Case study: Scoop for partial read from P2P database

Farnoush Banaei-Kashani and Cyrus Shahabi

Abstract In this paper we propose *Scoop*, a mechanism to implement the "partial read operation" for peer-to-peer databases. A peer-to-peer database is a database that its relations are horizontally fragmented and distributed among the nodes of a peer-to-peer network. The partial read operation is a data retrieval operation required for approximate query processing in peer-to-peer databases. A partial read operation answers to β-queries: given $\beta \in [0,1]$ and a relation R, a fraction β of the tuples in R must be retrieved from the database to answer a β-query. Despite the simplicity of the β-query, due to the distributed, evolving and autonomous nature of the peer-to-peer databases correct and efficient implementation of the partial read operation is challenging. Scoop is designed based on an epidemic dissemination algorithm. We model the epidemic dissemination as a percolation problem and by rigorous percolation analysis tune Scoop per-query and on-the-fly to answer β-queries correctly and efficiently. We prove the correctness of Scoop by theoretical analysis, and verify the efficiency of Scoop in terms of query cost and query time via extensive simulation.

Farnoush Banaei-Kashani
University of Southern California, Department of Computer Science, Los Angeles, CA 90089-0781 e-mail: banaeika@usc.edu

Cyrus Shahabi
University of Southern California, Department of Computer Science, Los Angeles, CA 90089-0781 e-mail: shahabi@usc.edu

1 Introduction

1.1 Motivation and Problem Definition

A peer-to-peer database is a database that its relations are horizontally fragmented and distributed among the nodes of a peer-to-peer network. Each node is potentially both a client and a server of the database. Peer-to-peer databases are distinguished from the more generic distributed databases by the sheer size of the database in terms of the participating nodes, as well as higher dynamism of the data, nodes, and the interconnection network. Correspondingly, these distinctive features necessitate provision of more scalable and dynamic data management mechanisms for peer-to-peer databases (as compared to those for the generic distributed databases). Peer-to-peer databases lie at the intersection of the database and networking research fields, and currently numerous researchers and focus groups from both communities are actively working to provide the required data management capabilities for this new generation of the distributed databases[1].

Here, we focus on unstructured peer-to-peer databases. With these databases, the process of query answering is comparable to that in regular unindexed databases. Due to the considerable amount of churn in unstructured peer-to-peer databases, construction and maintenance of distributed index structures (e.g., a DHT [24, 30]) are not cost-efficient. Without index, to answer a user query, first the query is disseminated throughout the network to scan the database and retrieve the data relevant to the query (analogous to the sequential scan in regular unindexed databases). Subsequently, the retrieved data are processed locally at the querying node to answer the user query[2]. In this paper, we introduce a basic data retrieval operation, termed the partial read operation, to execute the first step of the query answering, specifically as required for *approximate* query answering discussed below.

Approximate query answering is a desirable generalization of exact query answering, arguably essential for efficient query answering in unstructured peer-to-peer databases. With these unindexable databases, returning exact answers (i.e., the complete result-set) requires a complete scan of the entire database for each query. Considering the large size of the peer-to-peer databases and the abundance of the database users with numerous queries, answering all queries with exact result is obviously too costly to scale. Besides, with most of the peer-to-peer database applications the workload mainly consists of exploratory queries for which exact results are unnecessary and redundant [11]. With approximate query answering, queries are assumed to be satisfied by an approximate/partial result, which is a subset of the complete result-set of the query. Along with the query, users specify the required "completeness" of the approximate result that satisfies the query. Correspondingly,

[1] For instance, see the publications at the recurrent DBISP2P workshop (held in conjunction with VLDB) and the NetDB workshop (held in conjunction with ICDE).

[2] Here, by "query" we refer to user queries such as SQL queries. In this paper, we also use "query" to refer to β*-queries*, i.e., requests for retrieving the tuples of the base relations stored in the database, as we later define formally. The intended meaning is apparent from the context.

the database system can reduce the overhead of the data retrieval and query processing to the least sufficient to satisfy the query, effectively enhancing the efficiency of the query answering by eliminating the redundancy of the result. If required, still a query can be answered exactly. An exact query is simply a specific case of approximate query requiring the most complete result-set, the exact result.

Formally, an approximate query is defined as an arbitrary query with a user-defined *completeness ratio* $\varepsilon \in [0,1]$; any fraction ε of the complete result-set X, noted as $X^{(\varepsilon)}$, is sufficient to satisfy the approximate query. An exact query is an approximate query with $\varepsilon = 1$. Approximate queries are also called *partial* query. To answer a partial query, first the peer-to-peer database should be sampled to retrieve a certain fraction β_i (also called the completeness ratio, in lack of other conveying terms) from each input relation R_i of the query. Subsequently, using partial query processing techniques such as [23, 31] the retrieved data are processed locally at the querying node to generate the final partial result-set $X^{(\varepsilon)}$. The main challenge with answering partial queries is with devising a basic sampling operation that given any $\beta \in [0,1]$, retrieves a fraction β of an input relation R (i.e., retrieving a partial relation noted as $R^{(\beta)}$), where R is horizontally fragmented and distributed among the nodes of the peer-to-peer database. We term such a sampling operation the *partial read operation*. Partial read is a nonprobability sampling operation, because it is not expected to guarantee randomness of the sample retrieved from the database. It is informative to note that with *structured* peer-to-peer databases, since the placement of the data objects is a priori known, partial read is trivial to implement. In this paper, we focus on implementing the partial read operation in *unstructured* peer-to-peer databases.

To be applicable, the partial read operation must be both correct and efficient. A partial read is correct if $|R'|/|R| \geq \beta$, where R' is the retrieved portion of the input relation R. This is a sufficient condition for correctness of the partial read. However, ideally the redundancy of the data retrieval is minimal when $|R'|/|R| = |R^{(\beta)}|/|R| = \beta$. A peer-to-peer database is a fragmented database which is randomly distributed among a variable set of nodes with unknown population. Under such circumstances, to be correct the partial read operation must implement a distributed query dissemination scheme to visit sufficient number of nodes and retrieve sufficient number of tuples, such that in expectation the collective set of the retrieved tuples R' satisfies the correctness condition with high probability. On the other hand, the efficiency of the partial read operation is defined in terms of the query time (or sampling time) and query cost (or sampling cost), which respectively map to the total communication time and communication cost of disseminating the query for distributed data retrieval. There is a trade-off between the query time and the query cost, with higher cost to achieve shorter time and vice versa. An efficient partial read operation satisfies the ideal case of the correctness condition with a balanced query time versus query cost. Scoop is a mechanism to implement the partial read operation correctly and efficiently.

1.2 Related Work

Data retrieval from networks has been previously studied as a search problem, but none of the proposed search mechanisms can satisfy the correctness and efficiency requirements of the partial read operation. There are two main proposals for search in unstructured peer-to-peer networks: search by flooding [18, 33] and search by random walk [19, 1, 20, 6, 10]. With both of these search mechanisms, query is disseminated throughout the network by recursive forwarding from node to node. With flooding each node that receives the query forwards it to all of its neighbors, whereas with random walk query is forwarded to only one (uniformly or non-uniformly) selected random neighbor. None of these approaches can strike a balance between the two metrics of efficiency for search, i.e., the communication time and the communication cost. Flooding is most efficient in communication time but incurs too high of redundant communication to be practical, whereas random walk is potentially more efficient in communication cost but is intolerably slow in scanning the network. In [33], a two-tier hierarchy is proposed where flooding is restricted to the supernodes at the top tier. This solution only alleviates the communication cost of flooding and the problem resurfaces as the top tier scales. In [19], using k random walkers in parallel is proposed as a way to balance the communication cost and the communication time of the query. However, this proposal does not provide any rigorous solution for selecting the value of k for optimal performance.

Previous search mechanisms are not only inefficient, but also inappropriate for executing partial read. With Top-K queries, the relevant data is ordered in terms of a preference measure, whereas partial read allows unbiased data retrieval. Also, as mentioned above the main benefit of the approximate query answering is that it allows trading off completeness of the result for better efficiency by limiting the scan of the database to a just sufficiently large fraction of the database that satisfies the β-query. To enable such a trade-off, a search mechanism that is used to implement the partial read should allow adjusting the coverage of the database (i.e., the fraction of the network nodes, and hence data objects, visited during dissemination) according to the user specified parameter β of each query. With both flooding and random walk, TTL (Time-To-Live) is the control parameter that can be used to limit the coverage of the network. However, it is not clear how one can adjust TTL according to β for sufficient coverage (where the size of the network is also unknown). TTL is often set to a fixed value, a value that is selected in an ad hoc fashion based on the average performance of the typical search queries and must be re-adjusted as the peer-to-peer database evolves. Alternatively, TTL is gradually increased to expand the coverage, each time repeating the query dissemination from the beginning, until sufficient fraction of the database is covered to answer the query. Although this scheme may result in *correct* partial read, due to the redundancy of repeating the query dissemination, query cost can even exceed that of the regular flooding. Finally, another problem specific to the flooding is that the granularity of the coverage is too coarse (the number of covered nodes grow exponentially with TTL), rendering fine adjustment of the coverage impossible.

1.3 Scoop

1.3.1 Overview

We propose *Scoop* as a sampling mechanism to implement partial read. To sample the database, beginning from the originator of the query Scoop spreads the query throughout the network. While spreading, the query inspects the nodes of the network to locate and retrieve a fraction β of the tuples of the input relation R. With Scoop, the spread of the query is modelled on epidemic dissemination of diseases in social networks[3]. By epidemic dissemination, query spreading is probabilistic, i.e., when a node receives a query, it replicates and forwards the query to each of its neighbors independently with a forwarding probability $0 \leq p \leq 1$. Therefore, a node forwards the replicas of the query to zero or more neighbors at each time. Such a query forwarding algorithm is obviously more flexible as compared to both flooding and random walk and subsumes these dissemination mechanisms. The *communication graph* of the epidemic dissemination (i.e., the subgraph of the peer-to-peer network covered by the dissemination) is sparse with small values of p. The communication graph grows larger and denser with larger values of p such that with $p = 1$ the epidemic dissemination is equivalent to regular flooding which covers the entire network.

Scoop specifically implements SIR (Susceptible-Infected-Removed), which is a classic epidemic dissemination model [12]. Our main contribution with Scoop is derivation of a closed-form formula that given a partial read request, maps the value of the completeness ratio β to an appropriate value for the forwarding probability p such that the request is correctly satisfied. Leveraging on this derivation, Scoop *on-the-fly* and *per read-request* tunes p based on β such that the communication graph grows just sufficiently large to cover a fraction of the database that satisfies the partial read request. For partial read requests with small β the communication graph is sparse and as β increases the graph becomes denser. Since both the communication cost and communication time of the sampling increase proportional to the size of the communication graph, the partial read requests with higher completeness ratio are more expensive, as expected.

1.3.2 Correctness and Efficiency

In Section 3.3, we rigorously prove that Scoop satisfies the ideal case of the correctness condition for the partial read operation. Scoop achieves correctness without a priori knowledge about the size of the network. With each value of p Scoop covers a certain fixed fraction of the network nodes *independent* of the size of the network. In other words, for the same value of p, size of the communication graph is always pro-

[3] In the literature, sometimes *gossip-based* or *rumor-based* spreading techniques are also termed epidemic techniques [7, 15, 16, 5]. Here, we are not referring to such many-to-many communication techniques, but specifically to the techniques that are modelled after disease spreading in social networks.

portional to the size of the entire network, such that its relative size (i.e., the covered fraction of the network) is fixed. This property allows answering β-queries (which expect retrieving a fixed fraction of a relation) without acquiring and maintaining global knowledge about the current size of the network. Intuitively, this is feasible because unlike flooding and random walk with which a query never dies unless it is explicitly terminated (e.g., when TTL expires), with epidemic dissemination query forwarding is probabilistic and with some non-zero probability each replica of the query may naturally die at each step. The dissemination terminates whenever all replicas of the query die. The larger the network, the more it takes for the dissemination to die and, therefore proportionally, the larger is the communication graph of the dissemination.

Scoop is also efficient, in that it strikes a balance between the communication cost and communication time of the sampling. Since epidemic dissemination is essentially a flood-based technique, as we show in Section 6, its communication time is low and comparable with that of the regular flooding. On the other hand, due to the phase transition phenomenon associated with the SIR epidemic model, for the common case of the partial read requests the communication cost of the Scoop is up to two orders of magnitude less than that of the regular flooding and comparable with the low communication cost of the random walk. Intuitively, with epidemic dissemination the dense communication graph of the regular flooding, which with numerous loops represents a large amount of redundant and duplicate query forwarding, is reduced to a sparse communication graph. With fewer loops, the sparse graph contains less redundant paths and therefore, causes less duplicate queries, while covering almost the same set of nodes. Hence, epidemic dissemination can be tuned such that the overhead of the flooding is effectively eliminated while its reachability and communication time is preserved.

It is also important to mention that Scoop is simple to implement, and since it is a randomized mechanism, it is inherently reliable to use with dynamic peer-to-peer databases. Moreover, with Scoop the larger the degree of a node, the larger its load. However, as Gribble et al. have shown in [26], with peer-to-peer databases the degree of connectivity of the nodes is positively correlated with the amount of the shared resources of the nodes. Therefore, with Scoop the load is distributed among the nodes roughly proportional to the resources of the nodes, which is a desirable load balancing property.

1.3.3 Originality

The process of epidemic disease dissemination has been previously used as a model to design other information dissemination techniques [13]. Particularly, in the networking community, epidemic dissemination is termed probabilistic flooding and is applied for search and routing in various types of networks [17, 8, 27]. We distinguish Scoop from this body of work in two ways. First, although epidemic algorithms are simple to implement, due to their randomized and distributed nature they are often difficult to analyze theoretically. For the same reason, most of the previous

work restrict themselves to empirical study of the performance with results that are subject to inaccuracy and lack of generality. We employ the percolation theory to rigorously tune Scoop to its best operating point. Second, to the best of our knowledge those of the few percolation-based theoretical studies of epidemic algorithms often adopt simplistic mathematical models [12] that assume a homogenous topology (a fully connected topology) for the underlying network to simplify the analysis. However, recently it is shown that considering the actual topology of the network in the analysis extensively affects the results of the analysis [9]. We perform our analysis of Scoop assuming an arbitrary random graph as the underlying topology of the peer-to-peer network and specifically derive final results for a power-law random graph, which is the observed topology for some peer-to-peer databases [26].

We originally described our preliminary ideas with Scoop in a poster paper [2]. This paper is merely a descriptive introduction to Scoop (equivalent to the introduction section of the current paper). The current paper extends the poster paper by including the Scoop algorithm, analytical details, and experimental results.

1.3.4 Experimental Results

We performed an empirical study via simulation to evaluate the efficiency of Scoop. In lack of other existing solutions, we compared Scoop with hypothetical partial read operations that use scope-limited flooding (i.e., flooding with limited TTL) and k-random-walkers (with various k) for query dissemination. As we explained in Section 1.2, these dissemination mechanisms are not originally appropriate for execution of the partial read requests and we had to artificially inform them about the coverage required to satisfy each request. Our results show that even under such artificial conditions, Scoop still outperforms scope-limited flooding in communication cost while maintaining a reasonable communication time. Also, to our surprise, Scoop not only has a much better communication time as compared to that of the random-walk but also outperforms a 32-random-walker (the optimal case as suggested in [19]) even in communication cost.

1.3.5 Summary of Contributions

To summarize, we enlist our contributions with Scoop as follows:

- Identifying and formulating a basic query type, i.e. β-query, as an essential type of sub-query for answering generic approximate queries in peer-to-peer databases.
- Developing an epidemic-based query answering mechanism to evaluate β-queries in unstructured peer-to-peer databases with arbitrary random graph topology. The novelty with this query answering mechanism is in introducing a generic and rigorous analytic approach based on percolation analysis to control the epidemic dissemination for both correct and efficient query answering.

- Evaluation of the proposed query answering mechanism by extensive empirical analysis via simulation.

It is important to note that although Scoop is applicable to the more generic distributed databases, it is particularly designed to accommodate the scalability and dynamism requirements of the peer-to-peer databases (as discussed above).

1.4 Roadmap

The remainder of this paper is organized as follows. In Section 2, we define the partial read problem with further details. Section 3 introduces the generic case of Scoop for partial read from peer-to-peer databases with arbitrary random graph topology. In Section 4, we specialize our results from Section 3 to develop a specific case of Scoop for the real-world peer-to-peer databases with power-law topology. In Section 5, we briefly discuss several variants of Scoop. Section 6 presents the results of our empirical study on Scoop. Finally, Section 7 concludes the paper and discusses the future directions of this research.

2 Partial Read Operation

Given $\beta \in [0,1]$, a partial read operation must retrieve a fraction β of the relation R, which is horizontally fragmented and distributed among the nodes of the network N. We term such a partial read request a β-query. It is important to note that the semantics of the β-query is less restrictive as compared to that of the stronger *top-k* query. While both these types of queries are useful and popular in peer-to-peer databases, they require distinct query answering mechanisms. To answer a β-query, beginning from the originator of the β-query (which is the same node where the

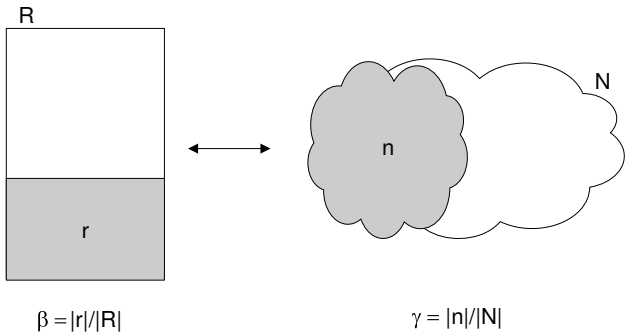


Fig. 1 Partial Read

user query is issued), query is disseminated through the network to visit a fraction γ of the nodes of the network which collectively store a sub-relation r such that $|r|/|R| = \beta$ (see Figure 1).

Regardless of the choice of the dissemination mechanism to implement the partial read operation, the main challenge is tuning the dissemination mechanism per β-query such that the query is correctly satisfied. Figure 2 depicts the parameter mapping process for tuning a generic dissemination mechanism. At Step I, considering the distribution of the tuples among the nodes, γ (i.e., the fractional size of the subset n of the nodes that should be visited to retrieve r) is calculated based on β. Subsequently, at Step II some parameter(s) of the dissemination mechanism (such as parameter p for the epidemic dissemination) is tuned based on γ such that the dissemination actually covers n nodes of the network. Specifics of Step II of the mapping process depend on the particular dissemination mechanism applied to implement the partial read.

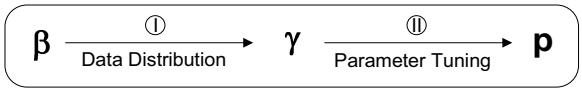


Fig. 2 Parameter Mapping Process for Tuning a Generic Dissemination Mechanism

With Scoop, we employ an epidemic dissemination mechanism to implement the partial read operation. We assume the tuples of the relation R are uniformly distributed among the nodes of the network (hence, the mapping at Step I will be trivial $\gamma = \beta$), and focus on the specifics of Step II for epidemic dissemination to derive p as a function of γ (or equivalently, β). Before describing Scoop and its tuning process, here we provide some definitions.

2.1 Definitions

2.1.1 Communication Graph

Communication graphs are used to represent and visualize query dissemination over networks. A network with the node-set N and link/edge-set E can be modelled as an undirected graph $G(N,E)$. For a query dissemination initiated at time $t = t_0$ on G, the *communication graph* at any time $t \geq t_0$ is a subgraph $G_t(N_t,E_t)$ of G, where $E_t \subseteq E$ is the set of links traversed by at least one query replica during the time interval $[t_0,t]$, and $N_t \subseteq N$ is the set of nodes visited by at least one query replica during the same time interval. Associated with any link e of G_t is a weight w_e that is the number of times the link e is traversed during the time interval $[t_0,t]$. We assume a discrete time model; thus, the dynamic process of disseminating a query is completely represented by the set of communication graphs $\{G_{t_0}, G_{t_0+1}, G_{t_0+2}, ..., G_{t_0+T}\}$, where at time

$t = t_0 + T$ the query dissemination is terminated (hence, for all $t \geq (t_0 + T)$, $G_{t_0+t} = G_{t_0+T}$). For example, Figure 3 depicts the 6 first communication graphs of a query that is initiated at node A and disseminated based on the random walk dissemination mechanism.

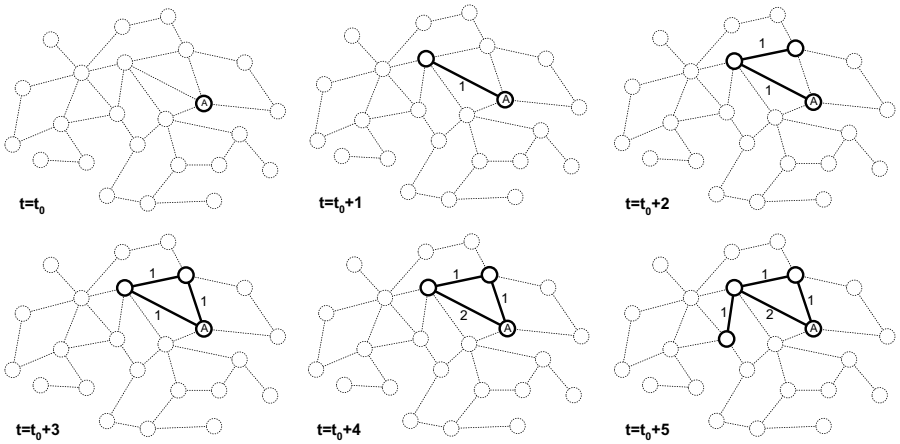


Fig. 3 Communication Graph

2.1.2 Efficiency Measures for Sampling

We define two metrics to measure the efficiency of query dissemination mechanisms used for sampling from networks:

1. *Query cost (or sampling cost, or communication cost)* **C**: Assuming uniform cost for traversing the links of the network and uniform query size, we model the communication cost of disseminating a query based on a particular dissemination mechanism as the total number of query replicas communicated between the nodes of the network during the entire process of query dissemination. In communication-graph terminology:

$$\mathbf{C} = \sum_{e \in E_{t_0+T}} w_e$$

2. *Query time (or sampling time, or communication time)* **T**: Assuming uniform latency for the network links, the sampling time is the total time it takes to disseminated the query. In communication-graph terminology:

$$\mathbf{T} = T$$

3 Scoop: Partial Read by Epidemic Dissemination

Epidemic dissemination is inspired by epidemic spreading of contagious diseases in social networks (i.e., networks that model a society, with nodes as people and links as social links between people who are in contact). A contagious disease first appears at one node (the originator of the disease), and then through the social links disseminates throughout the network in a flood-like fashion, from the infected person to its direct neighbors in the social network, from the infected neighbors to their neighbors, and so on. The success in transmission of the disease through a social link is probabilistic, i.e., the disease is transmitted from an infected node to its susceptible neighbor with probability p ($0 \leq p \leq 1$) and is ceased with probability $1-p$. The value of p depends on the infectiousness of the particular disease as well as some other environmental parameters, and with simple disease spreading models the value is generic to all links of the network. When the spreading terminates, the disease has covered/reached a sample h of the total node population H ($h \subseteq H$), where the relative size of h increases with increasing value of p (although not necessarily linearly).

With epidemic dissemination by Scoop, we model the query dissemination mechanism on the disease spreading process. By analogy, we take the dissemination of a query in a peer-to-peer database as the spreading of a disease in a social network. With this analogy, the infection probability p translates to the query forwarding probability, and the infected sample h maps to the sampled node-set n. Among various disease spreading models, we model epidemic dissemination on the *SIR (Susceptible-Infected-Removed)* disease spreading model. Below, first we describe the SIR disease spreading model, which readily defines our epidemic query dissemination mechanism with Scoop. Thereafter, for our SIR-based epidemic query dissemination mechanism we develop a percolation model to derive γ as a function of the query forwarding probability p for peer-to-peer databases with arbitrary random graph topology. The function $\gamma(p)$ is a one-to-one function. Therefore, in turn, it defines p as a function of γ, which enables Step II of the tuning process for Scoop.

3.1 SIR Epidemic Dissemination

With the SIR disease spreading model, at any particular time during dissemination of the disease (or equivalently, the query), each node of the network is in one of the three states susceptible (S), infected (I), or removed (R). A "susceptible" node is liable to infection but is not infected yet; an "infected" node is currently infected and is able to infect its neighbors; and a "removed" node has recovered from the infection and is both immune to further infection and impotent to infect other nodes. The discrete-time and dynamic process of SIR epidemic dissemination can be explained as follows. Initially, at time t_0 all nodes of the network are susceptible except the originator of the disease, which is infected. As the disease propagates throughout the network, if at time $t \geq t_0$ a susceptible node A has an infected neighbor B, at

time $t+1$ with probability p node A conceives the disease from B and becomes infected (see Figure 4 for the state transition diagram of a node). An infected node remains in the infectious state for a period of time τ (during which it is able to infect its neighbors), and then finally becomes removed. We assume $\tau = 1$ without loss of generality. The disease dissemination terminates (dies out) at time $t_0 + T$ (where $T \geq 1$) when all nodes of the network are either in the removed state (affected by the disease) or the susceptible state (survived the disease), and none of the nodes are in the infected state. By analogy, with the SIR-based epidemic query dissemination of Scoop, when the query dissemination terminates the set of the removed nodes is the set n of nodes visited by the query, and the set of susceptible nodes is the set $N \backslash n$ of the nodes not covered by the query dissemination (see Figure 1).

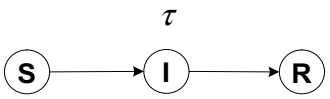


Fig. 4 State Diagram for SIR Epidemic Dissemination

3.2 Percolation Model

To tune the epidemic dissemination with Scoop, we need to answer two questions:

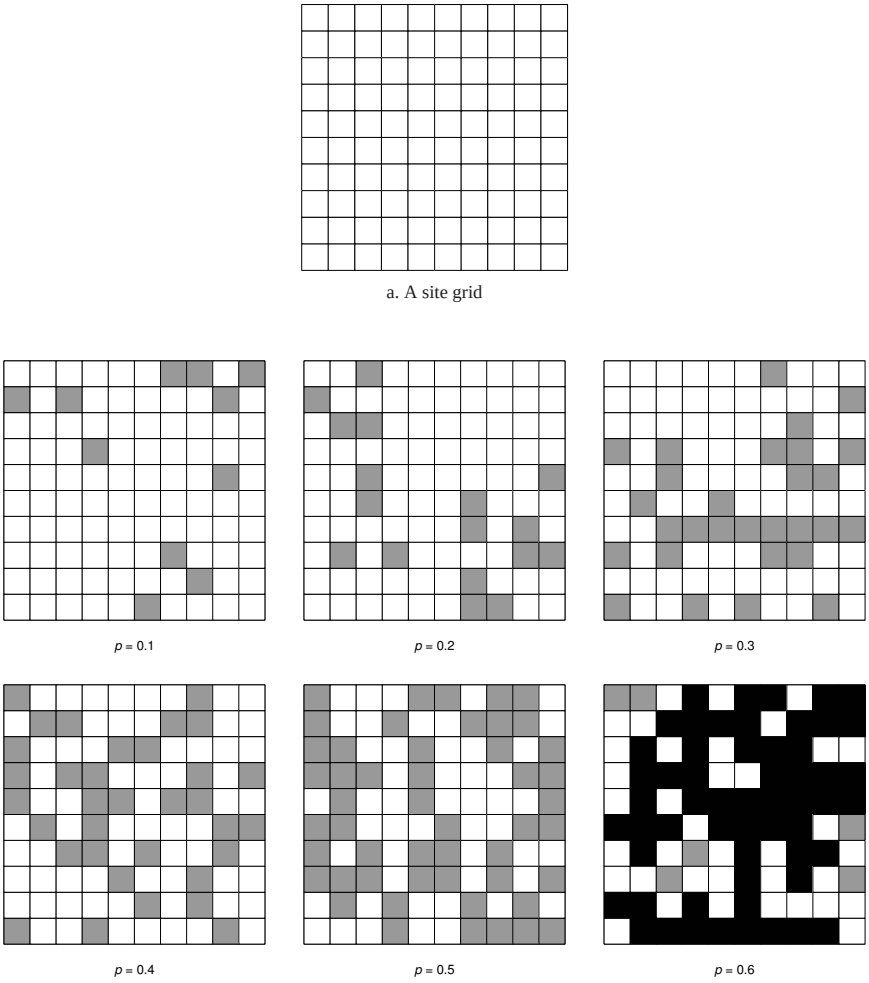
1. *How large p should be for the query dissemination to prevail a large network?* For a query to prevail a network N, we should have:

$$\lim_{|N|\to\infty} \frac{|n|}{|N|} = \gamma \tag{1}$$

   with $\gamma > 0$. In other words, the size of the covered node-set n must be comparable to the size of the entire network N, otherwise the partial read operation cannot satisfy any β-queries other than the trivial β-query with $\beta = 0$. With too small values of p, the dissemination dies out quickly, and the query fails to prevail and covers an infinitesimally small number of nodes as compared to the total number of nodes $|N|$ in large networks; i.e., we have:

$$\lim_{|N|\to\infty} \frac{|n|}{|N|} = 0$$

2. *How can we derive γ as a function of p?* γ is an increasing function of p. Having $\gamma(p)$, we can fulfill Step II of the tuning process (see Figure 2) by deriving $p(\gamma)$ as $\gamma^{-1}(p)$. In other words, given a β-query and assuming $\gamma = \beta$, we can tune the forwarding probability p of Scoop on-the-fly to satisfy the β-query.



a. A site grid

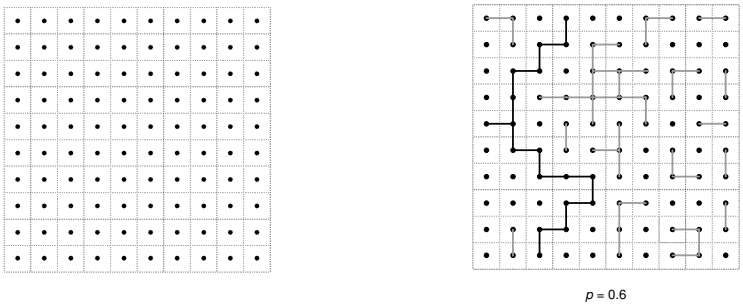
b. Site percolation instances

Fig. 5 Site Percolation Problem

To answer these questions, we model the epidemic dissemination process as a percolation problem. First, we illustrate this percolation problem by describing two toy percolation problems (Figures 5 and 6). Consider the grid in Figure 5-a. Each cell of the grid is called a site. Suppose we color each site of the grid independently with probability p. Figure 5-b depicts several instances of the colored grid with various probabilities. As p increases, larger clusters of colored sites appear. Particularly, at $p = 0.6$ there is a giant cluster (marked in black) that spans from the top side of the grid to the bottom side. When such a giant cluster appears in a colored grid, we say the grid percolates. It is proved [29, 21, 22] that there exists a critical probability

p_c (in this case, $p_c \approx 0.59$) such that whp[4] the grid percolates only if $p \geq p_c$. The size of the giant cluster depends on the value of p and as p increases the giant cluster grows such that at $p = 1$ the giant cluster covers the entire grid. The toy problem described above is called a site percolation problem on two dimensional grid.

Equivalently, one can define the dual bond percolation problem where a set of nodes are arranged into a grid (see Figure 6-a) with a bond (not shown) between every pair of adjacent nodes. Suppose each bond is colored independently with probability p. In this case a giant cluster is a cluster of nodes connected with colored bonds that percolates from one side to the other side of the grid (in Figure 6-b the giant cluster is marked in black).



a. A bond grid

b. Bond percolation instance

Fig. 6 Bond Percolation Problem

We model the epidemic dissemination process as a bond percolation problem on an arbitrary random graph. Our problem differs from the toy bond percolation problem described above in two ways. First, instead of a grid we assume an arbitrary random graph as the underlying bonding structure among the nodes. With an arbitrary random graph, each node is not restricted to a fixed number of neighbors (e.g., four neighbors as in two dimensional grid) but can have a random number of neighbors according to some arbitrary degree distribution. On the other hand, unlike a grid, a random graph is not delimited by sides. For a bond percolation problem on a random graph, a giant cluster is defined as a cluster of nodes (i.e., a set of nodes connected with colored bonds) where the size of the cluster is comparable to the size of the entire graph.

This bond percolation problem models the epidemic dissemination as follows: the underlying random graph represents the physical topology of the peer-to-peer network (i.e., the logical overlay); and a cluster generated with the coloring probability p, is an instance of the communication graph of a query if the query was initiated by a node within the cluster and disseminated with the forwarding probability p. With this model, we can answer the two questions raised before as follows:

[4] "whp" stands for "With High Probability".

1. *How large p should be for the query dissemination to prevail a large network?* The query prevails the network if and only if a giant cluster exists. Thus, whp query prevails the network if and only if $p \geq p_c$, where p_c is the critical percolation probability of the bond percolation problem.
2. *How can we derive γ as a function of p?* To derive $\gamma(p)$ we should derive the relative size of the giant cluster as a function of the coloring probability p, for all $p \geq p_c$.

Next, we solve the bond percolation problem for the critical probability p_c, and the size of the giant cluster as a function of p.

3.3 Tuning Scoop

3.3.1 Definitions

We use the generating-function formalism [32] to represent probability distribution functions. Particularly, the generating function $G_0(x)$ for the distribution of the node-degree k in an arbitrary random graph is defined as:

$$G_0(x) = \sum_{k=0}^{\infty} p_k x^k \tag{2}$$

where p_k is the probability that a randomly chosen node of the graph has degree k. From Equation (2), one can derive the n-th moment of the node degree distribution as follows:

$$\langle k^n \rangle = \left[(x\frac{d}{dx})^n G_0(x) \right]_{x=1} \tag{3}$$

Also, for a random graph represented by the generating function $G_0(x)$, we define $G_1(x)$, which is the generating function for the distribution of the degree of the nodes we arrive at by following a randomly chosen link from the graph. $G_1(x)$ depends on $G_0(x)$ and is derived as follows:

$$G_1(x) = \frac{1}{\langle k \rangle} G_0'(x) \tag{4}$$

3.3.2 Analysis

First, consider the bond percolation model described in Section 3.2. Suppose the coloring probability is p. A cluster of nodes connected by the colored bonds is itself a random graph embedded within the underlying random graph $G_0(x)$. It is easy to derive the generating function $G_0(x;p)$ for the degree distribution of the graphs representing the clusters based on $G_0(x)$:

$$G_0(x;p) = \sum_{m=0}^{\infty}\sum_{k=m}^{\infty} p_k \binom{k}{m} p^m (1-p)^{k-m} x^m \tag{5}$$

$$= \sum_{k=0}^{\infty} p_k \sum_{m=0}^{k} \binom{k}{m} (xp)^m (1-p)^{k-m} \tag{6}$$

$$= \sum_{k=0}^{\infty} p_k (1-p+xp)^k \tag{7}$$

$$= G_0(1+(x-1)p) \tag{8}$$

Similarly one can derive $G_1(x;p)$ as follows:

$$G_1(x;p) = G_1(1+(x-1)p) \tag{9}$$

Next, we derive the distribution of the size s (i.e., the number of nodes) of the clusters. Assume $H_0(x;p)$ is the generating function for the distribution of the size of the clusters. Observing that each cluster consists of a node connected to k other sub-clusters (where k is distributed according to $G_0(x;p)$), we derive the distribution of the cluster size by a recursive argument as follows:

$$H_0(x;p) = xG_0(H_1(x;p);p) \tag{10}$$

and similarly:

$$H_1(x;p) = xG_1(H_1(x;p);p) \tag{11}$$

From $H_0(x;p)$, we can also compute the average size $\langle s \rangle$ of the clusters using Equation (3):

$$\langle s \rangle = H_0'(1;p) = 1 + G_0'(1;p)H_1'(1;p) \tag{12}$$

However, according to Equation (11) we have:

$$H_1'(1;p) = 1 + G_1'(1;p)H_1'(1;p) = \frac{1}{1-G_1'(1;p)} \tag{13}$$

Thus:

$$\langle s \rangle = 1 + \frac{G_0'(1;p))}{1-G_1'(1;p)} = 1 + \frac{pG_0'(1)}{1-pG_1'(1)} \tag{14}$$

Now, since at the critical probability p_c the giant cluster appears, the average size of the clusters $\langle s \rangle$ goes to infinity at $p = p_c$; i.e.:

$$\langle s \rangle = 1 + \frac{p_c G_0'(1)}{1-p_c G_1'(1)} \to \infty \tag{15}$$

Therefore, the critical probability p_c can be computed as:

$$p_c = \frac{1}{G_1'(1)} = \frac{1}{\frac{\langle k^2 \rangle}{\langle k \rangle} - 1} \tag{16}$$

This concludes the solution for the first question raised in Section 3.2. To answer the second question, i.e., calculating the relative size of the giant cluster as a function of p, we observe that for $p \geq p_c$, $H_0(x;p)$ remains the distribution of the *finite* size clusters, i.e., all clusters except the giant cluster. Thus, we have:

$$H_0(1;p) = 1 - \gamma(p) \tag{17}$$

Using Equation (10) we can derive $\gamma(p)$ as follows:

$$\gamma(p) = 1 - G_0(y;p) \tag{18}$$

where according to Equation (11), $y = H_1(1;p)$ is the solution of:

$$y = G_1(y;p) \tag{19}$$

We solve these equations for $\gamma(p)$ numerically by iteration.

4 A Real-World Example of Scoop

In Section 3, we described the generic case of Scoop for the peer-to-peer databases with arbitrary random graph topology. As captured by empirical studies, some of the real-world peer-to-peer databases such as Gnutella [18] and Kazaa [28] have power-law random graph topologies [26, 25, 14, 11]. Here we specialize the generic case of Scoop for the peer-to-peer databases with power-law topology.

4.1 Network Topology

In this section, we assume the topology of the peer-to-peer database is a power-law (or scale-free) random graph, i.e., a random graph [4] with power-law probability distribution for node degrees. Intuitively, in a power-law random graph most of the nodes are of low degree while there are still a few nodes with very high connectivity. We define the power-law probability distribution function for the node degree k as follows:

$$p_k = Ck^{-\eta}e^{-k/\nu} \tag{20}$$

where η, ν, and C are constants. η is the skew factor of the power-law distribution, often in the range $2 < \eta < 3.75$ for real networks. For example, a case study reports $\eta = 2.3$ for Gnutella [26]. The less the skew factor, the heavier the tail of the power-

law distribution, which translates to larger number of highly connected nodes. A pure power-law distribution does not include the exponential cut-off factor ($e^{-k/\nu}$), allowing for nodes with infinite degree, which is unrealistic for real peer-to-peer databases. The cut-off factor with index ν shortens the heavy tail of the power-law distribution such that the maximum node degree for the nodes of the graph is in the same order of magnitude as ν. Finally, C is the normalization factor that is computed as $C = [\mathbf{Li}_\eta(e^{-1/\nu})]^{-1}$, where $\mathbf{Li}_\eta(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^\eta}$ is the η-th polylogarithm function of x.

4.2 Analysis

The generating function of the power-law degree distribution (see Equation 20) can be represented based on the polylogarithm function as follows:

$$G_0(x) = \frac{\mathbf{Li}_\eta(xe^{-1/\nu})}{\mathbf{Li}_\eta(e^{-1/\nu})} \tag{21}$$

From Equation (3), we can compute the first and the second moments of the power-law degree distribution[5]:

$$\langle k \rangle = (x\frac{d}{dx})G_0(x)\Big|_{x=1} = \frac{\mathbf{Li}_{\eta-1}(e^{-1/\nu})}{\mathbf{Li}_\eta(e^{-1/\nu})}$$
$$\langle k^2 \rangle = (x\frac{d}{dx})^2 G_0(x)\Big|_{x=1} = \frac{\mathbf{Li}_{\eta-2}(e^{-1/\nu})}{\mathbf{Li}_\eta(e^{-1/\nu})}$$

Consequently, from Equation (16) we can derive the critical probability p_c for a power-law graph as follows:

$$p_c = \frac{\mathbf{Li}_{\eta-1}(e^{-1/\nu})}{\mathbf{Li}_{\eta-2}(e^{-1/\nu}) - \mathbf{Li}_{\eta-1}(e^{-1/\nu})} \tag{22}$$

In Figure 7, we illustrate p_c as a function of ν for various η values in a real-world power-law peer-to-peer database, i.e., Gnutella. For Gnutella, the skew factor η is estimated as low as $\eta = 1.4$ and as high as $\eta = 2.3$, in different scenarios. Also, ν is in the range of 100 to 1000. As illustrated by this example, the critical probability p_c in power-law networks can be as low as 0.01.

We also solved Equation (18) to derive $\gamma(p)$ for power-law networks by numerical iteration. In Figure 8-a, we show our result for a power-law random graph with the skew factor $\eta = 2.3$.

[5] Note: $\frac{d}{dx}\mathbf{Li}_\eta(x) = \frac{1}{x}\mathbf{Li}_{\eta-1}(x)$

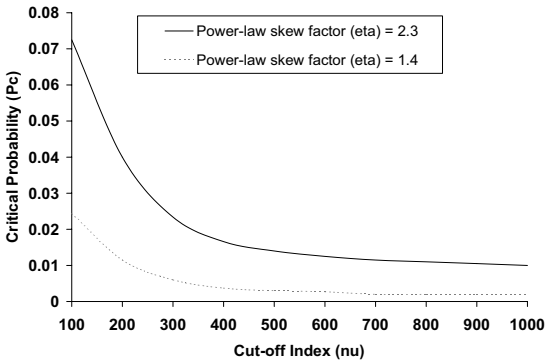


Fig. 7 Critical Probability in Power-Law Peer-to-Peer Databases

4.3 Algorithm

1. Seeding the Query (Phase I)
   To make sure the query dissemination is initiated by a node belonging to the giant cluster, the actual query originator initiates a selective walker to locate a highly connected node in the network. With the selective walk, at each hop the query is forwarded to a neighbor with the maximum connectivity degree until the walker reaches a node with a connectivity degree higher than the degree of all its neighbors. In [1], it is shown that in a power-law network whp such a selective walker finds a highly connected node belonging to the giant cluster in $O(\log N)$ hops, where N is the size of the network. Our experiments also verify this result.
2. Disseminating the Query (Phase II)
   Next, the SIR-based epidemic query dissemination is initiated at the highly connected node located by the selective walker. The query is disseminated with a forwarding probability $p > p_c$ selected according to Equation (18), such that $\gamma(p)$ satisfies the given β-query.

5 Variants of Scoop

Since in peer-to-peer databases some nodes may refrain from participating in query dissemination, here we introduce a family of variants for the basic case of Scoop to model this behavior. With this family, unlike the original SIR model where initially all nodes are in the "susceptible" state, some nodes begin in the "removed" state. In the context of the disease epidemic, these nodes represent the people who are vaccinated. We term such nodes the *blocked* nodes.

We consider three different variants with blocking for the basic Scoop, each representing a particular real-world scenario:

1. *Scoop with uniform blocking*: In this case, nodes are blocked with uniform probability. This case models the networks where nodes autonomously decide whether or not to participate in the query dissemination.
2. *Scoop with negative degree-correlated blocking*: In this case, the nodes with lower connectivity degrees are blocked with higher probability. For example, this case models the peer-to-peer file-sharing networks where low degree nodes are usually also low-bandwidth and therefore, to avoid congestion and possible isolation, may refrain from participating in query dissemination with higher probability.
3. *Scoop with positive degree-correlated blocking*: This case is opposite to the previous case, where nodes with higher connectivity degrees are more probably blocked. This case models the scenario where, for example, high degree nodes of a power-law network are attacked and disabled.

6 Experiments

We conducted two sets of experiments via simulation to 1) study the behavior of Scoop empirically, and 2) evaluate the efficiency of Scoop. For this purpose, we implemented a discrete-time event-driven simulator in C++. We used an Enterprise E220 SUN server to perform the experiments.

6.1 Methodology

With the first set of experiments, we studied the relation between the forwarding probability p and the size of the sample node-set covered by the Scoop query dissemination. Therefore, with these experiments data content of the nodes is irrelevant. With the second set of experiments, we evaluated the efficiency of various partial read operations in resolving β-queries. Our Monte Carlo simulation was organized as a set of "runs". For the first set of experiments each run consists of 1) selecting a network topology, 2) selecting a query originator, and finally 3) initiating 50 query disseminations per forwarding probability p (for each one of the partial read operations) while varying p from 0 to 1, and recording the average size of the covered node-set as well as **C** and **T**. For the second set of experiments a run comprises 1) selecting a network topology, 2) selecting an object-set (a multiset of tuples), 3) distributing the object-set among the network nodes, 4) selecting a query originator and finally 5) initiating the query for 50 times per β (for each of the partial read operations) while varying β from 0 to 1, and recording the average value of their efficiency numbers **C** and **T**. With this set of experiments, all issued queries are answered correctly by Scoop and we focus on reporting the efficiency of the query answering with Scoop. Each result data-point reported in Section 6.2 is the average result of 50 runs. The coefficient of variance across the runs was below 2.5% and

hence show the stability of the result. The high stability of the performance is an expected benefit of the randomization inherent in Scoop.

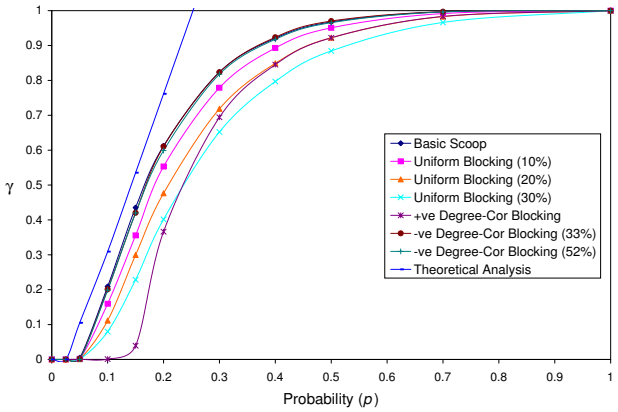
We generated a set of 100 undirected power-law graphs $G(N,E)$ each with $|N| = 50000$ and $|E| \approx 200000$. The skew factors of the graphs are all about $\eta = 2.3$ as measured in [26]. The minimum number of edges per node is 4, and the cut-off index of the graphs is $\nu = 100$. The graphs are generated using the preferential attachment model proposed by Barabasi et al. [3].

We considered a 5-dimensional content space and generated 100 object-sets. For each object-set, we first generated $|U| = 100000$ objects $u = \langle a_1, a_2, ..., a_5 \rangle$, where a_i is an integer value uniformly selected from the interval $[1,10]$. Thereafter, we replicated the objects according to the object replication scheme defined in Section 4.1 with the total number of objects $|R| = 500000$. R is uniformly distributed among the set of network nodes N.

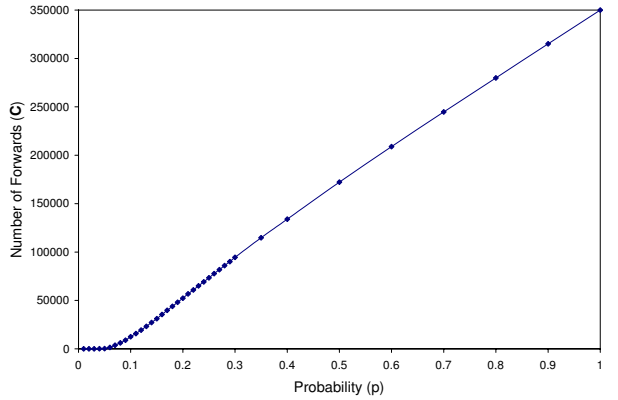
6.2 Results

Figure 8 illustrates the results of our first set of experiments. Figure 8-a depicts the relation between the query forwarding probability p and the relative size γ of the covered node-set n. First, we notice how close the results of our theoretical analysis conforms with the performance of the basic case of Scoop in practice, specially for our p values of interest close to the critical forwarding probability p_c. Also, we observe that while performance of the Scoop with negative degree-correlated blocking is almost identical to that of the basic Scoop (they overlap in the figure), with positive degree-correlated blocking, the coverage for the same forwarding probability decreases significantly. This shows 1) the importance of the highly connected nodes in the performance of Scoop, and 2) the independence of its performance from the nodes with lower connectivity degrees, which are often low-bandwidth and volatile. Figure 8-b confirms our previous conjecture that the sampling cost of Scoop is linearly proportional to the query forwarding probability. Also, (from Figures 8-a and 8-b) notice that with $p = 0.3$ almost 80% of the network nodes can be covered with only about 25% of the sampling cost of the regular flooding (with $p = 1$). Besides, the size of the covered node-set becomes sublinearly proportional to the network size starting at $p_c \approx 0.05$, where the sampling cost is almost two orders of magnitude less than that of the flooding. Finally, Figure 8-c illustrates how the sampling time reduces as the forwarding probability goes from p_c towards 1, because the giant cluster becomes more strongly connected. Also, we observe that in the worst case, the sampling time with Scoop only increases by a factor of 4 over the minimum possible sampling time with flooding.

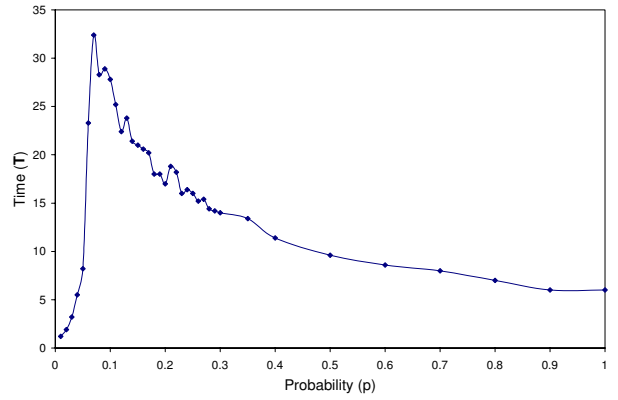
Figures 9 and 10 illustrate the results of our second set of experiments. With these experiments, we compared the efficiency of Scoop in answering β-queries with that of partial read operations based on random walk and scope-limited-flooding dissemination mechanisms. As we mentioned in Section 1.2, unlike Scoop, these two partial read operations are unable to determine whether they have covered a suffi-



a. Nodes sample size vs. forwarding probability

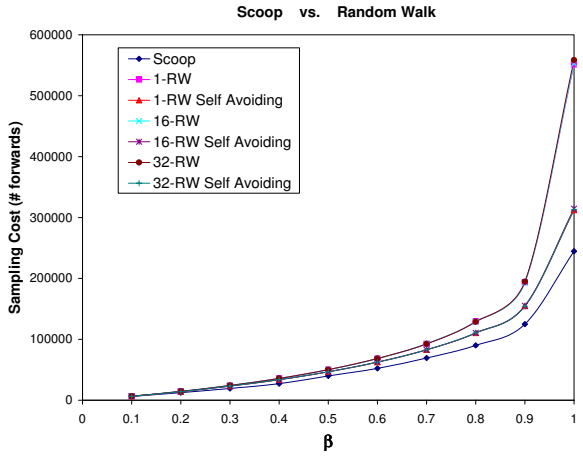


b. Sampling cost vs. forwarding probability

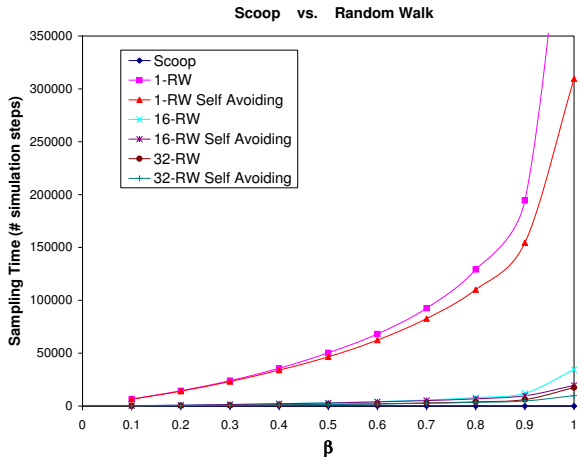


c. Sampling time vs. forwarding probability

Fig. 8 Verification of the Analytical Results



a. Sampling cost



b. Sampling time

Fig. 9 Scoop vs. Random Walk

ciently large fraction of the network to satisfy a particular β-query. Nevertheless, to be able to compare Scoop with these partial read operations, for each particular β-query with given β we calculated the absolute (not relative) number of nodes that must be covered to satisfy the query, and terminated the random walk and flooding as soon as their coverage exceed the required number of nodes. Scoop can decide on the required coverage on its own.

Figure 9-b shows that, as one can expect, the sampling time of Scoop is always incomparably shorter than that of the random walk, even with 32 parallel walkers (in the figure, the Scoop plot lies on the *x* axis). However, to our surprise, Scoop also

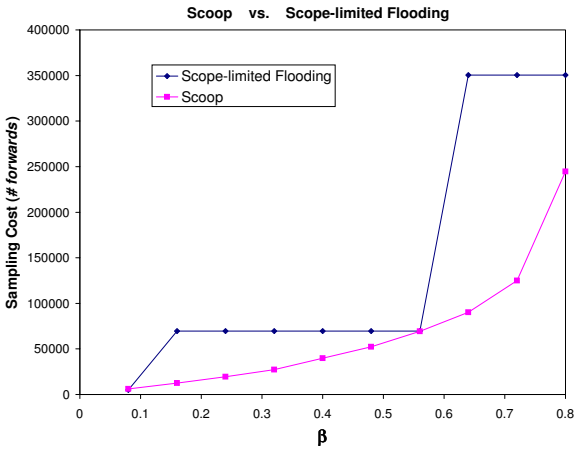
outperforms random walk in sampling cost (see Figure 9-a). In other words, to cover the same number of nodes the SIR-based dissemination uses a "lighter" communication graph with less query forwarding (see Section 2.1.1 for the definition of the communication graph) as compared to that of the random walk. This can be justified by considering the fact that multiple random walkers are traversing the topology with no (even implicit) inter-communication to avoid redundant traversal. However, with the SIR-based dissemination adopted by Scoop the more a part of the topology is explored the less the chance of it to be re-explored; hence, redundant traversal is reduced. Also note that as illustrated in Figure 9-a, the random walk algorithms with different number of walkers incur the same sampling cost. This should not be surprising; more walkers enhance the sampling time of the random walk by scanning the network in parallel, but a single random walker walks as much as 32 random walkers walk in aggregate to cover the same number of nodes. Finally, notice that among random walk algorithms, self-avoiding random walk (with which each random walker avoids its repeated paths) always outperforms regular random walk in sampling time.

Figure 10-a shows that Scoop always outperforms scope-limited flooding in sampling cost. Notice the step-like diagram for the scope-limited flooding. Since at each hop flooding scans an exponentially larger number of new nodes, unlike Scoop it cannot be tuned properly to cover a certain fraction of the network nodes with fine granularity. Finally, Figure 10-b shows that although the sampling time of Scoop always exceeds that of the flooding (which is optimal), even in the worst case it remains tolerable.

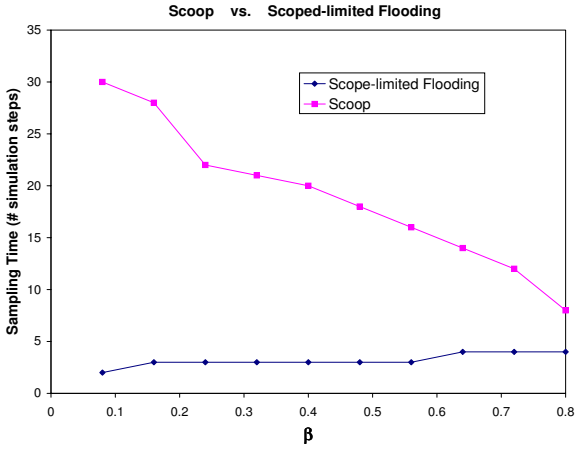
7 Conclusion and Future Work

In this paper, we proposed Scoop to answer β-queries. Scoop is easy to implement because each node of the peer-to-peer database can simply toss a biased coin with probability p to decide whether the query should be disseminated through a link or not. We showed through percolation analysis that the value of p can be rigorously computed for each β-query such that the query is correctly satisfied. Also, with a comparative empirical study we showed that Scoop outperforms flooding by up to two orders of magnitude in communication cost while maintaining a tolerable response-time. Also, as compared to a 32-random-walker, Scoop has not only faster response time but also less communication cost.

We intend to extend this study in three directions. First, since Scoop is essentially a flood-based dissemination technique with relatively short communication time (see Figure 8-c), considering the typical rate of churn in peer-to-peer databases we do not expect significant changes happening during a Scoop dissemination. Hence, we do not anticipate the performance of Scoop is noticeably affected by the typical rate of churn. However, for the peer-to-peer databases that may intrinsically have higher churn such that the time-scale of the dynamic changes in the peer-to-peer database is comparable to that of the query dissemination, we are planning to use



a. Sampling cost



b. Sampling time

Fig. 10 Scoop vs. Scope-Limited Flooding

dynamic percolation analysis to factor in the effect of the changes during the query dissemination. Second, we intend to explore the more specific characteristics of the topologies of the peer-to-peer databases such as degree correlations, sparsity, and expansion/conductance, and consider these characteristics (in addition to the degree distribution) in our percolation-based analysis of the query dissemination. Third, we plan to extend our proposed family of data retrieval operations to include operations for answering *continuous* data retrieval requests. For this purpose, we will adopt the SIS (Susceptible-Infected-Susceptible) disease dissemination model. Unlike SIR, which models epidemic diseases that disseminate once throughout the social net-

work and quickly disappear, SIS models *endemic* diseases that become resident in the social network and continuously disseminate.

Acknowledgments

This research has been funded in part by NSF grants IIS-0238560 (PECASE), IIS-0534761, IIS-0742811 and CNS-0831505 (CyberTrust), and in part from the METRANS Transportation Center, under grants from USDOT and Caltrans. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. L. Adamic, R.M. Lukose, A.R. Puniyani, and B.A. Huberman. Search in power-law networks. *Physics Review Letters*, 64(46135), 2001.
2. F. Banaei-Kashani and C. Shahabi. Partial selection query in peer-to-peer databases (poster paper). In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, April 2006.
3. A.L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
4. B. Bollobas. *Random Graphs*. Academic Press, New York, 1985.
5. S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, March 2005.
6. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems(ICDCS)*, 2002.
7. A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1987.
8. D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. B. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks. Technical Report CSD-TR 02-0013, UCLA, 2002.
9. A. Ganesh, L. Massouli, and D. Towsley. The effect of network topology on the spread of epidemics. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, March 2005.
10. C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, March 2005.
11. K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, 2003.
12. H. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, Otober 2000.
13. J. Hromkovic, R. Klasing, B. Monien, and R. Peine. Dissemination of information in interconnection networks (broadcasting and gossiping). *Combinatorial Network Theory*, pages 125–212, 1996.

14. M. Jovanovic. Modeling large-scale peer-to-peer networks and a case study of gnutella. Master's thesis, University of Cincinnati, 2001.
15. R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2000.
16. D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2003.
17. L. Li, J. Halpern, and Z. Haas. Gossip-based ad hoc routing. In *Proceedings of the Conference on Computer Communications (INFOCOM)*, 2002.
18. Limewire.com. Gnutella, 2006. http://www.limewire.com/.
19. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on supercomputing (ICS)*, June 2002.
20. Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
21. M. Molloy and B. Reed. A critical point for random grraphs with a given degree sequence. *Random Structures and Algorithms*, 6:161–180, 1995.
22. M.E.J. Newman, S.H. Strogatz, and D.J. Watts. Random graphs with arbitrary degree distribution and their applications. *Physical Review E*, 64(026118), 2001.
23. G. Ozsoyoglu, K. Du, S. Guruswamy, and W. Hou. Processing real-time, non-aggregate queries with time-constraintsin case-db. In *Proceedings of the 8th International Conference on Data Engineering (ICDE)*, February 1992.
24. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, August 2001.
25. M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P)*, August 2001.
26. S. Saroiu, P.K. Gummadi, and S.D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, January 2002.
27. N. Sarshar, P. Oscar Boykin, and V. Roychowdhury. Percolation search in power law networks: Making unstructured peer-to-peer networks scalable. In *Fourth International Conference on Peer-to-Peer Computing (P2P)*, 2004.
28. SharmanNetworks. Kazaa, 2006. http://www.kazaa.com/.
29. D. Stauffer and A. Aharony. *Introduction to Percolation Theory*. Taylor and Francis, second edition, 1992.
30. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, August 2001.
31. S.V. Vrbsky and J.W.S. Liu. Approximate - a query processor that produces monotonically improving approximate answers. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 5(6):1056–1068, December 1993.
32. H.S. Wilf. *GeneratingFunctionology*. Academic Press, second edition, 1994.
33. B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, March 2003.