Sharding Social Networks

Quang Duong
University of Michigan
Computer Science and Engineering
Ann Arbor, MI
duonganhquang@gmail.com

Sharad Goel
Yahoo! Research
111 West 40th Street
New York, NY 10018
5harad.6oel@gmail.com

Jake Hofman
Yahoo! Research
111 West 40th Street
New York, NY 10018
jhofman@gmail.com

Sergei Vassilvitskii
Yahoo! Research
111 West 40th Street
New York, NY 10018
sergeiv@gmail.com

ABSTRACT

Online social networking platforms regularly support hundreds of millions of users, who in aggregate generate substantially more data than can be stored on any single physical server. As such, user data are distributed, or sharded, across many machines. A key requirement in this setting is rapid retrieval not only of a given user's information, but also of all data associated with his or her social contacts, suggesting that one should consider the topology of the social network in selecting a sharding policy. In this paper we formalize the problem of efficiently sharding large social network databases, and evaluate several sharding strategies, both analytically and empirically. We find that random sharding—the de facto standard—results in provably poor performance even when nodes are replicated to many shards. By contrast, we demonstrate that one can substantially reduce querying costs by identifying and assigning tightly knit communities to shards. In particular, we introduce a scalable sharding algorithm that outperforms both random and location-based sharding schemes.

1. INTRODUCTION

As the popularity of online social networks such as Facebook, Twitter and LinkedIn grows, it becomes increasingly challenging to maintain scalable data infrastructures. In particular, the amount of uploaded user content, including photos, videos, and status messages, far exceeds the storage capacity of any single machine, and thus user data must necessarily be distributed, or sharded, across hundreds or even thousands of servers.

Rapidly retrieving these data poses a particularly challenging problem, as one must support billions of queries each day while balancing latency against memory and bandwidth constraints. The difficulty of the retrieval task is compounded by the fact that users often require access not only to their own data, but also to that of all their social contacts. Such neighborhood queries, in fact, are the key ingredient in generating popular, personalized user feeds—which present activities from a user's social circle after filtering for recency and relevance—and thus exert considerable strain on existing database systems.

This paper addresses the problem of sharding social networks so as to efficiently execute such neighborhood queries. A simple and oft-used sharding strategy is to randomly assign users to shards. At a high level, however, there are two near-universal features of social networks that suggest one can do substantially better. First, social networks generally contain densely connected communities of users. By ensuring tightly knit clusters of users are assigned to the same shard, one might greatly reduce the number of shards accessed per query. Second, a relatively small number of "locally popular" users (i.e., those that are frequently queried by certain communities) account for a disproportionally large number of shard accesses. Neighborhood queries might thus be further optimized by replicating local celebrities to the appropriate shards, analogous to maintaining a cache of frequently accessed resources. By leveraging these properties, network-aware sharding strategies offer the potential for considerable gains over the alternatives.

Several recent papers have in fact demonstrated the benefits of sharding network data based on correlated access patterns. Karagiannis et al. [8] deploy and evaluate Hermes, an optimization engine for large-scale enterprise email services in which observed communication data are analyzed to co-locate user data based on the implicit social graph. Such co-location reduces storage costs as senders and receivers of an email can access the same physical version of the message, obviating the need for each to keep separate local copies. Agarwal et al. [1] detail and analyze a similar system, Volley, for placing data across geo-distributed servers. Like Hermes, Volley is based on an implicit network extracted from access logs, and the system is shown to considerably reduce storage and bandwidth requirements relative to a baseline strategy of placing resources close to their primary requesters. Similarly, SPAR, a system developed by Pujol et al. [14] is designed to facilitate strict co-locality requirements in online social networks—where a user and all its neighbors must reside on the same server—and the objective is to minimize the number of necessary shards. In addition to this work on social network databases, there is an extensive literature on the more general problem of parallelizing indexes for traditional information retrieval [12, 13], though the topology of bipartite networks between queries and terms differs substantially from the friendship graphs central to our applications.

Past work has approached network-aware sharding from an almost exclusively empirical perspective, focusing on both storage and bandwidth. Here we formalize the problem as minimizing average load across shards, and both derive analytic results and present empirical evaluations. We show that in graph models with strong community structure, random shardings, though the de facto industry standard, are far from optimal and lead to provably worst-case performance. In contrast, by simply mapping communities in these networks to shards, one can greatly reduce the average load exerted by neighborhood queries.

As we show, finding optimal network-aware sharding strategies is unfortunately an NP-complete problem. In light of this limitation, however, we present a novel approximate method for network sharding that scales to networks with tens of millions of nodes and billions of edges. Our algorithm proceeds in two steps: Densely connected regions are first identified in the graph, and these communities are then greedily assigned to shards while respecting capacity constraints. We note that though our community detection method is tailored to network sharding, it in fact generalizes and lends insight into the widely used label propagation algorithm [15].

We validate this approach on two large internet-scale datasets, and find that average query load is more than halved relative to random sharding. For the smaller of the two networks we show that network-aware sharding additionally improves upon geographic sharding, and for the larger we demonstrate the ability to scale to networks that cannot be handled by existing methods (e.g., METIS). Decrease in average load, however, comes at the expensive of creating hotspots (i.e., shards with much higher-than-average loads), particularly in the case of networks with extreme degree skew. We show that this load imbalance is in large part mitigated by replicating, or caching, a relatively small number of local celebrities to shards, and moreover, this replication further reduces average load.

Roadmap. We proceed by formally defining the problem in Section 2 and proving that random sharding is a poor strategy for minimizing the average load. To motivate alternate sharding strategies, we review the stochastic block model for network structure in Section 3 and show that significantly better shardings exist for networks following this model (Theorem 6). After showing that networking sharding is NP-complete in Section 4, we give a scalable, two-stage approach for finding approximate solutions to the sharding problem in Section 5. We then evaluate this method experimentally on two real-world networks in Section 6 and discuss open questions in Section 7.

2. PROBLEM STATEMENT

Before formally defining the network sharding problem, we introduce some notation. Let $G = (V, E)$ be a directed graph with $n = |V|$ nodes, and $m = |E|$ edges. Let $N(i) = \{j : (i, j) \in E\}$ denote the neighborhood of the i-th node, and $d_i = |N(i)|$ its degree. For $S, T \subseteq V$, where the subsets are not necessarily disjoint, let $E(S, T) = E \cap (S \times T)$ be the set of edges from S to T. We abbreviate $E(\{i\}, T)$ as $E(i, T)$ and $E(S, \{j\})$ by $E(S, j)$.

Given a graph G, our goal is to cover the n nodes of G with T potentially overlapping sets $S_1, S_2, \ldots, S_T$. We refer to each of these sets as a *shard*, and insist that each shard contain at most M nodes. Semantically, each shard corresponds to the subset of G that will be stored on an individual machine. We refer to the collection of shards, $\mathcal{S} = \{S_1, \ldots, S_T\}$, as a *sharding*. We note that if $M \cdot T = n$, the shards are necessarily disjoint; otherwise let $\rho = MT/n > 1$ define the *replication ratio*, which characterizes the level of node duplication in the system.

As described in the introduction, the goal of the system is to support an efficient execution of *neighborhood queries*, queries that retrieve a node i together with all of its neighbors. Given a sharding, $\mathcal{S}$, there are many valid query plans that define which shards to use for any particular neighborhood query. Formally, a query plan is a set of indices:

$$\mathcal{Q} = \{e_{ij}\} \qquad 1 \le i \le n,\ j \in i \cup N(i),\ e_{ij} \in \{1, \ldots, T\}$$

that indicate where to access node j when executing a neighborhood query for i. A query plan is considered valid for a sharding $\mathcal{S}$ if it specifies nodes should be retrieved only from shards on which they actually reside (i.e., $j \in S_{e_{ij}}$).

From a query plan alone, it is easy to generate a compatible sharding (i.e., a sharding for which the query plan is valid). Namely, a node is assigned to a given shard if and only if at least one neighborhood query expects to find it there:

$$S_t = \left\{ j \;\middle|\; t \in \bigcup_{i : j \in i \cup N(i)} \{e_{ij}\} \right\}. \qquad (1)$$

We thus consider a query plan to be valid if the induced shards satisfy the capacity constraints $|S_t| \le M$. We note that in the case of a partitioning (i.e., $\rho = 1$), there is a unique compatible query plan corresponding to looking up each node on the shard containing it.

2.1 Objective

In practice, the overall performance of a query plan depends on several factors, including the number of queries per second handled by each shard, the latency of network connections, and the total network bandwidth consumed. A simple abstraction of these performance metrics is the average query load across shards. (We empirically examine the full load distribution in Section 6 with the aim of finding and remedying hotspots in the system.) Let λ_i be the rate at which node i queries the system. Denote by $L_{i,t}(\mathcal{Q})$ the load induced by user i on shard t under query plan $\mathcal{Q}$:

$$L_{i,t}(\mathcal{Q}) = \begin{cases} \lambda_i, & \text{if } t \in \bigcup_{j \in i \cup N(i)} \{e_{ij}\} \\ 0, & \text{otherwise} \end{cases}$$

Then the average load per shard is:

$$\text{load}(\mathcal{Q}) = \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} L_{i,t}(\mathcal{Q}). \qquad (2)$$

In our analysis, it is helpful to consider the average load generated by the user, which we call the *cost* of a query plan:

$$\mathrm{cost}(\mathcal{Q}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{T} L_{i,t}(\mathcal{Q}). \tag{3}$$

Note that the two measures are scalar multiples of each other as $\mathrm{load}(\mathcal{Q}) = \frac{n}{T}\mathrm{cost}(\mathcal{Q})$. Minimizing cost is thus equivalent to minimizing load, and so we switch between the two formulations as convenient. For simplicity in our presentation we assume that $\lambda_i = \lambda$ for all i (i.e., that all users query the system at the same, constant rate), but note that the results extend to the heterogeneous user setting as well.

We are now ready to formally state the NetworkSharding problem.

Problem 1 (NetworkSharding). *Given a graph G, the total number of shards, T, and a per shard capacity constraint M, find a valid query plan $\mathcal{Q}$ with minimal cost.*

2.2 Random Sharding

Under our cost metric the worst one can do is to access a node and each of its neighbors on distinct shards, in which case the cost is approximately the average network degree. This is exactly the situation that arises when one employs the common strategy of randomly assigning nodes to shards regardless of network structure. Theorem 2 formalizes this result, showing that even with substantial replication random sharding leads to near worst-case performance, generally requiring nodes to access distinct shards for each neighbor.

Theorem 2. *Let G be a graph with $n \geq 2$ nodes, and let $T, M \geq 1$ be integers such that $TM \geq n$. Consider a random sharding of G into T shards with capacity M chosen uniformly from all shardings that fill each shard to capacity, and let Q denote an optimal query plan compatible with that sharding. Then if $\rho = TM/n$ is the replication ratio, the expected cost of Q satisfies*

$$\frac{1}{n}\sum_{i=1}^{n} d_i - \frac{\rho M}{n}\cdot\frac{1}{n}\sum_{i=1}^{n} d_i^3 \leq \mathbb{E}\left[cost(\mathcal{Q})\right] \leq 1 + \frac{1}{n}\sum_{i=1}^{n} d_i \tag{4}$$

where d_i is the degree of node i.

Before proving the theorem, we make two observations. First, given a sharding, generating an optimal compatible query plan is in general an NP-complete problem. In particular, it requires solving a set cover problem for each node and its neighbors. Nevertheless, we show that this optimal query plan is necessarily poor when the sharding is chosen uniformly at random. Second, suppose d_i is $O(\log n)$ and the replication ratio is as high as $O(n^{1/2})$. Then even for very high memory capacities—for example, $M = O(n^{1/2-\epsilon})$—we still find that random sharding leads to near worst-case performance.

Proof. The upper bound follows trivially, since in the worst case each node must access its neighbors and itself in $d_i + 1$ different shards. Likewise, if $M = 1$ then each node must choose different shards for each neighbor, and if $M > n$ then the left-hand side of (4) is negative, so the lower bound holds in these degenerate cases.

To establish the lower bound for $2 \leq M \leq n$, fix a node i, and consider the probability that i has at least two neighbors in the t-th shard, S_t. Applying the union bound over all pairs of i's neighbors, we have

$$\begin{aligned}\mathbb{P}(|N(i)\cap S_t| \geq 2) &\leq \binom{d_i}{2}\binom{n-2}{M-2}\Big/\binom{n}{M} \\ &\leq d_i^2\frac{M(M-1)}{n(n-1)} \\ &\leq \left(\frac{d_i M}{n}\right)^2.\end{aligned}$$

Consequently, the probability that i has at most a single neighbor in any of the T shards is lower bounded by

$$1 - T\left(\frac{d_i M}{n}\right)^2 = 1 - \frac{\rho M d_i^2}{n}.$$

When no shard contains more than a single neighbor of i, i must access at least d_i shards. The expected number of shards i accesses is thus lower bounded by

$$d_i - \frac{\rho M d_i^3}{n}.$$

The result now follows by averaging the expected number of shard accesses over all nodes. □

Even when a graph has substantial community structure, Theorem 2 shows that random shardings, perhaps unsurprisingly, perform poorly at neighborhood queries. In contrast, as we show in Section 3, network-aware sharding strategies can perform considerably better in the context of stochastic block models.

3. STOCHASTIC BLOCK MODEL

Stochastic block models (SBMs) constitute a widely used family of random networks with community structure [7]. We specify the generative process underlying this model in Section 3.1 and show in Section 3.2 that a provably optimal sharding strategy maps these communities to individual shards. This analysis informs our approximate solution to the NetworkSharding problem, presented in detail in Section 5.

3.1 Model Description

Stochastic block models [7] constitute a simple, yet widely used, family of networks with community structure. In these models, nodes belong to one of K communities, or blocks, and the probability of an edge between any two nodes depends only on their corresponding block assignments. Specifically, the generative process for a network with n nodes and K blocks is as follows:

- For each node i, independently roll a K-sided die with bias $\vec{\pi}$ to determine the node's block assignment $z_i \in \{1, \ldots, K\}$.

- For each ordered pair of nodes (i, j), flip a coin with bias θ_+ (resp. θ_-) for nodes in the same (resp. different) blocks to determine if an edge exists from i to j.

Formally, for an SBM with parameters $\vec{\pi}$ and $\vec{\theta}$, the marginal distributions for block assignments z_i and edges indicated by

A_{ij} are:

$$p(z_i = k|\vec{\pi}) \quad = \quad \pi_k \tag{5}$$

$$p(A_{ij} = 1|z_i, z_j, \vec{\theta}) \quad = \quad \begin{cases} \theta_+ \text{ if } z_i = z_j \\ \theta_- \text{ if } z_i \neq z_j \end{cases}, \tag{6}$$

where $0 < \theta_\pm < 1$, $0 < \pi_k < 1$, and $\sum_{k=1}^K \pi_k = 1$.

Networks generated under this model may loosely be characterized as a mixture of Erdos-Renyi networks, with an edge density θ_+ within blocks and θ_- between. In the assortative case ($\theta_+ > \theta_-$), nodes tend to form more edges within than between their blocks, resulting in dense "communities."

3.2 Analysis

Despite their simplicity, such models capture one of the most salient features of social networks, namely the tendency of individuals to cluster into communities. Moreover, these models immediately illustrate the potential savings from network-aware sharding strategies. By assigning all nodes in a block to the same shard, a node need only access a single shard to retrieve all its in-block neighbors. Thus, in the worst case, nodes only access different shards for each of their out-of-block neighbors, leading to a total cost approximately equal to the average out-block degree. In contrast, Theorem 2 shows that the expected cost of a random sharding of this network—in fact, of any network—is approximately the average degree, which is generally substantially larger than the average out-of-block degree.

While the naive sharding strategy of assigning blocks to shards offers considerable gains over a random sharding, one might reasonably wonder if there is further structure embedded in these networks that one can exploit. Theorem 6 below shows that in fact there is not, that the best one can do is to simply leverage the first-order structure of these graphs. While this result is quite believable, rigorously establishing its validity requires a detailed understanding of the structure of Erdos-Renyi graphs.

We proceed by first establishing in Theorem 4 that Erdos-Renyi graphs have no discernible structure one can exploit for sharding. This result follows from an observation by Bollobás[3] that Erdos-Renyi graphs do not contain large, isolated sets of nodes. Finally, by considering SBMs as a mixture of Erdos-Renyi graphs of different densities, Theorem 6 establishes our main result that SBMs are best partitioned by assigning blocks to shards.

Theorem 3 (Bollobás[3]). *Suppose $\delta = \delta(n)$ and $D = D(n)$ satisfy $\delta pn \geq 3 \log n$, $D \geq 3 \log(e/\delta)$ and $D\delta n \to \infty$. Then for every $\epsilon > 0$, $\exists N$ such that for $n \geq N$ and $G \sim G(n,p)$, G satisfies the following property with probability at least $1 - \epsilon$: For every $U \subseteq V$ with $|U| = \lceil D/p \rceil$ the set*

$$T_u = \{x \in V \setminus U \mid N(x) \cap U = \emptyset\}$$

has at most δn elements.

Theorem 3 shows that in most Erdos-Renyi graphs, every suitably large set contains a neighbor of most nodes in the network. Importantly, it is not only the case that most large sets are well-connected to the rest of the network, but that in *most graphs, every large set* is well-connected. It is this subtle distinction that constitutes the primary difficulty in establishing the result.

Theorem 4 below recasts the Bollobás result as a statement about sharding, and can be seen as an analog of Theorem 2. Whereas the former result established that completely random *shardings* lead to poor performance, the latter shows that completely random *graphs* can not be efficiently sharded.

Theorem 4. *Suppose $p = p(n)$ satisfies $p \geq (6 \log n)/n$. Then for every $\epsilon > 0$, $\exists N$ such that for $n \geq N$, $G \sim G(n,p)$, and positive integers T, M with $TM = n$, the cost of an optimal query plan $\mathcal{Q}$ satisfies*

$$c_0 \min(T, np) \leq cost_{opt}(\mathcal{Q}) \leq c_1 \min(T, np) \tag{7}$$

with probability $1 - \epsilon$, where $c_0 = 1/50$ and $c_1 = 3$.

Remark 5. *By assuming $TM = n$ in Theorem 4, we exclude the possibility of nodes residing on multiple shards. That is, the result pertains to a traditional sharding in which nodes are partitioned, but not replicated, across machines.*

Proof. We start with the upper bound. The cost of any query plan is trivially at most $\min(T, \bar{d} + 1)$, where $\bar{d}$ is the average degree of G. Let E_i be independent random variables indicating whether the i-th edge is present in G. Then $\bar{d} = \frac{2}{n} \sum_i E_i$. We can use the Chernoff bound to conclude that:

$$\mathbb{P}\left(\bar{d} > 2 \cdot \mathbb{E}[\bar{d}]\right) \quad \leq \quad \left(\frac{e}{4}\right)^{\mathbb{E}[\bar{d}]}$$

Since $\mathbb{E}[\bar{d}] = (n-1)p \geq 5 \log n$, we can conclude that $\mathbb{P}\left(\bar{d} \leq 2np\right) \geq 1 - \epsilon/2$ for n sufficiently large, establishing the upper bound.

For the lower bound, suppose $\mathcal{S}$ is a sharding $(S_1, \ldots, S_T)$ of G. Since we assume $TM = n$, $\mathcal{S}$ necessarily partitions the nodes of G, and so there is a unique compatible query plan, which we denote by $\mathcal{Q}$. Since G is undirected, node i accesses the t-th shard if and only if $i \in N(S_t)$ (i.e., if i has a neighbor in S_t). Thus, if $I_A(i)$ indicates whether i is in the set of nodes A,

$$\begin{aligned} \text{cost}(\mathcal{Q}) &= \frac{1}{n} \sum_{i=1}^n \sum_{t=1}^T I_{N(S_t)}(i) \\ &= \frac{1}{n} \sum_{t=1}^T \sum_{i=1}^n I_{N(S_t)}(i) \\ &= \frac{1}{n} \sum_{t=1}^T |N(S_t)|. \end{aligned} \tag{8}$$

In other words, the cost of the query plan can be expressed as the sum of the sizes of each shard's neighborhood.

We next apply Theorem 3 to show that every suitable large node set has a large neighborhood. Specifically, take $\delta = 1/2$ and $D = 6$ in Theorem 3. Then for n sufficiently large and almost all graphs $G \sim G(n,p)$, any vertex set U of size $\lceil 6/p \rceil$ has $|T_U| \leq n/2$. So U has a neighborhood of size satisfying

$$\begin{aligned} |N(U)| &\geq n - |U| - \frac{n}{2} \\ &\geq \frac{n}{2} - \frac{n}{\log n} \\ &\geq \frac{n}{3} \end{aligned} \tag{9}$$

where the last inequality follows by taking $n \geq e^6$.

To finish the proof, we consider two cases. First suppose $M \geq \lceil 6/p \rceil$, that the machines have relatively high memory. Then by (9), for almost all graphs G and any sharding S of G, each shard has $|N(S_t)| \geq n/3$. Summing over the T shards and applying (8), we have $\text{cost}(\mathcal{Q}) \geq T/3$. So, when memory is high (and thus there are relatively few shards), nodes tend to access a constant fraction of the shards. Alternatively, suppose that the machines have small capacity, that $M < \lceil 6/p \rceil$. In this case, consider the union U of $\lceil \lceil 6/p \rceil / M \rceil$ distinct shards, which, as above, must necessarily have a large neighborhood. Moreover,

$$\sum_{\{t \,|\, S_t \subseteq U\}} |N(S_t)| \geq |N(U)| \geq \frac{n}{3}. \tag{10}$$

That is, the neighborhood sizes of the constituent shards sum to be relatively large. Finally, the number of such disjoint shard groupings one can form is

$$\begin{aligned} \left\lfloor \frac{T}{\lceil \lceil 6/p \rceil / M \rceil} \right\rfloor &\geq \frac{T}{(6/p+1)/M+1} - 1 \\ &= \frac{np}{6+p+Mp} - 1 \\ &\geq \frac{np}{7+\lceil 6/p \rceil p} - 1 \\ &\geq \frac{np}{14} - 1. \end{aligned}$$

Thus, by (10),

$$\text{cost}(\mathcal{Q}) \geq \frac{np}{42} - 1 \geq \frac{np}{50}$$

for n sufficiently large. □

As shown in Theorem 4, Erdos-Renyi graphs do not contain any subtle structure that permits efficient sharding. Using this fact, Theorem 6 completes our analytic results by demonstrating that SBMs are optimally partitioned by assigning nodes in a block to the same shard. For simplicity we assume the blocks, instead of having binomial size, are of fixed size n/K, and moreover, that the SBM is undirected.

Theorem 6. *Suppose $\theta_+(n) \geq \theta_-(n) \geq (6 \log n)/n$. Then for every $\epsilon > 0$, $\exists N$ such that for $n \geq N$, $K \geq 2$ an integer, G an undirected SBM graph with blocks of size exactly n/K, and positive integers T, M with $M \geq n/K$ and $TM = n$, the cost of an optimal (G, T, M)-query plan $\mathcal{Q}$ satisfies*

$$c_0 \min(T, n\theta_-) \leq \textit{cost}_{opt}(\mathcal{Q}) \leq c_1 \min(T, n\theta_-) \tag{11}$$

with probability $1 - \epsilon$, where $c_0 = 1/50$ and $c_1 = 3$.

Proof. Without loss of generality, we assume that nodes are numbered so that ones in the same community are adjacent. For the upper bound, consider the sharding $\mathcal{S}$ in which the first M nodes are placed into the first shard, the second set of M nodes are put into the second shard, and so on. Since communities, which have size n/K, are by assumption smaller than the shards, this sharding results in each community being spread over at most two shards. Consequently, for the unique compatible query plan $\mathcal{Q}$,

$$\text{cost}(\mathcal{Q}) \leq 2 + \frac{1}{n} \sum_{i=1}^{n} |N_-(i)| \tag{12}$$

where $N_-(i)$ is the set of i's neighbors outside of its block. To bound the sum in (12), note that

$$\mathbb{E}\left[\sum_{i=1}^{n} N_-(i)\right] = n^2 \frac{K-1}{K} \theta_-,$$

since each of the $n^2(K-1)/2K$ inter-community edges is present with probability θ_- and appears in the sum once from the perspective of each endpoint. Since the probability of each edge being present is independent of the rest, an application of Chernoff bounds allows us to conclude that:

$$\mathbb{P}\left(\frac{1}{n} \sum_{i=1}^{n} |N_-(i)| \leq 2n\theta_-\right) \leq \left(\frac{e}{4}\right)^{n^2 \frac{K-1}{K} \theta_-} \leq \left(\frac{e}{4}\right)^{n \log n}$$

Thus, for n sufficiently large, $\text{cost}(\mathcal{Q})$ is bounded by $3n\theta_-$ with probability at least $1 - \epsilon/2$. To complete the upper bound, we note that trivially $\text{cost}(\mathcal{Q}) \leq T$.

For the lower bound, starting with G define the (random) graph $\hat{G}$ by independently deleting each intra-community edge with probability $1 - \theta_-/\theta_+$. Let $\mathcal{Q}$ and $\hat{\mathcal{Q}}$ denote optimal query plans for G and $\hat{G}$, respectively, Since we only delete edges in constructing $\hat{G}$,

$$\text{cost}(\mathcal{Q}) \geq \text{cost}(\hat{\mathcal{Q}}).$$

Moreover, any given intra-community edge is independently present in $\hat{G}$ with probability $\theta_+(\theta_-/\theta_+) = \theta_-$. Since inter-community edges are likewise independently present with probability θ_-, we have $\hat{G} \sim G(n, \theta_-)$. Consequently, by Theorem 4, for n sufficiently large,

$$\begin{aligned} &\mathbb{P}\left(\text{cost}(\mathcal{Q}) \geq c_0 \min(T, n\theta_-)\right) \\ &\quad \geq \mathbb{P}\left(\text{cost}(\hat{\mathcal{Q}}) \geq c_0 \min(T, n\theta_-)\right) \\ &\quad \geq 1 - \frac{\epsilon}{2} \end{aligned}$$

which completes the proof. □

The proceeding analysis suggests that when sharding real-world networks, one should map tightly knit communities to shards. In practice, however, this task is complicated by the fact the we observe only the network, not the community assignments. Below we show that finding such a sharding is NP-hard and present an exact IP formulation for small-scale solutions. In light of this difficulty we present an approximate method for efficient sharding in Section 5.

4. OPTIMAL SHARDING

We show that finding an optimal solution to the NetworkSharding problem is NP-complete by a reduction from the 3-Partition problem. We complement this analysis with an Integer Programming formulation, which can be used to solve small-scale versions of the problem. Unfortunately, we find the linear program defined by relaxing the integrality constraint of the IP results in a trivial solution, and is thus not fruitful in developing an approximation algorithm.

Theorem 7. *The* NetworkSharding *problem is NP-complete.*

Proof. The proof is by reduction from the 3-partition problem [5]: Given a set A of $3T$ elements, a bound $B \in \mathbb{Z}^+$, and an integral size w_a for each element $a \in A$ with ${}^{B}\!/_{4} \leq w_a \leq {}^{B}\!/_{2}$, find a partition of A into disjoint sets $A_1, \ldots, A_T$

such that for each $1 \leq i \leq T$, $\sum_{a \in A_i} w_a = B$. Note that the sets A_i are necessarily of size 3 by the constraints on the weights w_a. We assume the sum of the weights B is bounded by a polynomial in T—the problem remains hard under this constraint.

From any instance of the 3-partition problem, consider the following instance of the NetworkSharding problem. We define the graph G as follows. With every element $a \in A$ we associate a clique $G_a = (V_a, E_a)$ on $|V_a| = w_a$ nodes. Then let the network G be the union of the individual cliques, $G = (\cup_a V_a, \cup_a E_a)$; and set the number of shards to T, each with capacity B.

We claim that there is a 3-partition of A if and only if the cost of the optimal sharding is exactly 1. Suppose first that there is a 3-partition of A into $A_1, \ldots, A_T$ as above. Then a valid sharding is obtained by setting $S_t = \cup_{a \in A_t} V_a$. It is easy to see that $|S_t| = \sum_{a \in A_t} |V_a| = \sum_{a \in A_t} w_a = B$, and hence the capacity constraint is satisfied. Moreover, the sharding cost is 1, since all of the neighbors of a node are stored on the same partition.

Conversely, suppose there is a sharding of G with cost 1. It must be the case that each clique G_i is assigned fully to one shard, since any edge with endpoints across two shards would increase the total cost beyond 1. Thus we can set $A_t = \{a_j : G_j \in S_t\}$. The capacity constraints imply that $\sum_{a \in A_j} w_a = B$ and thus $A_1, A_2, \ldots, A_t$ form a valid 3-partition of A. $\square$

4.1 Integer Programming Formulation

Below we give the Integer Programming (IP) formulation of the NetworkSharding problem. To do so, we introduce several indicator variables. Let $\ell_{i,t}$ indicate whether the t-th shard is accessed when querying the i-th node. Then we can express the cost objective as:

$$\frac{1}{n} \sum_{i=1}^{n} \sum_{t=1}^{T} \ell_{it}.$$

Let e_{ijt} be the variable denoting whether node $j \in \{i\} \cup N(i)$ is looked up on shard t when node i is queried. In the IP formulation we insist that $e_{ijt} \in \{0, 1\}$. We define the consistency constraint, ensuring that every neighbor is looked up on exactly one shard as:

$$\sum_{t=1}^{T} e_{ijt} = 1 \qquad \forall i, j \in N(i) \cup \{i\}.$$

Further, to make sure that ℓ_{it} correctly counts the number of shards accessed, we have:

$$\ell_{it} \geq e_{ijt} \qquad \forall j \in N(i) \cup \{i\}.$$

Finally, let r_{it} indicate whether the i-th node is placed on the t-th shard. Since we can only look up a node on shard t if it is in fact stored there, we have

$$r_{jt} \geq e_{ijt} \qquad \forall i, j \in N(i) \cup \{i\}.$$

The last constraint guarantees that the sharding does not violate the memory constraints of the system. Let M be the total amount of memory available on each shard, then:

$$\sum_{i=1}^{n} r_{it} \leq M \qquad \forall t.$$

Putting these constraints together, we have the full formulation:

$$\begin{aligned}
\text{Minimize} \quad & \frac{1}{n} \sum_{i=1}^{n} \sum_{t=1}^{T} \ell_{it} \\
\text{Subject to:} \quad & \sum_{t=1}^{T} e_{ijt} = 1, \quad && \forall i, j \in N(i) \cup \{i\} \\
& \ell_{it} \geq e_{ijt}, && \forall i, j \in N(i) \cup \{i\}, t \\
& r_{jt} \geq e_{ijt}, && \forall i, j \in N(i) \cup \{i\}, t \\
& \sum_{i=1}^{n} r_{it} \leq M, && \forall t \\
& e_{ijt} \in \{0, 1\}, && \forall i, j \in N(i) \cup \{i\}, t
\end{aligned}$$

Although Integer Programs such as the one above can be efficiently solved for small networks, this approach does not obviously scale to networks with millions of nodes, even if the total number of shards is small.

The standard approach is to then translate the Integer Program into a Linear Program (LP). We accomplish this by relaxing the integrality constraint on e_{ijt} to:

$$e_{ijt} \geq 0, \quad \forall i, j, t.$$

However, the resulting LP has a trivial solution. It is easy to check that setting $e_{ijt} = 1/T$ for all i, j, t satisfies all of the constraints and leads to the optimal sharding cost of 1.

5. APPROXIMATE SHARDING

In this section we develop a scalable, two-stage method to approximately minimize the cost of sharding a given network. In the first stage we adopt a Bayesian approach to identifying block structure in the network [6], employing variational methods for approximate inference. In the second stage we map these blocks to shards such that the sharding cost is kept low while respecting memory constraints on the shards. We then present a simple method for replicating nodes to further reduce the sharding cost and, importantly, to reduce hotspots among the shards.

5.1 Block inference

While there is a large literature on the task of community detection, few of the developed methods scale to networks with millions of nodes and billions of edges. One of the more popular methods that does accommodate networks of this scale is that of label propagation [15], wherein nodes iteratively update their community assignments based on a majority vote over their neighbors' memberships. In this section, we extend past work on community detectionto scale to networks with billions of edges, showing that conventional label propagation is a limiting case of a more general framework, namely max-product variational inference for stochastic block models. The result is a scalable variant of label propagation which takes a discounted vote over neighbor memberships to compute updates. We review the work presented in Hofman and Wiggins [6] below before discussing this previously unknown connection, and present the result in Algorithm 1.

The block model presented in Section 3 provides a means of generating random networks with local structure (i.e., blocks) $\vec{z}$, given known parameters $\vec{\theta}$ and $\vec{\pi}$. Likewise, it allows us to calculate the likelihood of observing a given network and its block assignments under this model. Let $B_k = \{i \mid z_i = k\}$ denote the set of nodes in the k-th block and recall that $E(B_k, B_l)$ is the set of directed edges from block B_k to block B_l. The likelihood can be then be written as:

$$p(\mathbf{A}, \vec{z}|\vec{\theta}, \vec{\pi}) = \theta_+^{m_{++}}(1-\theta_+)^{m_{+-}}\theta_-^{m_{-+}}(1-\theta_-)^{m_{--}}\prod_{k=1}^{K}\pi_k^{n_k}, \quad (13)$$

where $m_{++} = \sum_{k=1}^{K}|E(B_k, B_k)|$ is the number of edges contained within blocks, $m_{+-} = \sum_{k=1}^{K} n_k(n_k-1) - m_{++}$ is the number of non-edges contained within blocks, $m_{-+} = m - m_{++}$ is the number of edges between different blocks, $m_{--} = n(n-1) - m - m_{-+}$ is the number of non-edges between different blocks, and $n_k = |B_k|$ is the number of nodes in the k-th block. In the case of sharding a network, however, a complementary scenario emerges: we observe a network $\mathbf{A}$ and would like to infer block assignments $\vec{z}$ (and possibly the parameters $\vec{\theta}$ and $\vec{\pi}$).

In principle, then, one can solve the block inference problem by assuming a prior on the model parameters and computing the posterior distribution over block assignments, placing nodes in their most probable blocks. Using Bayes' rule, the posterior over block assignments and parameters given the observed network is

$$p(\vec{z}, \vec{\theta}, \vec{\pi}|\mathbf{A}) = \frac{p(\mathbf{A}, \vec{z}|\vec{\theta}, \vec{\pi})p(\vec{\theta}, \vec{\pi})}{p(\mathbf{A})}, \quad (14)$$

where $p(\vec{\theta}, \vec{\pi})$ quantifies our beliefs about parameter values prior to observing the network, and

$$p(\mathbf{A}) = \sum_{\vec{z}}\int d\vec{\theta}\int d\vec{\pi} p(\mathbf{A}, \vec{z}|\vec{\theta}, \vec{\pi})p(\vec{\theta}, \vec{\pi}) \quad (15)$$

is the marginal likelihood of observing the network. The Appendix fully describes our choice of conjugate priors over $\vec{\theta}$ and $\vec{\pi}$—namely Beta and Dirichlet distributions, respectively—such that the integrals in the marginal likelihood are easily calculated. Unfortunately, however, the sum over all possible block assignments $\vec{z}$ in Equation (15) is fully coupled, containing K^n terms and rendering exact calculation of the posterior intractable for even moderately sized networks.

This difficulty motivates the variational mean-field approach to approximating the posterior presented in Hofman and Wiggins [6], a method that performs well in a number of inference tasks [2]. Variational methods provide an approximation to intractable sums such as Equation (15) by rephrasing the problem as one of optimization over a constrained family of posterior distributions. Stated more precisely, we search for a fully factorized distribution that is close to the true posterior under the Kullback-Leibler (KL) divergence, an (asymmetric) measure on distributions developed in the information theory literature [4]. That is, we look for a fully factorized distribution of the form

$$q(\vec{z}, \vec{\pi}, \vec{\theta}) = \left[\prod_{i=1}^{N} q_{z_i}(z_i)\right] q_{\vec{\theta}}(\vec{\theta})q_{\vec{\pi}}(\vec{\pi}) \quad (16)$$

that minimizes $D_{KL}\left[q(\vec{z}, \vec{\pi}, \vec{\theta})||p(\vec{z}, \vec{\theta}, \vec{\pi}|\mathbf{A})\right]$, where the divergence $D_{KL}[q||p] \equiv \sum_x q(x)\ln\frac{q(x)}{p(x)}$ is non-negative and zero if and only if q and p are identical.

While we can neither calculate nor minimize this divergence directly—as it is a functional of the unknown posterior—we can rewrite the divergence as follows:

$$D_{KL}\left[q(\vec{z}, \vec{\pi}, \vec{\theta})||p(\vec{z}, \vec{\theta}, \vec{\pi}|\mathbf{A})\right] = F_{\mathbf{A}}[q] + \ln p(\mathbf{A}), \quad (17)$$

where

$$F_{\mathbf{A}}[q] = -\sum_{\vec{z}}\int d\vec{\theta}\int d\vec{\pi} q(\vec{z}, \vec{\pi}, \vec{\theta})\frac{\ln p(\mathbf{A}, \vec{z}|\vec{\theta}, \vec{\pi})p(\vec{\theta}, \vec{\pi})}{q(\vec{z}, \vec{\pi}, \vec{\theta})} \quad (18)$$

is termed the variational free energy. Equation (17) is easily verified by substituting the free energy into the right hand side, combining integrands, and employing Bayes' rule. As a result of this identity, minimization of the free energy $F_{\mathbf{A}}[q]$ is equivalent to minimization of the KL divergence, as the two differ by the problem-specific constant $\ln p(\mathbf{A})$.

Furthermore, the sum in the mean-field variational free energy $F_{\mathbf{A}}[q]$ decouples yielding a tractable objective function, owing to the factorized form of Equation (16). To see this, recall the form of the likelihood from Equation (13) and examine the network-dependent term in the integrand. Substituting the likelihood and taking the logarithm reveals that the free energy is linear in the expected edge counts m_{++}, m_{+-}, m_{-+}, and m_{--} under the approximate posterior $q(\vec{z}, \vec{\pi}, \vec{\theta})$. As this distribution is fully factorized, the resulting expectations are pairwise over node assignments, and thus inexpensive to compute.

The task of exactly calculating the posterior $p(\vec{z}, \vec{\theta}, \vec{\pi}|\mathbf{A})$ can be replaced by that of finding a close approximate posterior $q(\vec{z}, \vec{\pi}, \vec{\theta})$ that minimizes the free energy $F_{\mathbf{A}}[q]$, and thus the KL divergence. Minimization of the free energy is achieved by iterative coordinate descent over the factors in Equation (16), a procedure guaranteed to find a local optimum of $F_{\mathbf{A}}[q]$. The resulting algorithm—termed variational Bayes—is an extension of expectation-maximization (EM), involving iterative updates of the variational distributions over parameters and block assignments until convergence in the free energy.

Upon convergence, variational Bayes provides an approximate distribution over block assignments for each node, from which we can assign each node to its most probable block. Unfortunately in the large-scale network setting of interest here we cannot directly apply variational Bayes, as simply storing these distributions in memory is prohibitive—for example, with 10 million nodes and 10,000 blocks, the required storage exceeds that of most commodity machines. To scale variational inference to networks of this size and complete the connection with traditional label propagation, we dispense with storage of the full distributions $q_{z_i}(z_i)$ and instead calculate only the most probable node assignment $\hat{z}_i$ for each node, known as max-product inference [2].

This modification, while relatively simple, admits a novel and intuitive reinterpretation of approximate Bayesian inference for stochastic block models as a framework for generalized label propagation. This previously unobserved result, detailed in Algorithm 1, proceeds as follows: in the first step, each node takes a discounted vote over its neighbors' block assignments to determine an updated block assignment; and in the second, the weights J, J' and $\vec{h}$ determining these discounts are updated via expectations over the variational distributions on $\vec{\theta}$ and $\vec{\pi}$. As shown in the Appendix, these expectations are easily calculated as functions of the estimated edge counts within and between blocks (m_{++} and m_{-+}) and the block sizes $\vec{n}$.

Algorithm 1 VBLabelProp

```
1: Randomly initialize the block assignments $\vec{z}$
2: repeat
3:    Based on the current block assignments, compute weights $J$, $J'$, and $h$, which are functions of the inferred edge/node counts $m_{++}, m_{-+}$, and $\vec{n}$:
      $J \leftarrow \mathbb{E}_{q_{\vec{\theta}}}\left[\ln \frac{\theta_+(1-\theta_-)}{\theta_-(1-\theta_+)}\right]$
      $J' \leftarrow \mathbb{E}_{q_{\vec{\theta}}}\left[\ln \frac{1-\theta_-}{1-\theta_+}\right]$
      $\vec{h} \leftarrow \mathbb{E}_{q_{\vec{\pi}}}\left[\ln \frac{1}{\vec{\pi}}\right]$
4:    Update each node's community assignment by a discounted vote over its neighbors assignments:
5:    for $i = 1 \to N$ do
6:       $z_i \leftarrow \underset{k}{\operatorname{argmax}}\ Jw_k - J'(n_k - \delta_{z_i,k}) - h_k$
         where $w_k = |E(i, B_k)| + |E(B_k, i)|$ is the total number of in and out edges between the $i$-th node and block $B_k$
7:    end for
8: until convergence in $F_{\mathbf{A}}[q]$
```

In the limiting case where blocks are assumed to be of uniform size (i.e., $n_k = n/K$), no discounting is applied and we recover conventional label propagation. The algorithm, termed VBLabelProp, has a low memory footprint—linear in the number of nodes and edges in the network—and its runtime scales linearly with the number of edges. We provide further details in the Appendix.

5.2 Partitioning Blocks

VBLabelProp provides a scalable method for inferring block assignments in large-scale networks to identify tightly knit groups of nodes. As discussed in Section 3, by storing these nodes on the same shard we greatly reduce the number of shards accessed when querying a node's neighbors. In practice, however, constructing this mapping is non-trivial, as the inferred sizes of the largest blocks in social network data can substantially exceed the available capacity of any single machine. Moreover, while Theorem 6 shows that SBMs have no second order structure one can exploit for sharding, in real-world networks we expect there is some gain to be had by intelligently placing well-connected blocks on the same shard. As such, we present BlockShard, a greedy method for zero-replication partitioning of blocks to shards that maintains a low sharding cost. We defer the discussion of replicating nodes to Section 5.3.

The inputs to BlockShard include the adjacency matrix $\mathbf{A}$, the blocks $\mathcal{B}$ produced by VBLabelProp, and the maximum shard capacity M. The output is a mapping from nodes to shards, with each node appearing only on a single shard, which consequently induces a unique compatible query plan.

The intuition behind BlockShard is to greedily map blocks to shards to minimize the current sharding cost. We start by allocating the largest block to the first shard and, if necessary, overflowing nodes to adjacent shards, filling each to capacity before moving to the next. Subsequent blocks are greedily selected to have highest average in-degree from the current, partially filled shard—or if the current shard is empty, the largest unallocated block is chosen. This condition may be interpreted as selecting the most "popular"

Algorithm 2 BlockShard

```
1: Input $\mathbf{A}, \mathcal{B}, M, \mathcal{S}$
2: Sort blocks by decreasing size $|B_k|$
3: Start with the first shard: $S_t \leftarrow S_1$
4: while unallocated nodes do
5:    if current shard $S_t$ is empty then
6:       Select the largest unallocated block $B_k$
7:    else
8:       Select the block with the largest in-degree from shard $S_t$: $B_k \leftarrow \underset{B}{\operatorname{argmax}} \frac{1}{|B|}|E(S_t, B)|$
9:    end if
10:   Sort nodes $i \in B_k$ by decreasing in-degree $|E(S_t, i)|$ from shard $S_t$
11:   Place nodes in shards starting with $S_t$, overflowing to adjacent shards as needed
12:   Update $S_t$ to the shard containing the last node allocated
13: end while
```

block from the perspective of nodes currently allocated to the shard. If the block size exceeds remaining shard capacity, nodes within the selected block are ordered by their in-degree from the current shard and accordingly overflowed to adjacent shards.

Pseudocode for BlockShard is given in Algorithm 2. An efficient implementation is achieved by maintaining an updatable heap over nodes, keyed by their in-degrees from the current shard. Runtime is then linear in the number of edges in the network, as desired for sharding large networks.

5.3 Replicating Nodes

When the total system capacity $M \cdot T$ exceeds the number of nodes n, utilizing excess memory to replicate data offers the potential benefits of decreasing the cost of reads and improving the system's overall fault tolerance. We quantify the amount of excess memory in the system via the replication ratio $\rho = TM/n \geq 1$, and present a simple method to add replication to any valid partitioning of nodes across shards—such as that produced by BlockShard.

One reasonable approach to replication is to simply copy the globally most accessed nodes to the excess space on each shard. While providing some improvement in performance, we opt for a variant of this strategy that considers the local structure of each shard, filling excess memory with "locally popular" nodes on each shard. In particular, when using BlockShard to partition nodes, we first run the algorithm with an artificially lower shard capacity $M' = \lceil n/T \rceil$ to obtain a partial sharding. Then for each partially filled shard S_t, we rank nodes by their in-degree $|E(S_t, i)|$ from the shard, filling it with the most accessed nodes. Pseudocode for the replication scheme, NodeRep, is given in Algorithm 3.

Algorithm 3 NodeRep

```
1: Input $\mathbf{A}, \mathcal{S}, M$
2: for shard $S_t \in S$ do
3:    while current shard $S_t$ is not full do
4:       Select a node current not in $S_t$ with the largest in-degree $|E(S_t, i)|$ from shard $S_t$
5:       Allocate the selected node to shard $S_t$
6:    end while
7: end for
```

To construct a valid query plan $\mathcal{Q}$ from the output of NodeRep, we avoid solving the costly set cover problem and instead build an index as follows. Consider the initial partitioning of nodes across shards (i.e., before replication) to define a node's primary shard. When executing a neighborhood query for node i, as many neighbors as possible are retrieved from i's primary shard, with the remaining ones fetched from their respective primary shards. As we show below, this method of replication is particularly effective for heavily skewed degree distributions and achieves a substantial reduction in both average load and hotspots among shards.

6. EMPIRICAL EVALUATION

We evaluate the performance of the approximate block sharding method developed above on two large-scale social network datasets. We show that network-aware shardings provide a reduction in average system load over both random sharding and sharding based on geography. We examine the effect of replication on system performance, showing that even a small degree of node duplication offers large savings in cost for both random and network-aware shardings. We conclude with a discussion of the variance in load over shards introduced by network sharding.

6.1 Data

We examine two online social networks appropriate for the NetworkSharding problem, as both contain a large number of nodes and edges, and require queries over node neighborhoods. Specifically, we look at the blogging community LiveJournal and the microblog platform Twitter, in which individuals frequently access activity feeds constructed from their local neighborhood of contacts.

LiveJournal. Our first real-world dataset comes from a 2004 snapshot of LiveJournal [11], a free online community that allows members to maintain and share journals and blogs. The full network contains over 5.1 million nodes connected by more than 150 million directed edges that specify for each user a set of "friends" whose blogs or journals she "follows". In addition to the network structure, the dataset contains a subset of city, state, and country information for 3.1 million users. We restrict our attention to the network induced by the 1.6 million users for whom all three of these fields are specified.

Twitter. The second network we investigate is a publicly available snapshot of the Twitter social graph [10]. The snapshot, completed in July of 2009, contains more than 41 million user profiles and 1.4 billion directed edges connecting these profiles. Users post short status updates and subscribe to—or "follow"—other users to receive their updates.

Both networks contain a giant (connected) component comprising more than 95% of all nodes in their respective networks. In representing the following relationships in these networks, we use the convention that an edge from i to j indicates that i follows j. In both networks, constructing a user's feed requires a query over recent activity from that user and all of her out-neighbors.

Other than obvious differences in scale, a significant distinction between the LiveJournal and Twitter networks is the skew in their degree distributions. In particular, the Twitter community contains a small number of highly popular "celebrities" with millions of followers, whereas the most popular profiles on LiveJournal have fewer than twenty thousand subscribers. This difference will become relevant when examining the effects of node replication on system performance.

6.2 Results

We examine the performance of various sharding schemes on the LiveJournal and Twitter networks. For LiveJournal, we compare random sharding and geographic sharding to BlockShard using both VBLabelProp and METIS to infer block structure. We repeat this analysis for the Twitter network, omitting geographic sharding and METIS due to lack of location information for users and prohibitive space constraints given the size of the network, respectively. For geographic sharding, we key each user by a concatenation of their country, state, and city, sort users by this key, and vertically fill shards in this order.

For both networks we assume that each profile requires roughly 1MB of in-memory space and that shard storage is on the order of 40GB of RAM, leaving a capacity of roughly 40k nodes per shard. As a result, the LiveJournal network requires 50 shards, and Twitter requires 1000, at zero replication ($\rho = 1.0$). We set the number of blocks for VBLabelProp to ten times the number of shards. All methods are evaluated in terms of average load and load balance, both with and without replication.

6.2.1 Average load

We first examine the average load, or equivalently, the sharding cost in the zero replication setting ($\rho = 1.0$). Figure 1 demonstrates that network-aware sharding offers a substantial reduction in cost over random sharding as well an improvement over geographic sharding. For example, randomly sharding LiveJournal results in an average access of 7 shards per query. While geographic sharding nearly halves this cost, network-aware shardings offer further improvements with VBLabelProp and METIS accessing 3.2 and 2.6 shards on average, respectively. Likewise, randomly sharding Twitter results in the average access of 26 shards per query, while VBLabelProp accesses 9 shards on average. In other words, network-aware shardings result in an approximately 60% reduction in average system load over Random. The observed difference between the two network-aware sharding methods for LiveJournal is likely due to the relatively balanced block sizes obtained by METIS. We would expect similar results for Twitter, however the memory requirements for METIS are prohibitive given 32GB of RAM.

Replication. Next we investigate performance when excess storage is used to accommodate popular nodes replicated by NodeRep. We vary the replication ratio ρ from 1.0 (zero replication) to 1.45 (45% replication) for all experiments and examine the relative and absolute changes in cost. Note that we consider "horizontal scaling," wherein an increase in ρ corresponds to an increase in the number of shards $T = T_0\rho$, while holding the capacity per shard M fixed.

Figure 1 shows the change in sharding cost for both LiveJournal (center) and Twitter (right) as we vary the replication ratio, with the points on the far left corresponding to the zero replication ($\rho = 1.0$) results in the left chart of Figure 1. Notably, a small amount of replication in random sharding (1%) for Twitter results in a large reduction in sharding cost (23%), which quickly asymptotes as replica-

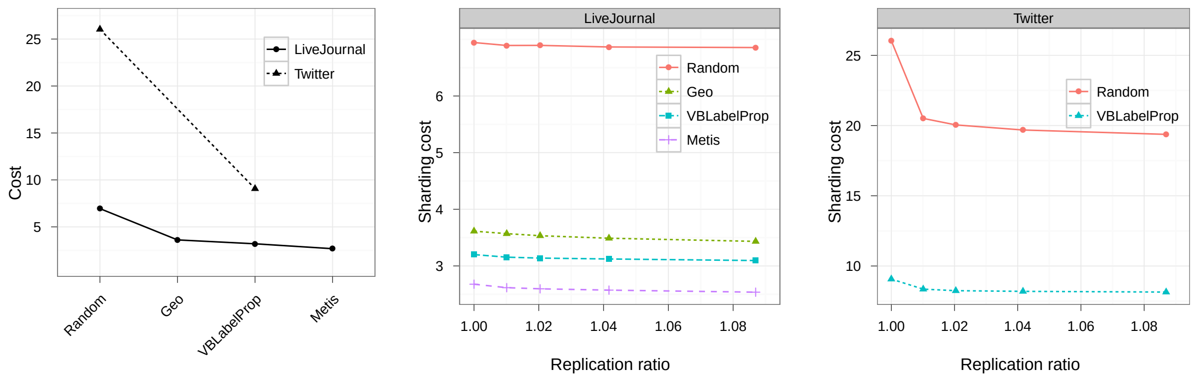

Figure 1: Sharding cost across methods. The left panel shows a comparison across methods for both networks at zero replication. The center and right panels show variation in cost as replication is added to the system for LiveJournal and Twitter, respectively.

tion increases further. That is, assigning even a small number of local celebrities to shards results in substantial gains. In contrast, random sharding for LiveJournal admits a relatively modest boost from replication. Intuitively these differences are due to the degree of skew in the two networks—while there is a relatively small population of highly popular celebrity accounts in Twitter, LiveJournal has a less heavy-tailed in-degree distribution.

Geographic and network-aware shardings also benefit from node replication, although to a lesser extent than Random, as these methods have already captured a substantial amount of network structure. Although omitted in the figure, as replication is increased beyond the point of $\rho = 1.09$, performance degrades for both methods: with a fixed shard capacity, increasing ρ effectively reduces the primary storage available on each shard, resulting in fragmentation of local structure and an increase in cost. Thus, a certain degree of replication improves performance across methods, replication provides diminishing returns from 1% ($\rho = 1.01$) to 9% ($\rho = 1.09$), after which replication tends to slightly degrade system performance.

6.2.2 *Load Balance*

In addition to evaluating system performance in terms of mean load, we now examine load balancing—or expected distribution of load across shards—for the various sharding techniques. Recalling Section 2.1, we calculate the load on an individual shard via the sum of rates over all nodes that access that shard, i.e. $L_t = \sum_i L_{i,t}$. Intuitively this captures the expected number of nodes that access shard S_t per unit time. Calculating these rates exactly would require knowledge of user logins and accesses to activity feeds—information which unfortunately is not provided in either data set. Thus we approximate these rates by assuming a uniform query rate across users.[1]

As we saw above, network-aware sharding more than halves the mean load across shards compared to random sharding. However, as shown in Figure 2, this reduction in mean load comes at an increase in variance over random sharding. Specifically, we compute the standard deviation over shards normalized by the mean load—termed the "load dispersion"—and plot the results for zero replication in the left panel of Figure 2.

As expected, we see that random sharding has the lowest relative variance across shards followed by geographic sharding. Further details of the cumulative load distribution across shards at zero replication are given for LiveJournal and Twitter in the center and right panels of Figure 2. Increased variance notwithstanding, we see that under network-aware sharding almost all shards (greater than 95%) experience less load compared to random and geographic sharding.

The observed skew under Random is, once again, explained by examining the role of celebrities in the system. Twitter contains an exceedingly small number of profiles (< 100) with hundreds of thousands of followers, after which profile popularity rapidly decreases, with tens of millions of profiles followed by at most a few users. Thus the small fraction ($\approx 10\%$) of shards which host these celebrities receive a disproportionate amount of traffic, accounting for the observed variance.

Replication. Node replication mitigates the load imbalance noted above (Figure 3). As revealed in the difference between the left (LiveJournal) and right (Twitter) panels, a small degree of replication (1%) drastically reduces variance in load for Twitter across both sharding methods, whereas LiveJournal sees more modest improvements. Again, this difference is due to the high degree of skew in popularity on Twitter, where we see that even random sharding experiences a reasonable degree of variance in per-shard load. Similar to the effect of replication on mean load, we see diminishing returns of increasing replication on load balance.

In summary, while the improvement in mean load achieved by network-aware sharding comes at the expense of a slight degradation in load balance, introducing a small degree of replication largely compensates for this effect.

7. DISCUSSION

We have formally defined the NetworkSharding problem and shown that considering network structure significantly improves system performance. Our results hold both in theory, where we show that network-aware sharding results in substantial savings over random sharding for networks generated using an SBM; and in practice, for both the LiveJournal and Twitter networks, where network-aware

[1]Load results are indistinguishable under alternate approximations to query rate, including degree-correlated rates ($\lambda_i \propto \log d_i$) and rates proportional to post volume (i.e. number of tweets for Twitter profiles).

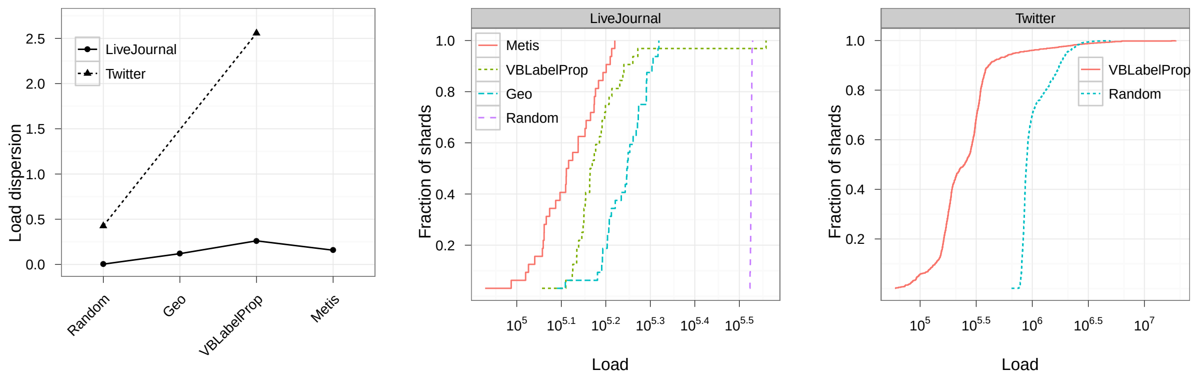

Figure 2: Variation in and distribution of load across shards for zero replication. The left panel summarizes standard deviation in load, normalized by mean load, across methods for both networks. The center and right panels show the full cumulative distributions of per-shard load for LiveJournal and Twitter, respectively.

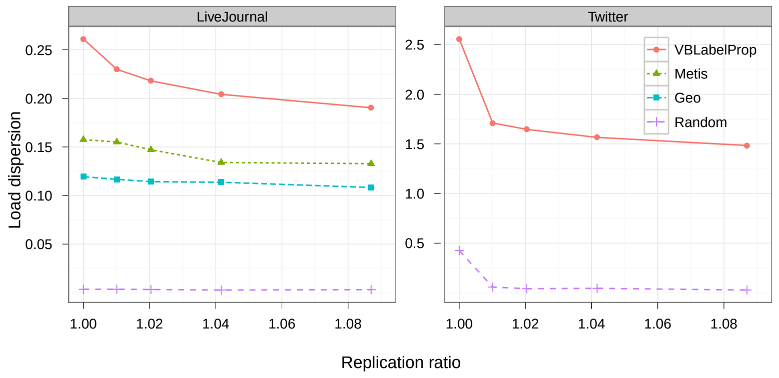

Figure 3: Effect of replication on load balance. The left and right panels show the variation in load dispersion as replication is added to the system for LiveJournal and Twitter, respectively.

sharding more than halves the average load. Moreover, we find that allowing a small amount of replication further reduces mean load while improving load balance.

Many interesting questions remain. Experiments show that the decrease in average load was accompanied by an increase in variance, and that, for a handful of shards, load increases under network-aware sharding. We leave open the problem of formulating sharding strategies that improve the load for *all* of the shards. Further possible improvements include online implementations capable of updating sharding assignments as the network evolves.

8. REFERENCES

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *Seventh USENIX Conference on Networked Systems Design and Implementation*, pages 2–2, 2010.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[3] B. Bollobás. *Random Graphs*, volume 73. Cambridge University Press, 2001.

[4] T. Cover and J. Thomas. *Elements of Information Theory*, volume 6. Wiley Online Library, 1991.

[5] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, NY, USA, 1979.

[6] J. M. Hofman and C. H. Wiggins. Bayesian approach to network modularity. *Physical Review Letters*, 100:258701, Jun 2008.

[7] P. Holland. Local structure in social networks. *Sociological Methodology*, 7:1–45, 1976.

[8] T. Karagiannis, C. Gkantsidis, D. Narayanan, and A. Rowstron. Hermes: clustering users in large-scale e-mail services. In *First ACM Symposium on Cloud Computing*, pages 89–100, 2010.

[9] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–13, Washington, DC, USA, 1998.

[10] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Nineteenth ACM International Conference on World Wide Web*, pages 591–600, 2010.

[11] D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins. Geographic routing in social networks. *Proceedings of the National Academy of Sciences*, 102(33):11623–11628, Aug. 2005.

[12] A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 348–355, New York, NY, USA, 2006. ACM.

[13] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Inf. Retr.*, 10:205–231, June 2007.

[14] 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 *ACM SIGCOMM Conference*, SIGCOMM '10, pages 375–386. ACM, 2010.

[15] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76:036106, Sep 2007.

APPENDIX

We provide details of the VBLabelProp algorithm omitted from Section 5.1 for brevity. We specify explicit functional forms for the variational free energy, the distributions over parameters, and the expectations with respect to these distributions.

First, we note that the variational free energy in Equation (18) can be written as the sum of three terms, all expectations with respect to the variational distribution $q(\vec{z}, \vec{\pi}, \vec{\theta})$:

$$\begin{aligned} F_{\mathbf{A}}[q] = & - \mathbb{E}_q\left[\ln p(\mathbf{A}, \vec{z}|\vec{\theta}, \vec{\pi})\right] - \mathbb{E}_q\left[\ln p(\vec{\theta}, \vec{\pi})\right] \\ & - \mathbb{E}_q\left[\ln q(\vec{\theta}, \vec{\pi})\right], \end{aligned}$$

where the first term is the expected complete-data likelihood, the second is the cross-entropy between the prior $p(\vec{\theta}, \vec{\pi})$ and $q(\vec{z}, \vec{\pi}, \vec{\theta})$, and the third is the entropy of the variational distribution itself. Taking the logarithm of Equation (13), expressing all edge counts in terms of m_{++} and sums over n_k, and collecting common terms, we can write the complete-data likelihood as

$$\begin{aligned} \ln p(\mathbf{A}, \vec{z}|\vec{\theta}, \vec{\pi}) \;=\; & J m_{++} - J' \sum_{k=1}^{K} n_k(n_k - 1) - \sum_{k=1}^{K} h_k n_k \\ & + \; m \ln \theta_- + [n(n-1) - m] \ln (1 - \theta_-) \end{aligned}$$

where we have defined the weights

$$\begin{aligned} J &= \ln \frac{\theta_+(1-\theta_-)}{\theta_-(1-\theta_+)} \\ J' &= \ln \frac{1-\theta_-}{1-\theta_+} \\ h_k &= \ln \frac{1}{\pi_k} \end{aligned}$$

to be positive for the assortative case ($\theta_+ > \theta_-$). Intuitively we interpret these as follows: J weights the local term involving edges within blocks, while J' and h_k balance this term by global terms based on the number of possible edges within blocks and block sizes, respectively.

As mentioned in Section 5.1, we assume priors which are conjugate to the specified likelihood terms—namely Beta distributions over θ_+ and θ_-, and a Dirichlet distribution over $\vec{\pi}$. Specifically, the Beta distribution with hyperparameters α and β is given by

$$p(\theta) = \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}$$

while the Dirichlet distribution with hyperparameters $\vec{\alpha}$ is given by

$$p(\vec{\pi}) = \frac{1}{\Gamma(\vec{\alpha})} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1},$$

where $B(\alpha, \beta)$ and $\Gamma(\vec{\alpha})$ are the beta and gamma special functions, respectively. As these conjugate prior distributions are in the same family as the resulting posteriors, calculating posteriors against their respective likelihoods involves simple algebraic updates—rather than potentially costly numerical integration—which may be interpreted as adding counts (from observed data) to pseudo-counts (specified by hyperparameter values) to determine updated hyperparameter values.

Likewise, expectations under these distributions have relatively simple functional forms. Specifically, when calculating the weights given above, we have expectations of log-parameter values, given by the following:

$$\mathbb{E}_{\vec{\theta} \sim Beta(\alpha, \beta)}\left[\ln \theta\right] = \psi(\alpha) - \psi(\alpha + \beta)$$

and

$$\mathbb{E}_{\vec{\pi} \sim Dir(\vec{\alpha})}\left[\ln \frac{1}{\pi_k}\right] = \psi(\alpha_k) - \psi(\sum_{k=1}^{K} \alpha_k),$$

where $\psi(x)$ is the digamma function. Using the above, then, we have the following values for the expected weights J, J', and $\vec{h}$, as abbreviated in Algorithm 1 for VBLabelProp:

$$\begin{aligned} \mathbb{E}_{q_{\vec{\theta}}}\left[J\right] &= \psi(m_{++} + \alpha_+) - \psi(m_{+-} + \beta_+) \\ & - \; \psi(m_{+-} + \alpha_-) + \psi(m_{--} + \beta_-) \\ \mathbb{E}_{q_{\vec{\theta}}}\left[J'\right] &= \psi(m_{--} + \beta_-) - \psi(m_{-+} + \alpha_- + m_{--} + \beta_-) \\ & - \; \psi(m_{+-} + \beta_+) + \psi(m_{++} + \alpha_+ + m_{+-} + \beta_+) \\ \mathbb{E}_{q_{\vec{\pi}}}\left[h_k\right] &= \psi(n_k + \alpha_k) - \psi(\sum_{k=1}^{K} \alpha_k), \end{aligned}$$

where α_+ and β_+ are the hyperparameters for the prior on θ_+; α_- and β_- are the hyperparameters for the prior on θ_-; and $\vec{\alpha}$ are the hyperparameters for the prior on $\vec{\pi}$.