Data Replication for Distributed Graph Processing

Li-Yung Ho
Institute of Information Science
Academia Sinica,
Department of Computer Science and
Information Engineering
National Taiwan University
Taipei, Taiwan
Email: lyho@iis.sinica.edu.tw

Jan-Jan Wu
Institute of Information Science,
Research Center for
Information Technology Innovation
Academia Sinica,
Taipei, Taiwan
Email: wuj@iis.sinica.edu.tw

Pangfeng Liu
Department of Computer Science
and Information Engineering,
Graduate Institute of
Networking and Multimedia,
National Taiwan University
Taipei, Taiwan
Email: pangfeng@csie.ntu.edu.tw

***Abstract*—We present a data replication framework for distributed graph processing. First we partition a graph and store each partition in a machine. Then we replicate all partitions and assign replicas to machines, where each machine can store only a limited number of replicas. The goal is to replicate the partitions so that each partition has at least a certain number of replicated copies, and the cost is minimized. The cost is defined as the *data traffic* needed to run general graph processing algorithms. The cost metric is the *overall* transmission cost of all machines, and the *maximum* transmission cost of a single machine. We propose an optimal algorithm based on linear programming to solve the problem of minimizing the overall transmission cost. We also propose an optimal algorithm to solve a special problem of minimizing the maximum transmission cost of a node.**

***Index Terms*—algorithm, data replication, binary integer programming, minimum cost flow, totally unimodular, social networks**

I. Introduction

Graph is a general model to represent the relationships among objects. For instance, the links among websites, the network among machines or the relations among data are representative examples. The relation of data described by a graph, also known as *graph data*, have been extensively studied recently [1], [2], [3].

The main driving forces of graph data are the popularity of various social applications. Facebook [4], Twitter [5] and Youtube [6] have become the most popular social applications. The data stored on these social websites have different formats and are highly related. For example, one may tag a friend on a photo or twitter a message to friends. It is natural to model a social network with a graph. A node in the graph is a person in the social network, and an edge in the graph represents the relationship between two persons. Furthermore, queries on a social network can be easily described as graph traversals. As a result, how to efficiently process graph data has drawn much attention both in the academia and in the industry.

In recent years, a number of graph processing systems have been proposed to tackle the challenge of large scale graph processing. For example, Microsoft proposes an in-memory graph processing system named Trinity [2]. They claim that Trinity can handle billions of graph nodes. Google proposed Pregel [1], a bulk synchronization model to process large scale graph data, that targets at off-line graph analytic applications. Facebook uses Tao [7] to manage a social network containing over 800 million people. These works all propose a distributed graph processing system in order to handle big graph data that a standalone machine cannot process. These systems have different designed goals and target different applications and queries. Nevertheless, these systems must address the common issues in how to partition a graph for distributed processing, and how to replicate data to achieve fault tolerance and data availability.

Due to the large amount of data and limited capability of a single machine, the graph data are partitioned over multiple machines. Each partition is stored in a machine of a cluster. The first approach to partition a graph is to use a *hash function*. The system hashes a vertex ID to a machine ID, and store the vertex to the corresponding machine. Pregel, Trinity and Pegasus all use hashing to partition a graph. The second approach to partition a graph is to use METIS [8], which minimizes the number of edges that connect different partitions while balancing the amount of data among partitions. Systems use METIS to partition the graph data including [9], [10], [11]. Ho et al. [12] show that METIS achieves better data locality than hashing on real-world social networks.

Figure 1 illustrates an example of graph partition and the edges between two partitions. The graph in Figure 1 is partitioned into 4 parts separated by the blue dotted lines. The graph data of each part are stored in a machine. However, there are edges crossing different partitions (red line), which are called the *cut edges*. In distributed graph processing, the graph data are transmitted according to the topology of the graph. As a result, a *cut edge* means the data have to be transmitted across different partitions. Replicating the data of a partition can eliminate such data transmission. For example, if we replicate the data of partition 2 to the machine stored the data of partition 1, we can access the data of both partitions locally, without any remote access through the two cut edges between partition 1 and partition 2.

Data replication plays an important role in a large scale distributed system. Commodity server cluster has become the basic building block for cloud system nowadays. In such a system, failure is inevitable because the system consists of

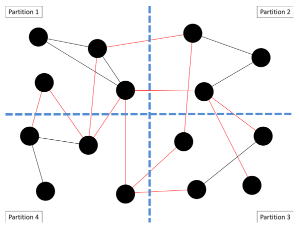


Fig. 1. A graph in four partitions

a large amount of machines and software components. A cloud system must replicate data to prevent data corruption or unavailability, in order to provide nonstop services.

We consider the following graph processing in this paper. We first partition a graph into several parts and store each part in a different machine. We refer to this copy as the *primary copy* of a partition. We assume that each partition has similar number of nodes and edges, which can be done by either hashing or METIS [8]. Then we replicate the primary copy into several *secondary copies* to achieve high availability and reliability. Note that how to partition a large graph is out of the scope of this paper. We do not discuss graph partitioning in this paper.

We study a *data replication problem* for distributed graph processing. We are interested in the following problem: how do we assign secondary copies to machines, so that the copies stored on the same machine are highly *connected*? Here we define the connectivity of two partitions to be *the number of edges* between these two partitions. If the partitions stored in a machine are highly connected, we will be able to access the neighbors of a node locally most of the time. This improves data locality of distributed graph processing and reduces the traffic to remote machines.

Formally, we partition a graph into N partitions, and store the i-th partition on the i-th machine. Then we replicate each partition as k replicas, where each replica is stored in a machine. We also assume that each machine can store *at most* l partitions. That is, we assign $N \cdot k$ partitions to N machines so that each machine has no more than l partitions, and a specified objective function is optimized.

We consider two objective functions for our data replication problems. The first objective function is the *total* costs of remote access of all machines. The second objective function is the *maximum* remote access cost among all machines. That is, we want to *reduce* the total remote access cost of all machines, or *balance* the remote access cost among all machines.

To quantify these objectives, we define communication cost as *the number of edges* between two partitions. For example, if we replicate the i-th partition on the j-th machine, which already has the j-th partition, we can eliminate the remote communication cost for any query that traverses between the i-th and the j-th partitions. We will formalize the data replication problem and the objective functions in Section III.

Data replication has been extensively studied in distributed system [13], [14], [15], [16], [17], [18]. However, we are not aware of any discussion on data replication of a graph. Moreover, the inherent connection among partitions increases the complexity of data replication. That is, one partition prefers to co-locate with specified partitions because they are well connected. This property complicates the partition assignment.

We present a unified *binary integer programming* model to discuss the above two data replication problems for distributed graph processing. The contributions of this paper are as follows.

- To our knowledge, this work is the first effort to study data replication problem for distributed graph processing.
- We develop an unified model to formulate two data replication problems crucial to distributed graph processing.
- We propose two polynomial-time optimal algorithms to solve these two data replication problems.

The rest of the paper are organized as follows. We review the related previous works in section II. We describe the data replication problems and two optimal algorithms to solve them in Section III. We conduct experiments to evaluate the effectiveness of the optimal algorithms in Section IV. Finally, we conclude this paper in Section V.

II. Related Works

Data replication is a common technique to achieve reliability and improve the performance of data retrieval in a distributed system. However, a data replication system must enforce the consistency among multiple copies of data, and retrieve data efficiently from multiple sites. Previous works mainly focus on how to minimize the response time of retrieving data [16], [17] and balance the load of machines storing the replications [15], [14], [13]. In contrast we consider a more general and complicated problem in which the replicated data have preferences to each other. We will describe the major differences between previous works and ours below.

Chen and Rotem [16] propose an optimal algorithm to retrieve a set of replicated data, such that the response time is minimized, by multiple runs of maximum flow algorithm. They assume that the response time of a machine is proportional to the amount of data retrieved from that machine. Therefore given the set of data to retrieve, one must determine which replica to retrieve, so that the maximum amount of data retrieved from any machine is minimized. Nihat et al. [17] improves the flow algorithm by Chen and Rotem [16] by eliminating the multiple runs of maximum flow algorithm.

Xu et al. [15] consider the problem of retrieving blocks of a file on Hadoop File System [19]. A file on HDFS consists of multiple blocks, and each block is replicated k times and each replica is stored on a machine of the Hadoop cluster. To retrieve a file the system needs to gather its blocks. We will do so by assigning a machine to retrieve a block. Since a block is replicated on multiple machines, we have to decide which replica should be retrieved by that machine. The amount of replicas to be retrieved should be evenly distributed among all machines. They define *evenly* as the case that the numbers of blocks assigned to retrieved by any two machine will differ by at most 1. If a machine is assigned to retrieve a replica that it does not have, then we refer to this retrieval as "remote". The goal of this problem is to minimize the total number of remote retrievals, so that the network traffic is reduced.

Our system model differs from those of Chen and Rotem [16], Nihat et al. [17], and Xu et al. [15]. We consider the cost of *replication* so that we need to consider all replicas on all machines. In contrast the algorithms in previous models in [16], [17], [15] consider *retrieval* cost, which is the cost of retrieving a *single* replica.

Shivakumar and Widom [18] considered user profile replication in a *Personal Communication Service* system. The replication problem is formulated as a network flow problem. The network consists of user profile nodes and database nodes. A link from a user profile node to a database node indicates the profile is replicated on the database. A user profile must be replicated at a fixed number of databases, and a database can only hold a limited number of user profiles. Although the system model is the same as the model in our data replication problem, we take a completely different algebraic approach to solve the problem instead of using maximum flow minimum cost algorithm to solve it. In addition, we also consider *the maximum cost* objective function in a unified framework, instead of only the *the total cost* objective function as in [18], [15].

In summary, we propose a general and unified model for the data replication problem in the context of distributed graph processing. The previous works are either special cases of our data replication problem or only consider a subset of objective functions we considered.

III. Data Replication Problem

We define *data replication problem* as to find the most *cost effective* way to replicate a set of data on a set of machines. We will define and discuss various cost metrics for data replication problem in the following subsections. We also propose two polynomial time algorithms that finds the optimal solutions for two data replication problems with different objective functions.

A. System Model

We model a data replication system as a directed bipartite graph $G = (V_1 \cup V_2, E)$. V_1 and V_2 are sets of vertex, and E is the set of edges. V_1 represents the set of data and V_2 represents the set of machines. To replicate a data u on a set of machines $\delta_u \subseteq V_2$, we add an edge from u to every vertex in δ_u. As a result every edge goes from a vertex in V_1 to a vertex in V_2 so G is bipartite. Formally, for each edge $e = (u, v)$, we have $u \in V_1$ and $v \in V_2$. Also note that we replicate k copies of a data object for fault tolerance and data availability, therefore the cardinality of δ_u is k, i.e., $|\delta_u| = k$.

A machine v may store a limited amount of replicas from different data. Let σ_v denote the set of data that were replicated on machine v. Formally, $\sigma_v = \{u | v \in \delta_u, \forall u \in V_1\}$. We assume that each machine has a capacity l such that a machine can hold at most l data replicas, therefore we have $|\sigma_v| \leq l$.

We use a function Γ to denote the cost to replicate a data on a machine. Formally Γ function maps an edge, which is from a data to a machine, to a cost, i.e., $\Gamma : E \mapsto \mathbb{R}$.

With the cost function Γ we can associate a replication cost to a data u or a machine v. We denote the cost of a data u as c_u, and is defined as the sum of cost to store all replicas of this data. Formally c_u is the sum of edges adjacent to data u, as indicated by Equation 1. Similarly, we denote the cost of a machine v as c_v, and is defined as the sum of cost to store all replicas on a machine. Formally c_v is the sum of edges adjacent to machine v, as indicated by Equation 2.

$$c_u = \sum_{\forall v \in \delta_u} \Gamma(e_{uv}) \qquad (1)$$

$$c_v = \sum_{\forall u \in \sigma_v} \Gamma(e_{uv}) \qquad (2)$$

B. Objective Functions

We now consider the *objective functions* of the data replication problem in this paper. An objective function is the metric to evaluate a data replication algorithm. After the replication algorithm replicates data to machines, we calculate the *cost* according to the objective function so that we can determine the effectiveness of the algorithm. The goal is to minimize the cost according to a particular objective function.

We consider two objective functions. The first objective function is *the sum of the cost*, and we denote it as θ_1. Formally $\theta_1 = \sum_{\forall u \in V_1} c_u$. Note that this definition is from the viewpoint of data. Equivalently we can also define θ_1 from the viewpoint of machine, since $\sum_{\forall u \in V_1} c_u = \sum_{\forall v \in V_2} c_v$. The second objective function is *the maximum of the cost*, and we denote it as θ_2. Formally $\theta_2 = \max c_w$, where $w \in V_1 \cup V_2$. That is, we consider the maximum of all costs for every data and machine.

Both objective functions have their merits. θ_1 indicates the *total costs* to replicate data to machines. Usually the cost is measured as data traffic and workload. By minimizing the total costs, we minimize the total data traffic and workload among all machines and data sources. On the other hand, θ_2 measures the cost of the most expensive node. By minimizing the cost of the most expensive node, we *balance* the data traffic and workload among all machines and data sources.

C. Problem Definition

We now formally define the *data replication problem* (DRP) as follows.

Definition 1: Given a bipartite graph $G = (V_1 \cup V_2, E)$, a replication factor k, a capacity constraint l, and a objective function θ, we want to find an assignment δ_u for each node $u \in V_1$, such that θ is minimized, given the following two constraints. The first constraint indicates that every data must be replicated at k different machines, and the second constraint indicates that every machine will not have more than l data replicas.

$$\begin{aligned} |\delta_u| &= k, \quad \forall u \in V_1 \\ |\sigma_v| &\leq l, \quad \forall v \in V_2 \end{aligned}$$

In the following subsections We propose efficient algorithms that gives optimal mappings for *DRP* with objective θ_1, and for a special case of *DRP* with objective θ_2.

D. Minimize the overall costs

To solve *DRP* with the minimum total costs, we first transform it into a *binary integer programming problem* and we show how to solve it optimally. We would like to minimize the following objective function.

$$\theta_1 = \sum x_{uv} \cdot \Gamma(e_{uv}) \tag{3}$$

The minimization of Equation 3 is subject to the following constraints. We use x_{uv} to denote whether data u selects machine v to store a replica – If so we set x_{uv} to 1, otherwise, we set it to 0.

$$\sum_{\forall v \in V_2} x_{uv} = k, \quad \forall u \in V_1 \tag{4}$$

$$\sum_{\forall u \in V_1} x_{uv} \leq l. \quad \forall v \in V_2, \tag{5}$$

$$x_{uv} \in \{0,1\} \tag{6}$$

We now re-write Equation 3 into Equation 7. Let γ be a constant vector of dimension $|E|$, γ_i be the i-th element of γ, and e_i be the i-th element of E. Then we define $\Gamma(e_i)$ to be the cost of edge e_i. Also we use X to denote the vector consisting of all x_{uv}, where the i-th element of of X is the i-th edge in E. That is, X is a $0-1$ vector of dimension $|E|$, and the i-th element indicates whether the i-th edge is selected. It is clear that the total costs are the summation of cost of those edges (u, v) where x_{uv} is 1, as in Equation 7.

$$\theta_1 = X \cdot \gamma \tag{7}$$

The minimization of Equation 7 is a *binary integer programming*, which is in general an NP-hard problem. However, we will show that we can solve the data replication problem in this case optimally in polynomial time, since Equation 3 has a property called *total unimodularity*. Before we introduce unimodularity we first define the *incidence matrix*.

Definition 2: An *incidence matrix* $\mathbb{I}$ of a graph $G = (V, E)$ is a matrix with $|V|$ rows and $|E|$ column. Let a_{ij} be the element of i-th row and j-th column of matrix $\mathbb{I}$. We define a_{ij} as follows.

$$a_{ij} = \begin{cases} 1 & \text{if the } i\text{-th vertex is incident to the } j\text{-th edge.} \\ 0 & \text{otherwise} \end{cases}$$

We now re-write Equation 4, and 5 with the incidence matrix $\mathbb{I}$. The constraints in Equation 4 and 5 are now combined into Equation 8. Recall that X is the vector consisting of all x_{uv}. b is a vector of dimension $|V|$ and the elements of b are either k or l, where k is the replication factor and l is the capacity constraint of a machine.

$$\mathbb{I} \cdot X \leq b \quad b \in \{k, l\}^{|V|} \quad X \in \{0,1\}^{|E|} \tag{8}$$

We can use a linear programming solver to derive a lower bound on the minimization objective for a binary integer programming. In addition, the solution from the linear programming does not necessarily consisting of 0 and 1, as required by a binary integer programming. However, we will show that if the incidence matrix in Equation 8 is *totally unimodular*, we can obtain the optimal solution of Equation 8 by a linear programming solver, i.e., the solutions from the linear programming will consist of only 0 and 1. We now define the concept of totally unimodular.

Definition 3: An integer matrix $\mathbb{I}$ is *totally unimodular* if every square submartix has determinant 0, 1 or -1.

Hoffman and Kruskal [20] showed that if a matrix $\mathbb{I}$ is totally unimodular, then the vertices of the polytope defined by $\mathbb{I}$ are all integral.

Theorem 1: [20] If $\mathbb{I}$ is a totally unimodular matrix, then all the vertices of the polytope $\mathbb{P} = \{x | \mathbb{I} \cdot x \leq b, x \geq 0\}$ are integral for any integer vector b.

We now establish the optimality of using linear programming solver to solve the data replication problem. Ahuja et al. [21] showed that incidence matrix for a bipartite graph is totally unimodular. Therefore we can apply Theorem 1 on our incidence matrix $\mathbb{I}$ and argue that the optimal solutions of a liner programming always happen at the vertices of a polytope, which will have integral coordinates. That is, the optimal solution will have x_{uv} either 0 or 1. Therefore we can use linear programming solver to obtain the optimal solutions of data replication problem with the objective of minimizing the total costs.

Theorem 2: Equation 8 can be solved optimally by linear programming.

The transformed binary integer programming problem is equivalent to a *maximum flow minimum cost problem*. To construct the network, we add two extra nodes – a source and a sink. The source links to every nodes of V_1 with an edge of capacity k, and every node of V_2 has a link to the sink of capacity l. Every link between V_1 and V_2 has capacity 1 and has a cost determine by the Γ function. The data replication problem with θ_1 is equivalent to finding a maximum flow with minimum cost. We can solve it by existing algorithms, e.g., cycle-canceling algorithms [22], [23] or network simplex algorithms [24].

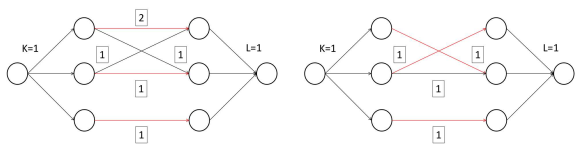
E. Minimize the maximum cost

Data replication problem with the maximum cost objective function is more difficult than with the total cost objective function. In this paper we only consider a special case in which the replication factor of data and the capacity of machine are both set to 1. We denote this problem as *DRP(k=1, l=1, θ_2)* or *DRP(1,1,θ_2)*, which is equivalent to finding a *perfect matching*.

Since the data is replicated only once, the mapping between data and machine is a *matching*. We define the *cost* of a matching between data and machine to be the maximum cost of any edge in the matching. Formally, given a bipartite graph $G = (V_1 \cup V_2, E)$ and a cost function $\Gamma : E \mapsto \mathbb{R}$. We define the cost of a matching M to be $C(M) = max\{\Gamma(e)|e \in M\}$ The goal is to find a matching that minimizes the cost.

We propose Algorithm 1 to solve *DRP(1,1,θ_2)* optimally. The key idea of Algorithm 1 is to select a bound on the edge cost, and remove edges that have higher cost. If we cannot find a perfect matching with the remaining edges, that means the bound is too small. If we do find a perfect matching, that means the bound can be reduced. By doing a binary search on the edge cost bound, we are able to find a matching in which the maximum edge cost is minimized, which is an optimal *DRP(1,1,θ_2)* solution.

Figure 2 illustrates the idea of Algorithm 1. The network consists of three data nodes and three machine nodes. The number in the square box indicates the cost of the edge. Figure 2(a) depicts a perfect matching (red lines) of cost 2. Algorithm 1 then reduces the bound to 1 and removes any edge that has cost larger than 1. As a result, the edge with cost 2 in Figure 2(b) is removed. In addition, we find a perfect matching with cost 1 and the cost of the perfect matching is minimized.



(a) perfect matching with cost 2 (b) perfect matching with cost 1

Fig. 2. Example of minimizing maximum cost

We now describe Algorithm 1 in details. We initially set the upper bound and lower bound to the maximum and minimum cost of an edge respectively (line 1 and line2). In the while loop, we run a maximum flow algorithm to check if there is a flow of volume $|V_1|$ passing through a network NET_B from s to t (line 5), i.e., if we can find a perfect matching. If we can (line 7), then we set the upper bound to B, otherwise (line 9), we set the lower bound to B. Then we calculate a new value of B (line 4), then call the function $NetWorkConstruction$ to construct a new network (line 5) and run the maximum flow algorithm on the newly constructed network again until the upper bound meets the lower bound.

IV. Experiment

We conduct experiments to investigate the performance of proposed algorithms. We compare the optimal solutions with the results of a hash algorithm and a heuristic greedy algorithm. There are two performance metrics – the total communication cost of a cluster, and the maximum communication cost of a machine in the cluster. We also developed an

Algorithm 1 Optimal Algorithm for *DRP(1,1,θ_2)*

```
Require: A bipartite graph $G = (V_1 \cup V_2, E)$, a cost function
    $\Gamma : E \mapsto \mathbb{R}$
Ensure: A matching $M$ with minimum cost
 1: $MAX \leftarrow max\{\Gamma(e)|e \in E\}$
 2: $MIN \leftarrow min\{\Gamma(e)|e \in E\}$
 3: while $MAX \neq MIN$ do
 4:   $B \leftarrow \frac{MAX-MIN}{2}$
 5:   $NET_B \leftarrow NetWorkConstruction(G, \Gamma, B)$
 6:   $M = MaxFlow(NET_B)$
 7:   if $|M| = |V_1|$ then
 8:     $MAX \leftarrow B$
 9:   else
10:     $MIN \leftarrow B$
11:   end if
12: end while
13: return $M$
```

application that sends messages to its neighbors multiple times to examine the performance of different replication strategies.

The hash algorithm works as follows. A partition selects a machine to store its replica by a hash function. If the chosen machine already has the replica or does not have room for the replica, the partition will find another machine to store the replica. A partition repeats this process until it replicates enough replicas. Note that the hash function uses a pseudo random number generator to give deterministic results.

The greedy algorithm works as follows. We first sort all the edges in increasing cost order. We then select edges one at a time according to this increasing cost order. During the process we may not be able to select an edge for two reasons. First the partition may have been replicated enough number of times. Second the machine may have reached its capacity. This process repeats until each partition has enough number of replicas.

We focus our graph processing experiments on social network analysis since it has drawn much attention in both in academia and industry in recent years. We collect three real-world social networks as our experiment graph data – Youtube [6], LiveJournal [25] and Flicker [26]. Mislove et al. [27] collect these data and publish them in an anonymous form. Table I summarizes the numbers of nodes and edges from these three social networks.

Social Networks	Nodes (millions)	Edges (millions)
YouTube	1.16	3.01
Flicker	1.86	15.7
LiveJournal	5.28	48.8

TABLE I
THE NUMBERS OF NODES AND EDGES OF THREE SOCIAL NETWORKS

In the following we first analyze the theoretical communication costs of the social networks with different replication strategies, then we evaluate the three replication strategies by the application that sends messages to its neighbors multiple times.

A. Theoretical Cost

We use METIS [8] to partition a social network into 16 partitions as the input to our *Data Replication Problem*, since we will use 16 machines in our experiments. We use a 16×16 matrix A to store the number of edges between any two given partitions. We assume that the communication cost between two partitions is proportional to the number of edges between them, and the proportional factor is 1. So we also use A_{ij} of matrix A to denote the communication cost between the i-th partition and the j-th partition.

We assume that the i-th machine has the primary copy of the i-th partition, we can also interpret the matrix A as follows. The rows of matrix A indicate the partition IDs and the columns of A indicate the machine ID. As a result we estimate the communication cost of the j-th machine to be the sum of the elements in the j-th column of A. Table II shows the matrix A of LiveJournal [25].

Replicating a partition on a machine eliminates the communication cost for the machine to retrieve it. For example, if we replicate the k-th partition on the j-th machine, the primary copy of the j-th partition will now be able to access the k-th partition *without* communication, i.e., a_{kj} becomes zero because we eliminate the communications for the j-th machine to retrieve the k-th partition.

B. Total Communication Cost

We evaluate the communication cost of the three replication strategies by comparing the overall communication cost. Each partition has two replicas as in [28], [29], and each machine has no more than two replicas. Table III summarizes the communication costs among hashing, greedy algorithm, and the optimal algorithm. The optimal algorithm has the lowest costs, and is used as a comparison basis. The greedy algorithm has a cost similar to that of the optimal algorithm. The hash method has the highest cost.

Note that the greedy algorithm may not always have two replicas for every partition. Despite the fact that we can always find a subgraph in which every machine connects to two partitions, and every partition connects to two machines. However, we may connect a partition to the machine where it is stored, due to lack of selection at the end of the greedy method. In that case this partition will have one less replica than it should have, and we use this subgraph as the input to the experiments.

The hash algorithm may not always have two replicas for every partition either, due to the same reason as in the greedy method. However, since the hashing is not deterministic, we can run the hashing algorithm repeatedly until we find a feasible solution.

We observe that the running time of three algorithms are negligible, which is about 0.165 seconds. This is because we have small problem size, that is, a 16×16 matrix. We conclude that all three algorithms are efficient enough in practice, especially in a cluster of reasonable size.

Table IV shows optimal replication that minimizes total communication cost in LiveJournal network. The element at the i-th row and the j-th column of Table IV indicates whether i-th partition is replicated at j-th machine. The table indicates that the j-th machine tends to replicate the j-th partitions to those machines with partitions that are well connected with the j-th partition. By doing so these machines can reduce the communication costs needed to retrieve it.

Social Networks	Hashing	Greedy	Optimal
YouTube	1363869	912671	900119
Flicker	8148224	6044052	6026941
LiveJournal	19290313	13310707	13251424

TABLE III
TOTAL NUMBER OF CUT-EDGES OF THREE REPLICATION STRATEGIES

	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
P0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
P1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
P2	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
P4	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
P5	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
P6	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
P7	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
P8	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
P9	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
P10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
P11	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
P12	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
P13	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
P14	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
P15	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

TABLE IV
THE OPTIMAL REPLICATION THAT MINIMIZES OVERALL COMMUNICATION COST IN LIVEJOURNAL

C. Maximum Communication Cost

We now focus on minimizing the maximum communication cost of a single machine. We consider the special case in which the capacity constraint and replication factor are both set to 1.

Social Networks	Hashing	Greedy	Optimal
YouTube	176093	176029	130628
Flicker	1555560	1655362	1222324
LiveJournal	2297041	2471157	2031564

TABLE V
MAXIMUM NUMBER OF CUT-EDGES FROM A MACHINE

Table V compares the maximum communication cost of a machine from different replication strategies for different social networks. Again we use the optimal strategy as the comparison basis. We observe from Table III and V find that the gap between number of cut-edges between the hash and the optimal method in minimizing the maximum communication cost, is much smaller than in minimizing the total communication cost. The running times of the three algorithms are also negligible, which are about 0.175 seconds.

Table VI shows the optimal replication that minimizes the maximum communication cost in LiveJournal network. It shows that two partitions tend to replicate each other due to their high connectivity. For example, partition $P14$ and $P15$ are strongly connected so they replicate each other.

	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
P0	0	94162	105028	35092	64300	43060	51329	72930	49424	97454	31465	46631	21874	2195	26217	56324
P1	94161	0	119222	56290	113717	69796	81430	111788	60133	115740	49037	62087	32701	2678	40470	98117
P2	105027	119222	0	71801	103119	98286	90073	142868	41634	140978	59220	68978	33181	3414	49860	91060
P3	35092	56290	71801	0	52984	48601	42938	67064	40284	101722	32234	44759	20429	2437	24431	50513
P4	64299	113717	103119	52984	0	77552	91749	128719	53146	144878	51235	59341	27235	2810	36938	64099
P5	43060	69796	98286	48601	77552	0	61173	84755	75796	131012	38625	61774	22744	2502	32132	55908
P6	51329	81430	90073	42938	91749	61174	0	192094	46633	174393	56964	60979	27108	3705	43771	61218
P7	72930	111788	142868	67064	128719	84755	192093	0	45949	190825	72308	71302	38028	3689	46420	82468
P8	49425	60133	41634	40284	53146	75796	46633	45949	0	541315	114290	193844	32054	11574	50106	169814
P9	97454	115740	140978	101722	144878	131012	174393	190825	541315	0	287743	275838	105770	39405	77456	148351
P10	31465	49037	59220	32234	51235	38625	56964	72308	114290	287742	0	109012	50537	11226	69223	132066
P11	46631	62087	68978	44760	59341	61774	60978	71302	193844	275838	109012	0	47691	10826	160691	117639
P12	21873	32701	33181	20429	27235	22744	27108	38028	32054	105770	50537	47691	0	1630911	49794	73269
P13	2195	2678	3414	2437	2809	2502	3705	3689	11574	39405	11226	10826	1630911	0	9137	13210
P14	26217	40470	49860	24431	36938	32132	43771	46420	50106	77456	69223	160690	49794	9137	0	200385
P15	56324	98117	91059	50513	64099	55908	61219	82468	169814	148351	132066	117639	73269	13210	200385	0

TABLE II
THE CUT-EDGES MATRIX OF LIVEJOURNAL

	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
P0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
P4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
P5	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
P6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
P7	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
P8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P9	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
P10	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
P11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P12	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
P15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

TABLE VI
THE OPTIMAL REPLICATION THAT MINIMIZES THE MAXIMUM COMMUNICATION COST IN LIVEJOURNAL

D. *Application*

We develop an application SendToNeighbor to evaluate the performance of the three replication strategies. At the beginning of SendToNeighbor a subset of vertices in the graph send messages to their neighbors. Those vertices after receiving messages will send messages to their neighbors. This process repeats until a specified number of iteration. After the specified number of iterations all vertices halt and the computation finishes. SendToNeighbor sets the length of the message to 100 characters, sets the number of iteration to 25, and randomly selects 80% of the vertices to start sending messages at the beginning. Also note that SendToNeighbor will only read data so we do not need to consider data consistency among replicas.

Sending messages to neighbors is a typical communication pattern in social network applications. For example PageRank [1], Influence Spreading [12], and Max Value propagation [1] all involve sending messages to neighbors.

We implement SendToNeighbor on Hama [30]. Hama is an open source implementation of Pregel [1], which is based on bulk synchronization model. We customize Hama into our replication strategies by implementing the Partitioner interface. At the beginning Hama sends each vertex to a machine for processing. The Partitioner decides which machine processes which vertex. We implement our replication logic in Partitioner so that the primary copy of a vertex is sent to a specified machine, and the secondary copies are sent to other machines according to the replication strategy.

When a vertex v at machine m wants to send a message to its neighbor n, there are two possibilities. First, n has a replica at machine m, so the message will go to the replica. Otherwise the message will go the machine that has the primary copy of n.

The experiment configuration is as follows. We run the experiment on a 16-node cluster. Each node equips with an Intel Xeon E5504 2GHz CPU and 10GB of ram. We use Hama 0.6.0 to implement all the replication strategies. The performance metric is the execution time of all the iterations. Note that this performance metric does not include the time to load and replicate data before the iterations can start. Each data point is the average from 10 runs. We set the replication factor and the capacity constraint to 2 while minimizing the overall communication cost, and to 1 while minimizing the maximum communication cost.

Table VII and VIII compare the execution time of SendToNeighbor of different replication strategies and social networks. The speedup in the column indicates the performance improvement compared with Hashing method. In Table VII, the execution time of SendToNeighbor using hashing algorithm is 30% slower than using the optimal solution with LiveJournal as input, and overall communication cost as the objective function. Since the optimal replication strategy minimizes the number of cut edges, it has less communication messages than the hashing method does. The execution time using greedy algorithm is within 5% of the optimal algorithm. On the other hand, as shown in Table VIII, when we try to minimize the maximum communication cost, the execution time of SendToNeighbor using hashing algorithm is 13% slower than using the optimal solution, also with LiveJournal as input. Table VII and VIII also indicates that the execution time difference among the three strategies is not significant for small networks, e.g., YouTube. The reason is that the relatively small number of messages will not form communication bottleneck.

Table IX shows the number of messages and we confirm that the optimal replication strategy has the fewest messages. Similarly Table X show the number of messages while minimizing maximum communication cost. Table IX and X indicate that fewer messages improves performance.

Social Networks	Hashing	Greedy(Speedup)	Optimal(Speedup)
YouTube	53.7294	44.3369(17.42%)	43.5863(18.88%)
Flicker	423.5676	284.1521(32.91%)	283.826(32.99%)
LiveJournal	966.7852	680.124(29.65%)	642.7784(33.51%)

TABLE VII
EXECUTION TIMES (IN SECONDS) OF SendToNeighbor WITH MINIMIZING TOTAL COMMUNICATION COST

Social Networks	Hashing	Greedy(Speedup)	Optimal(Speedup)
YouTube	53.221	51.861(2.56%)	48.523(8.83%)
Flicker	371.1072	412.1372(-11.06%)	356.9483(3.82%)
LiveJournal	859.5838	838.5003(2.45%)	743.4389(13.51%)

TABLE VIII
EXECUTION TIMES (IN SECONDS) OF SendToNeighbor WITH MINIMIZING THE MAXIMUM COMMUNICATION COST

V. CONCLUSION

In this work, we study a data replication problem (*DRP*) for distributed graph processing. The goal of *DRP* is to replicate partitions of a graph such that each partition has at least a certain number of replicated copies, and the cost is minimized. We consider two cost metrics in this paper. One is to minimize the total communication cost, and the other is to minimize the maximum cost of a perfect matching.

We propose two optimal algorithms for both objectives respectively. We solve *DRP* with minimizing total cost by a linear programming solver, and we solve *DRP* with minimizing maximum cost by an algorithm based on binary search technique and maximum flow algorithm.

Finally, we conduct experiments to evaluate the effectiveness of the proposed algorithms and compare the results with the hashing assignment strategy and a heuristic greedy algorithm. In minimizing total communication cost experiment, the optimal solutions save up to 30% of cost than the results of the hashing assignment. In minimizing maximum cost experiment, the optimal solutions outperform the hashing assignment up to 13%.

REFERENCES

[1] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 international conference on Management of data*, 2010.

Social Networks	Hashing	Greedy	Optimal
YouTube	38623769	29766218	26877441
Flicker	216762750	162729836	162285028
LiveJournal	514344316	379463144	360391339

TABLE IX
NUMBER OF MESSAGES WITH MINIMIZING TOTAL COMMUNICATION COST

Social Networks	Hashing	Greedy	Optimal
YouTube	39198196	34795395	31738458
Flicker	213402911	222252678	196641328
LiveJournal	524851846	428433548	408247450

TABLE X
NUMBER OF MESSAGES WITH MINIMIZING THE MAXIMUM COMMUNICATION COST

[2] "Trinity," http://research.microsoft.com/en-us/projects/trinity/default.aspx.
[3] "Neo4j," http://neo4j.org/.
[4] "Facebook," http://www.facebook.com/.
[5] "Twitter," http://twitter.com/.
[6] "Youtube," http://www.youtube.com/.
[7] V. Venkataramani, Z. Amsden, N. Bronson, G. Cabrera III, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, J. Hoon, S. Kulkarni, N. Lawrence, M. Marchukov, D. Petrov, and L. Puzar, "Tao: how facebook serves the social graph," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.
[8] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Supercomputing, 1998. SC98. IEEE/ACM Conference on*, 1998.
[9] 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*, 2010.
[10] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *Proc. VLDB Endow.*, vol. 3, no. 1-2, Sep. 2010.
[11] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new parallel framework for machine learning," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2010.
[12] L.-Y. Ho, J.-J. Wu, and P. Liu, "Distributed graph database for large-scale social computing," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012.
[13] T. Pitoura, N. Ntarmos, and P. Triantafillou, "Replication, load balancing and efficient range query processing in dhts," in *Advances in Database Technology - EDBT 2006*, ser. Lecture Notes in Computer Science, Y. Ioannidis, M. Scholl, J. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, Eds., 2006, vol. 3896.
[14] H. Yamamoto, D. Maruta, and Y. Oie, "Replication methods for load balancing on distributed storages in p2p networks," *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, vol. 0, 2005.
[15] Y. Xu, P. Kostamaa, Y. Qi, J. Wen, and K. K. Zhao, "A hadoop based distributed loading approach to parallel data warehouses," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011.
[16] L. T. Chen and D. Rotem, "Optimal response time retrieval of replicated data (extended abstract)," in *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1994.
[17] N. Altiparmak and A. S. Tosun, "Integrated maximum flow algorithm for optimal response time retrieval of replicated data," in *Parallel Processing (ICPP), 2012 41st International Conference on*, 2012.
[18] N. Shivakumar and J. Widom, "User profile replication for faster location lookup in mobile environments," in *Proceedings of the 1st annual international conference on Mobile computing and networking*, 1995.
[19] "Hadoop file system," http://hadoop.apache.org/hdfs/.
[20] A. J. Hoffman, J. B. Kruskal, I. Alan, J. Hoffman, and J. B. Kruskal, "Chapter 3 integral boundary points of convex polyhedra," 1956.
[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*, 1993.
[22] P. T. Sokkalingam, R. K. Ahuja, and J. B. Orlin, "New polynomial-time cycle-canceling algorithms for minimum cost flows," *NETWORKS*, vol. 36, pp. 53–63, 1996.
[23] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles," *J. ACM*, vol. 36, no. 4, Oct. 1989.
[24] J. B. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," in *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, 1996.
[25] http://www.livejournal.com/.
[26] http://www.flickr.com/.
[27] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, 2007.
[28] "Hadoop," http://hadoop.apache.org/.
[29] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.
[30] "Hama," http://en.wikipedia.org/wiki/Apache-Hama.