (2) and Eq. (3), respectively, we have

$$\Gamma^{write} = \frac{2}{(M-1)\sum_{i=1}^{N} w_i} \sum_{s=1}^{M} s \sum_{i=1}^{N} w_i p_{is} - \frac{M+1}{M-1}. \quad (4)$$

The range of Γ^{write} is $0 \leq \Gamma^{write} \leq 1$. To balance the server load, we need to minimize this Gini coefficient.

The optimization problem is expressed as follows:

Problem III.1 (Optimal Socially-Aware Partitioning)**.** *Find* P

$$\begin{aligned} \underset{P}{\textit{minimize}} \quad & \left[\Lambda^{read}, \Gamma^{write}\right]^T \\ \textit{subject to} \quad & 1) \ \sum_{s=1}^{M} p_{is} = 1 \ \forall 1 \leq i \leq N \\ & 2) \ \sum_{i=1}^{N} w_i p_{is} \leq \sum_{i=1}^{N} w_i p_{it} \ \forall 1 \leq s < t \leq M \end{aligned}$$

where

$$\Lambda^{read} = \sum_{i=1}^{N} r_i \left(1 + \sum_{j=1}^{N} e_{ij} \sum_{s=1}^{M} (1 - p_{is}) p_{js}\right)$$

and

$$\Gamma^{write} = \frac{2}{(M-1)\sum_{i=1}^{N} w_i} \sum_{s=1}^{M} s \sum_{i=1}^{N} w_i p_{is} - \frac{M+1}{M-1}.$$

IV. Proposed Framework

Problem III.1 belongs to the class of non-trivial multi-objective optimization problems for which we cannot identify a perfect solution simultaneously optimizing all the objectives. To problems of this kind, an alternative goal is to look for *Pareto-optimal* solutions. A Pareto-optimal solution is one that is not "dominated" by any other solution; solution A is dominated by solution B if A is no better than B in every objective and worse in at least one objective.

A popular approach to finding Pareto-optimal solutions is to use evolutionary algorithms (EA) [16]. EA, after many generations of crossover and mutation, can eventually result in a good approximation of these solutions. The drawback of EA, however, is due to its slow convergence to a stable solution state. In our case, the size of the solution space is N^M, with N unknown variables (the server assignment for each of the N users), the value for each variable anywhere between 1 and M. A social graph can contain millions of users, making the search space too large for a typical EA process to be effective.

Our proposed solution framework, which we hereafter refer to as S-PUT, relies on EA for its eventual guarantee toward Pareto-optimality, but applies it in a much more effective way. Specifically, S-PUT consists of two phases. In the *Initial Partitioning* phase, a graph partitioning algorithm is used to obtain the initial population for the EA process. In the *Final Partitioning* phase, the EA process takes place to result in the final set of optimized partition assignments.

A. Initial Partitioning

It is observed that if we denote $f_{ij} = r_i e_{ij} + r_j e_{ji}$, the total read load can be expressed as

$$\Lambda^{read} = \sum_{i=1}^{N} r_i + \frac{1}{2} \times \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} \sum_{s=1}^{M} (1 - p_{is}) p_{js}.$$

Thus, to minimize Λ^{read}, we need to minimize

$$\sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} \sum_{s=1}^{M} (1 - p_{is}) p_{js}.$$

This quantity is the sum of f_{ij} of pairs of socially-connected users, i and j, who are assigned to different servers.

Consider an undirected weighted graph G formed by the vertices and links of the original social graph, where each vertex i of G is associated with a weight w_i and each link (i, j) of G with a weight f_{ij}. Our optimization problem is equivalent to finding an optimal partitioning of graph G into M components such that:

- The edge cut, i.e., sum of the weights of inter-component links, is minimum (so that Λ^{read} is minimized), and
- The total vertex weight of each component is balanced (so that Γ^{write} is minimized).

This is a classic constrained weighted graph partitioning problem known to be NP-hard [17]–[20], but approximation algorithms have been proposed. Among them is METIS [18], arguably the best approximation algorithm for partitioning a large graph into equally-weighted components with minimum edge cut. According to its website (http://glaros.dtc.umn.edu/gkhome/views/metis), METIS can partition a 1-million-node graph in 256 parts in just a few seconds on today's PCs.

In our framework, our first step is to obtain a set of partition assignments, each being a result of applying METIS on graph G using a different "seed". Seed is a random factor used in METIS; a different assignment can be obtained by using a different value for the seed. These METIS partition assignments are used to populate the first generation of EA. The number of these assignments is equal to the "population size", an input parameter for the EA process, explained in the next section.

B. Final Partitioning

EA is an iterative process of generations, starting with an initial population of candidate solutions in the first generation and iteratively improving the population from one generation to the next, eventually reaching the final solutions in the last

generation's population. The evolution of the population is driven by three main mechanisms: recombination, mutation, and selection. Recombination and mutation create the necessary diversity, thus facilitating novelty in the population. After recombination and mutation, a selection step takes place to select the best quality individuals for the next generation's population. A fitness function is used to determine the quality of each individual.

Among many evolutionary algorithms, the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [21] and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [22] are de facto for solving multi-objective optimization problems. Although either can work in our framework for the evolution process, here we describe this process using SPEA2, which we have evaluated in our simulation study.

1) Representation of a Population: We represent a population as a set of individuals, each being a base-M string of length N, $s_1 s_2 ... s_N$, corresponding to a possible partition assignment: user i is assigned to server s_i. For example, for a network of 1000 nodes to be partitioned across 16 servers, an individual is an array of 1000 integers, each having a value between 0 and 15.

2) Evolution Process: SPEA2 maintains two types of population, $\mathcal{P}_h$ (called the "regular population", size $|\mathcal{P}|$) and $\mathcal{A}_h$ (called the "archive", size $|\mathcal{A}|$), for each generation h. These populations, $\mathcal{P}_h$ and $\mathcal{A}_h$, are updated as follows.

1) First generation ($h = 0$): $\mathcal{P}_0$ is the initial population of $|\mathcal{P}|$ individuals (result of applying METIS with $|\mathcal{P}|$ different seeds) and $\mathcal{A}_0$ is set to empty.
2) Generation $(h + 1)$:
   a) $\mathcal{A}_{h+1}$ = set of non-dominated individuals of $\mathcal{P}_h \cup \mathcal{A}_h$ (truncated to $|\mathcal{A}|$ individual if the population size is larger than $|\mathcal{A}|$, or padded with lowest-fitness individuals among the dominated if the population size is less than $|\mathcal{A}|$).
   b) $\mathcal{P}_{h+1}$ = set of $|\mathcal{P}|$ individuals after application of recombination and mutation on $\mathcal{A}_{h+1}$.

When the maximum number of generations, h^* (given as input), is reached, the final partition assignments will be the non-dominated individuals in the final archive $\mathcal{A}_{h^*}$.

3) Recombination Mechanism: In the recombination mechanism, a number of pairs of individuals, called parents, are randomly selected from the current population and each pair will be replaced by two new individuals, called offsprings. We use the Two-Point Recombination mode (the other modes provided by SPEA2 are One-Point Recombination and Uniform Recombination). Suppose that the two parents are $s_1 s_2 ... s_N$ and $s'_1 s'_2 ... s'_N$. First, two random positions in the string are chosen, i and j ($1 < i < j < N$). Then, the offsprings are $s_1 ... s_{i-1} s'_i ... s'_j s_{j+1} ... s_N$ and $s'_1 ... s'_{i-1} s_i ... s_j s'_{j+1} ... s'_N$. The number of parent couples to be replaced in the recombination mechanism is determined by a probability $p_{recombine}$ (given as input). Therefore, we should expect $|\mathcal{A}| \times p_{recombine}$ parents to be replaced by their offsprings.

4) Mutation Mechanism: In the mutation mechanism, a number of single parents are randomly selected from the current population, each to be replaced by an offspring. The offspring is created by setting a random value at each of a number of random bits of the parent string. For example, if only one position is to be assigned a random value, an offspring of a parent $s_1 s_2 ... s_N$ can be $s_1 .. s_{i-1} t s_{i+1} ... s_N$, where $i \in [1, N]$ and $t \in [0, M-1]$ are both chosen at random. There are two probability parameters for the mutation mechanisms: mutation probability p_{mutate} and gene mutation probability p_{gene_mutate}. The number of single parents to be replaced is $|\mathcal{A}| \times p_{mutate}$ and the number of random bits to be randomized in each parent is $N \times p_{gene\_mutate}$.

5) Dominated-ness and Fitness Function: An individual p dominates an individual q, denoted by $\mathsf{p} \succ \mathsf{q}$ if the corresponding partition assignment of p is no worse than that of q in terms of both Λ^{read} and Γ^{write}, with at least one objective *strictly* better. The fitness of an individual p in generation h is defined as

$$fitness(\mathsf{p}) = \sum_{\mathsf{q} \in \mathcal{P}_h \cup \mathcal{A}_h : \mathsf{q} \succ \mathsf{p}} strength(\mathsf{q}) + \frac{1}{\sigma_{\mathsf{p},k} + 2}$$

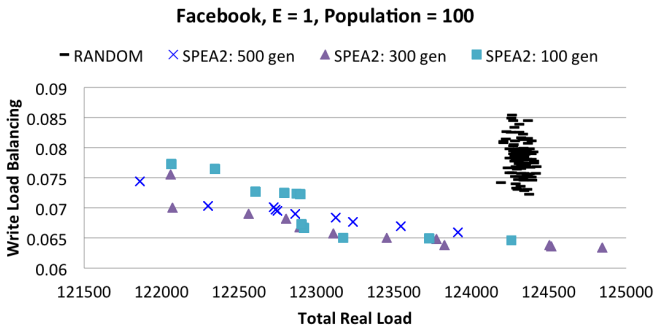
where $strength(\mathsf{q})$ is the number of individuals dominated by p in the set $\mathcal{P}_h \cup \mathcal{A}_h$ and $\sigma_{\mathsf{p},k}$ denotes the distance in the objective space from individual p to its k-nearest individual. As common setting, k is set to $\sqrt{|\mathcal{P}| + |\mathcal{A}|}$.

V. Evaluation Study

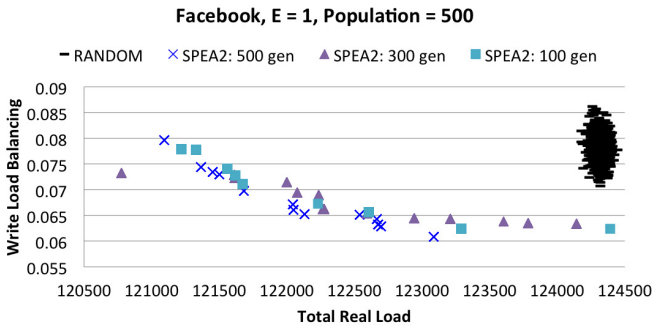
We present here the results of a simulation study to evaluate S-PUT in comparison with the conventional EA and graph-based approaches to our data partitioning problem. We used SPEA2 to represent the traditional EA approach and METIS the classic graph partitioning approach. To simulate the social graph, we have obtained two real-world samples: a Facebook graph obtained from a dataset made available by Max-Planck Software Institute for Software Systems and a Gowalla graph made available by the University of Cambridge. The Facebook graph contains $N = 63,392$ users in New Orleans region, with 816,886 links resulting in an average degree of 25.7. The Gowalla graph has $N = 196,591$ users and 950,327 links resulting in an average degree of 9.7. The latter graph has many more nodes but sparser connectivity.

We assume that the read and write rates of a user are proportional to its social degree, thus these rates are given values in the range (0, 1) proportional to the degree. We consider two models for the social strength between neighboring nodes: the *constant* model where every relationship has identical strength 1 and the *random* model where the strength is uniformly generated in the range (0, 1).

The number of servers is set to $M = 16$. The parameters for SPEA2 are: regular population size $|\mathcal{P}| \in \{100, 500\}$, archive size $|\mathcal{A}| = |\mathcal{P}|$, recombination probability $p_{recombine} = 0.8$, mutation probability $p_{mutate} = 0.5$, gene mutation probability $p_{gene\_mutate} = 0.001$, and number of generations $h^* \in \{100, 300, 500\}$.

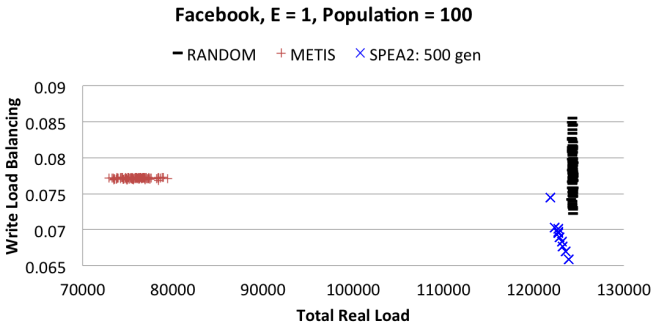


(a) Population: 100 individuals

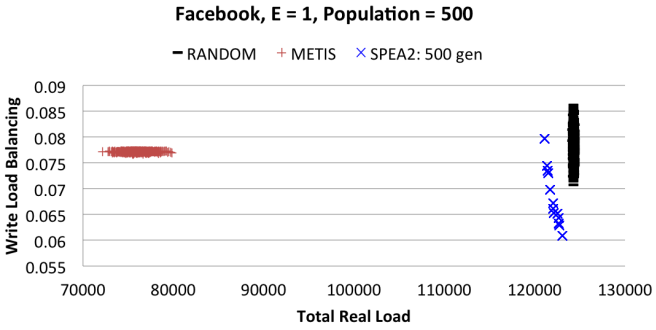


(b) Population: 500 individuals

Fig. 1. SPEA2 after 100, 300, and 500 generations: Insignificant improvement in terms of total read load, some improvement in terms of load balancing which, however, is not needed



(a) Population: 100 individuals



(b) Population: 500 individuals

Fig. 2. SPEA2 vs. METIS: While METIS offers excellent load balancing, comparable to RANDOM and SPEA2, it is superior in terms of total read load

A. *Effectivess of SPEA2*

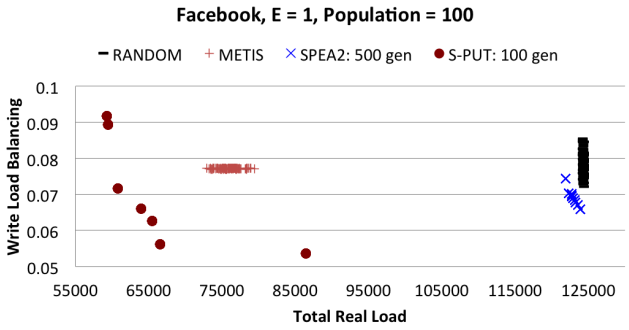
First, we evaluate the effectiveness of SPEA2, representing the evolutionary approach to solving the proposed partitioning problem. In this simulation, SPEA2 starts in the first generation consisting of random partition assignments (referred to as RANDOM assignments).

Figure 1(a) shows the initial population of 100 individuals (RANDOM) and the final population of non-dominated individuals after 100 generations, 300 generations, and 500 generations. The random assignments have a total read load consistently around 124,500 and a load balancing coefficient around 0.08. Gini coefficient below 0.1 implies excellent load balancing. After 100 generations of applying SPEA2, we observe some improvement. If we look at the final solution of SPEA2 with best load balancing, the reduction compared to the initial population is 19% ($0.065/0.08 \approx 81\%$). Although 19% is a good percentage number, this is an improvement over a load distribution that is already excellent. What we wish to see is a more significant improvement on the total read load. However, if we look at the final solution with best read load, the reduction compared to the initial population is only a tiny 2% ($122000/124500 \approx 98\%$).

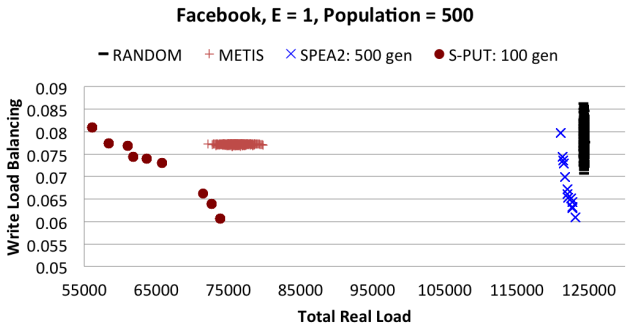
Similar observations are observed in the case the population size is 500 individuals, which are shown in Figure 1(b). In either case, 100 individuals or 500 individuals, the improvement does not seem to get much better after 300 generations and 500 generations, suggesting that we have seen the practical best of SPEA2 (unless we have to run many more generations, which is prohibitively long).

We have shown that SPEA2 is limited in its effectiveness. On the other hand, evolutionary algorithms are typically known as an effective way to result in Pareto-optimal solutions for multi-objective optimization problems. Therefore, the next question in our study is to see if indeed we cannot do better than SPEA2. To answer this question, we compare the result of SPEA2 after 500 generations to the METIS method that finds partitioning assignments as described in Section IV-A. This comparison is illustrated in Figure 2, which shows a clear contrast between these two methods. While METIS results in excellent load balancing, comparable to SPEA2 and RANDOM, METIS is superior in terms of total read load. In the case the population size is 100 individuals, an average METIS partitioning assignment offers a total read load that is an 40% improvement over RANDOM (compared to just 2% improved by SPEA2). A similarly significant improvement is also observed if the population size is 500 individuals. It is noted that METIS is faster to converge than SPEA2.

What stood out in this study is that (1) a typical use of an evolutionary algorithm, SPEA2 in our experiment, is not effective in finding a good partition assignment, and (2) it

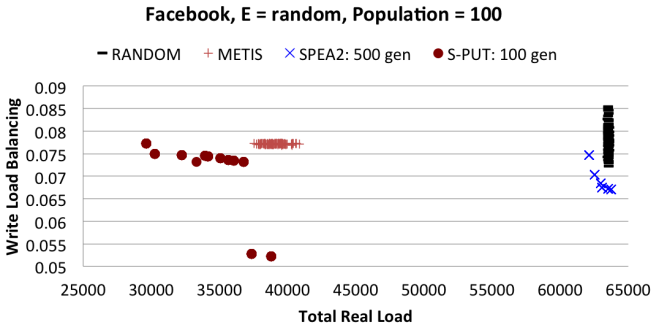


(a) Population: 100 individuals

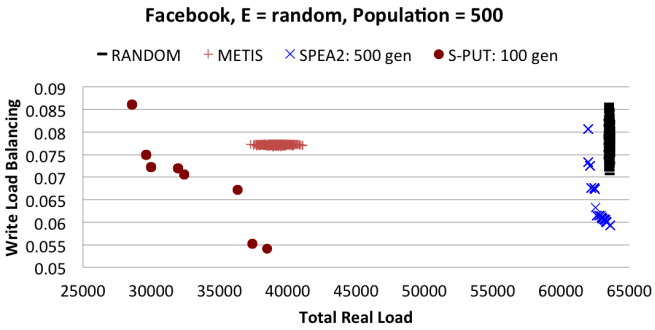


(b) Population: 500 individuals

Fig. 3. S-PUT: a highly effective EA process with final partition assignments substantially better than METIS and SPEA2



(a) Random strengths, 100 individuals



(b) Random strengths, 500 individuals

Fig. 4. S-PUT results: Facebook graph with randomly generated social bond strengths

is possible to have a partitioning technique that runs faster and offers substantially better optimization quality than SPEA2 does.

B. Effectiveness of S-PUT

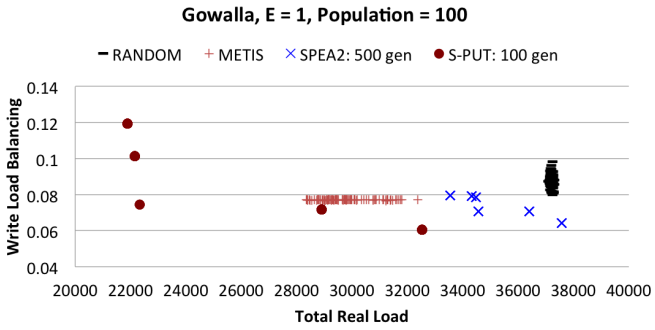
Here, we discuss the results of running our proposed technique, S-PUT. These results are illustrated in Figure 3. There are two noteworthy observations from this illustration. Firstly, the EA process in S-PUT is highly effective. After 100 generations, the improvement of S-PUT over its initial population (METIS) in terms of total read load is as high as 20% ($60,000/75,000 \approx 80\%$) for the case with 100 individuals, and 26% ($55,000/75,000 \approx 74\%$) for the case with 500 individuals. The typical SPEA2 process improves only about 2% over its initial population. The improvement in terms of load balancing in S-PUT is no worse than that in SPEA2. Secondly, the quality of S-PUT partition assignments is remarkable compared to METIS and SPEA2. In the case with 100 individuals, if we use the quality of RANDOM as benchmark, on average, S-PUT offers a total read load that is $65,000/124,500 \approx 52\%$ of RANDOM, while the total read loads for METIS and SPEA2 are 62% and 98%, respectively. These percentage numbers are similarly observed in the case with 500 individuals. Note that in this study, S-PUT stops after 100 generations whereas SPEA2 stops after 500 generations.

C. Effect of Input Social Graph and Social Bond Strengths

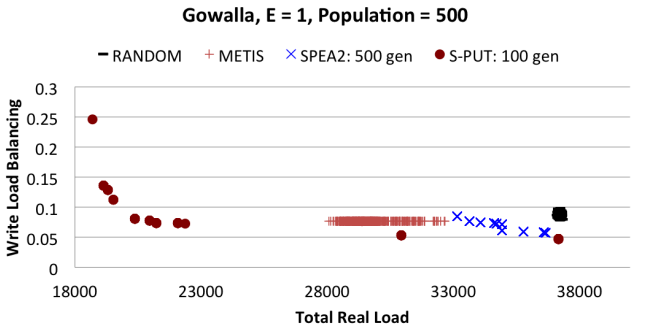
The results we have discussed above are with the Facebook graph in which the social bond strength between adjacent nodes is identical ($E = 1$). In this section, we will see if S-PUT remains superior when the social bond strength is randomly generated or when the social graph is different (Gowalla graph).

Figure 4 shows the S-PUT results for the Facebook graph in which the social bond strength is random. There is still a clear contrast between the group of S-PUT/METIS results and the group of SPEA2/RANDOM results, the former obviously offering better partition assignments than the latter. All of these methods offer excellent load balancing (Gini coefficient is less than 0.1), but S-PUT is no question the clear winner in terms of total read load. As an EA process, S-PUT is faster than SPEA2 to reach a good Pareto "front" (i.e., set of non-dominated solutions). After 100 generation, S-PUT can improve the total read load over its initial population by as much as 25%, whereas SPEA2 after 500 generations can improve over its initial population by only 2%, which is insignificant.

The case with Gowalla graph offers a slightly different picture. As seen in Figure 5 and Figure 6, the EA process in both SPEA2 and S-PUT is more effective than the case with Facebook. In other words, with the Gowalla graph, it is quicker for both SPEA2 and S-PUT to improve over their initial

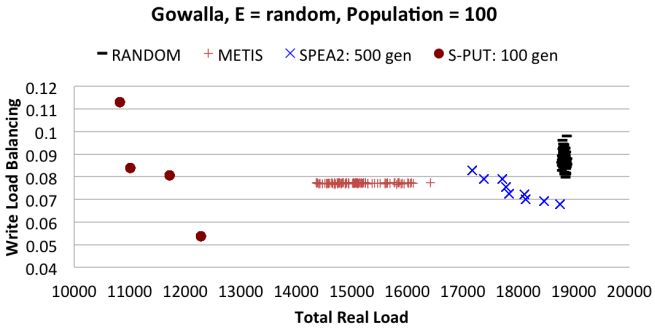


(a) Identical strengths, 100 individuals

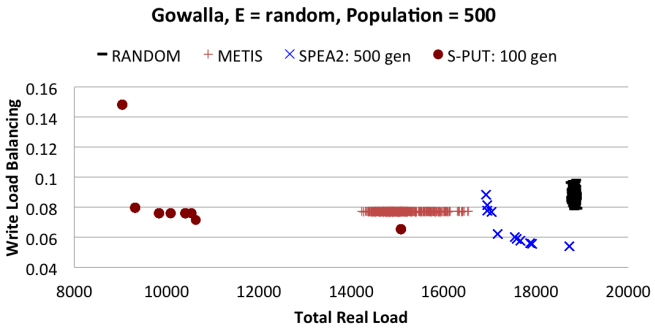


(b) Identical strengths, 500 individuals

Fig. 5. S-PUT results: Gowalla graph with identical social bond strengths



(a) Random strengths, 100 individuals



(b) Random strengths, 500 individuals

Fig. 6. S-PUT results: Gowalla graph with random social bond strengths

population. This helps SPEA2 get closer to METIS, albeit still inferior. Nevertheless, S-PUT clearly remains the best, standing out from the other methods. Consider the case with 100 individuals as population size. For the same load balancing (Gini coefficient narrowly around 0.8), looking at the solution with the best total read load for the case of identical social strength bonds, S-PUT's load is 76% of METIS, 65% of SPEA2, and 58% of RANDOM (Figure 5(a)). For the case of random social strength bonds, the total read load of S-PUT is 80% of METIS, 66% of SPEA2, and 60% of RANDOM (Figure 6(a)). These percentage numbers only change slightly for the case with 500 individuals as population size.

Throughout all the simulation runs with various configurations, S-PUT consistently outperforms the other competing methods by significant margins. Not only that it is faster than the typical EA process to improve over the first generation but it incurs significantly less total read load while maintaining comparable if not better load balancing.

D. Run time

Since S-PUT is an EA-based framework, the computation time is a trade-off for the quality of the final partition assignments. Table I presents the time for the EA process in S-PUT to complete. Here, we show the time information for the case $E = 1$, but for the case E is random the completion time should be similar because changes in the values of E do not affect the run time. For both graphs, as expected, the EA completion time is linearly proportional to the population size and the number of generations. For example, with 100 individuals, it took 30 minutes to run 100 generations on the Facebook graph and 50 minutes on the Gowalla graph. This time is roughly tripled with 300 generations and quintupled if we continue to 500 generations or have a population size of 500 individuals. In our discussion of the simulation results in prior sections, S-PUT with population size of 100 individuals already outperforms the competing methods if it runs EA for only 100 generations, which corresponds to 30 minutes of running on Facebook and 50 minutes on Gowalla. This is encouraging in terms of time complexity. Note that our simulation runs on a moderate Linux workstation with 8GB memory and 2.66GHz dual-core Intel Xeon CPU 3070.

VI. Conclusions

Social locality is a property that should be preserved in the data storage of any OSNs in order to improve the server efficiency. We have modeled the socially aware partitioning problem as a multi-objective optimization problem aimed at minimizing the total read load and balancing the write load, taking into account the user read/write activity and social relationship. As these are two competing objectives, a natural approach is to employ an evolutionary algorithm (EA) to find a Pareto-optimal solution. However, we have shown that a typical application of EA does not work acceptably. We have also shown that METIS, which is the basis for today's

TABLE I
COMPLETION TIME FOR THE EA PROCESS IN S-PUT FOR FACEBOOK AND GOWALLA GRAPHS: SIMULATION RUNS ON A LINUX WORKSTATION WITH 8GB MEMORY AND 2.66GHZ DUAL-CORE INTEL XEON CPU 3070

	Facebook: 63,392 users, 816,886 links			Gowalla: 196,591 users, 950,327 links		
Population/Num. Gen.	100 gen	300 gen.	500 gen.	100 gen	300 gen.	500 gen.
100 individuals	30m	1h27m	2h30m	50m	2h25m	4h
500 individuals	2h23m	7h22m	12h10m	4h	12h	20h10m

partitioning techniques for OSNs, is far from being optimal. Our proposed framework, S-PUT, presents a different way of applying EA, simple yet highly effective. S-PUT is superior to METIS in both total read load and load balancing. Although run-time is a trade-off for any EA process, S-PUT has been shown to converge to a set of excellent partition assignments within a reasonable amount of time. In practice, social graphs can be substantially so large that S-PUT may become too long to converge. In this case, we recommend that the original graph be partitioned into smaller subgraphs first and then S-PUT be applied on each subgraph.

The current framework of S-PUT is most suitable for implementation in a storage system with identical communication cost to go from one server to any other. In our future work, we will extend S-PUT for the case where this cost may vary. Also, we will investigate how to apply S-PUT to systems where geographic locality needs to be factored in, not just social locality. In S-PUT, once a partition has been determined, any arbitrary permutation of the servers can be used to assign each group of users to a separate server; S-PUT does not dictate the choice of this permutation. Therefore, to accommodate geographic locality, the application developer can choose a permutation that tries to preserve this property. We will explore this approach in our future research.

ACKNOWLEDGEMENTS

This work is partly supported by the research project No. 102.01.25.09 granted by Vietnam National Foundation for Science and Technology Development (NAFOSTED).

REFERENCES

[1] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGPLAN Not.*, vol. 35, pp. 190–201, November 2000.

[2] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," *SIGOPS Oper. Syst. Rev.*, vol. 35, pp. 188–201, October 2001.

[3] P. L. Reiher, J. S. Heidemann, D. Ratner, G. Skinner, and G. J. Popek, "Resolving file conflicts in the ficus file system," in *USENIX Technical Conference*, 1994, pp. 183–195.

[4] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Trans. Comput.*, vol. 39, pp. 447–459, April 1990.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, October 2003.

[6] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "Farsite: federated, available, and reliable storage for an incompletely trusted environment," in *Proceedings of the 5th symposium on Operating systems design and implementation*, ser. OSDI '02. New York, NY, USA: ACM, 2002, pp. 1–14.

[7] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in bayou, a weakly connected replicated storage system," *SIGOPS Oper. Syst. Rev.*, vol. 29, pp. 172–182, December 1995.

[8] C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *PVLDB*, vol. 3, no. 1, pp. 48–57, 2010.

[9] B. Carrasco, Y. Lu, and J. M. F. da Trindade, "Partitioning social networks for time-dependent queries," in *Proceedings of the 4th Workshop on Social Network Systems*, ser. SNS '11. New York, NY, USA: ACM, 2011, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/1989656.1989658

[10] D. A. Tran, K. Nguyen, and C. Pham, "S-CLONE: Socially-aware data replication for social networks," *Comput. Netw.*, vol. 56, no. 7, pp. 2001–2013, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.02.010

[11] 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," in *Proceedings of the ACM SIGCOMM 2010 Conference*. New York, NY, USA: ACM, 2010, pp. 375–386.

[12] P. R. C. E. Kallen, N. and J. Kalucki, "Introducing gizzard, a framework for creating distributed datastores," Twitter Engineering Website, April 2006, http://engineering.twitter.com/2010/04/ introducing-gizzard-framework-for.html.

[13] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, April 2010.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, October 2007.

[15] I. Hoque and I. Gupta, "Disk layout techniques for online social network data," *IEEE Internet Computing*, vol. 16, pp. 24–36, 2012.

[16] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms - a comparative case study," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, ser. PPSN V. London, UK: Springer-Verlag, 1998, pp. 292–304. [Online]. Available: http://dl.acm.org/citation.cfm?id=645824.668610

[17] S. Arora, S. Rao, and U. Vazirani, "Expander flows, geometric embeddings and graph partitioning," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, ser. STOC '04. New York, NY, USA: ACM, 2004, pp. 222–231.

[18] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, pp. 359–392, December 1998.

[19] M. E. Newman, "Modularity and community structure in networks," *Proc Natl Acad Sci U S A*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006.

[20] J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 631–640. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772755

[21] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[22] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization," in *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, K. Giannakoglou *et al.*, Eds. International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.