Analysis of Search and Replication in Unstructured Peer-to-Peer Networks

Saurabh Tewari, Leonard Kleinrock

Computer Science Department, University of California at Los Angeles

Los Angeles, CA90095

{stewari, lk@cs.ucla.edu}

ABSTRACT

This paper investigates the effect of file replicas on search performance in unstructured peer-to-peer networks. We observe that for a search network with a random graph topology where file replicas are uniformly distributed, the hop distance to a replica of a file is logarithmic in the number of replicas. Using this observation we show that flooding-based search time is optimized when the number of replicas is proportional to the file request rates. Further, at this optimal replica distribution, the query-processing load is comparable to the optimal query-processing load for typical file popularity distributions. We also compare flooding search to the random walk search under their optimal replica distributions and find that flooding has logarithmically better search time than random walk yet incurs comparable query-processing load for typical file popularity distributions. The paper also gives some distributed algorithms to achieve optimal or near-optimal replica distributions automatically. Earlier, we had shown that the linearly proportional replica distribution resulted in uniform download utilization factor which gives optimal download performance; in this paper we show that this distribution also results in a uniform download utilization factor for each file stored at a node (independent of the file request rate) which causes nodes to be indifferent as to which files they store. We extend these results in this paper by showing that the optimality holds not only for equal file sizes and homogenous peers, but also for unequal file sizes and a certain class of non-homogeneity in peer service capacities.

Keywords

Peer-to-peer, Replication, Search Performance, Random Graphs, Unstructured Networks, Flooding.

1. INTRODUCTION

Peer-to-peer networks offer a promise of systems that automatically scale in capacity as the number of users increases and yet are extremely robust, automatically adapting to failures of nodes/links as well as changes in usage patterns, all at virtually no cost. These loosely organized networks of autonomous entities (user nodes or "peers"), which make their resources available to other peers, represent a new computing paradigm where the service consumers are, now, the service providers as well. These systems offer the potential of a tremendous improvement over the traditional client-server architectures.

Moving away from traditional architectures, however, implies that the existing methods of finding the desired objects do not work well. In particular, querying a centralized index to find the location of the desired object is a return to the client/server paradigm. Besides, measurement studies of peer-to-peer systems [13, 30] report that the "peers" usually stay in the network for short periods of time and hence, keeping content indexes farther than one hop away from the source has limited utility. In this paper, we focus on fully distributed peer-to-peer networks where the content index of a node is maintained only at that node, and a search for a file must probe individual peers to find the location of the file. Since it is infeasible for a node to keep addresses of all nodes in a large dynamic network, each node stores the addresses for a subset of peers and other nodes are reached via these neighbors. Given this search overlay network, in the absence of any information about the location of the queried object, the two main alternatives for search are *flooding* search and *random walk* search. As the name implies, in the flooding search, the query is sent to all the neighbors and if they do not have the object they forward the query to all their neighbors and so on. In contrast, in random walk search, the querying node sends the query to one randomly selected neighbor and if that neighbor does not have the desired object, it will forward the query to one of its neighbors (selected randomly). Thus, with each additional hop, the number of peers queried increases linearly for random walk (independent of the search network topology), and exponentially for flooding (for typical search network topologies). A major part of our work is to compare of the relative performance of these two approaches in relation to the number of replicas of each file.

Our first major contribution, covered in Section 3, is establishing a logarithmic relation between the hop distance to the nearest replica of a file and the number of replicas, when the search network has a random graph topology. This logarithmic relation between the hop distance to the nearest replica and the number of replicas is found to hold for other popular topologies as well. Section 4 contains our second major contribution, where we explore the performance of controlled flooding search (flooding that terminates as soon as the first replica is found) and provide results on (i) the optimal replica distribution (we find that the optimal number of replicas is linearly proportional to the file request rates), (ii) the average search distance and (iii) the search traffic/query-processing load at this search-time optimal replica distribution. We compare this search time and query-processing load expressions to the search time and the query-processing load expressions under the square-root proportionality replica distribution known to be optimal for query-processing load and establish that the linearly proportional replica distribution incurs a comparable query-processing load to the square-root proportionality replica distribution at typical file request rate distributions. Our third major contribution, covered in Section 5, is a comparison of controlled flooding search and random walk search under their respective optimal replica distributions. We extend the results in [8] to find the optimal average search distance and the optimal query-processing load for random walk search and show that, under their respective optimal distributions, controlled flooding search provides logarithmically better search time than random walk search and, at the same time, has similar search traffic. We address many of the assumptions in our analysis in Section 6 and discuss how our results are affected as various complexities of practical systems are brought into the model.

Before proceeding to a discussion of related work in Section 2, we note that our study ignores an alternative class of peer-to-peer systems where searching for the content is not required [7, 26, 31] either due to strict mapping of file contents and file locations [26, 31] or through use of centralized indexing [24]. While these systems do have certain benefits [23, 26, 31, 34], these come with their own issues [20, 24] involving the dynamic environment of peer-to-peer networks.

2. RELATED WORK

Since search is a critical component in unstructured peer-to-peer- networks, several researchers have addressed this issue. An early concern in peer-to-peer networks was the large amount of search traffic generated [25]. [22] is one of the earlier works in the area that examines several of the relevant issues. The paper notes the exponential growth in the number of nodes queried with each additional hop in conventional flooding search and suggests that a finer-grained control of the number of nodes queried is more appropriate to limit the search traffic. While [22] does suggest using the *Expanding Rings* method for adaptively terminating flooding search, it proposes to use multiple random walks for search as it offers a linear growth in nodes queried by additional hops and has less likelihood of querying the same node repeatedly than flooding search. [22] also discusses the role of replication in improving performance and a detailed analysis of replication strategies is presented in [8]. [8] is very similar to our work in this paper. They seek to optimize the file replica distribution so as to minimize the query-processing load (they call it "average search size") and find the optimal replica distribution to be one of square root proportionality to the request rates. Like our work, they also assume that the file replicas are uniformly distributed over the network. They also present distributed algorithms that achieve nearly optimal replica distribution (assuming the search method is a random walk). In addition to providing analogous results for the search time in controlled flooding search, we compare the search times and the query-processing loads in the controlled flooding and random walk searches and show that the controlled flooding search generates search traffic comparable to the random walk search for typical file request rate distributions while completing the file searches logarithmically faster when the two methods are operating under their respective optimal replica distributions.

Whereas controlled flooding search has received little interest (other than by [22] and [3]), improving random walks has been the topic of several papers. The proposals include constructing and/or taking advantage of more suitable topologies [1, 6, 28], or augmenting the random walk with memory-based hints [32]. [11] presents some analytic results on random walks and discusses benefits of random walks over hop-limited flooding when the metric is number of replicas discovered. However, it ignores the critical search time metric where flooding holds logarithmic superiority. While [3] does address controlled flooding search analytically, optimizing the replica distribution is not pursued. [9] presents a detailed model incorporating many of the characteristics of deployed peer-to-peer file sharing systems including the existence of correlation between the number of replicas of a file and file popularity. Again, optimizing the replica distribution is not pursued and the flooding search model is hop-limited flooding. Further, their model for the number of peers queried for a hop-limit h is different than ours.

Some of the work in balanced search trees [15, 16] can also be considered similar to our work in that the depth of a key for search trees of non-distinct keys [15] is similar to the hop distance to the nearest replica of a file in our problem, and the optimal average search depth in these trees [16] corresponds to the optimal search distance in our case.

3. HOP DISTANCE TO A REPLICA AS A FUNCTION OF NUMBER OF REPLICAS

Our peer-to-peer system model consists of *nodes* and *files*. The term *files* represents any generic content while a *node* corresponds to a user or peer (these terms are used interchangeably in this paper). Each file has a certain request rate associated with it, reflecting user interest in that file. A file can have more than one replica in the system. When a user requests a file, a search for the file is initiated and other nodes in the network need to be queried if the file is not available locally. Since, in a large network, a node cannot easily store addresses for all peer nodes, nodes store addresses for a subset of all the peer nodes and the remaining nodes are reached via the "neighbor" peers. We assume these node interconnections to be bidirectional and all nodes belong to a single strongly-connected graph which we call the *search network*.

We are interested in the shortest distance from a "querying" node (the node searching for a file) to a replica of the file measured in number of hops (i.e. the number of hops taken by a breadth-first search from the "querying" node). We call this metric the *hop distance*. Let $\tau_i(n_i)$ be the expected shortest distance from a querying node to a replica of file i when there are n_i replicas[1] of the file in the network and $n_i \geq 1$.

[1] We assume that the replicas of a file are uniformly distributed over the entire network. Specifically, when there are n_i replicas of a file and M nodes in the network, we use the approximation that the probability of finding file i at a randomly selected node is n_i/M. In Section 6, we discuss this assumption in relation to our work.

The hop distance to a replica clearly depends on the topology of the search network. For example, if the search network was a fully-connected graph, the hop distance to a non-local file will be 1 regardless of the number of replicas. We assume the search network to have an Erdos-Renyi random graph topology [4]. When all nodes are "truly" peers (i.e. are homogenous), random graphs are considered be a good choice of topology [22]. In Section 3.1, we investigate the relation between the hop distance to a replica and the number of replicas of the file in random graph topologies. Based on comprehensive simulations, we conclude that the hop distance to a file is logarithmically related to the inverse of the number of replicas of that file. When the nodes are not homogenous, topologies such as power-law random graphs and superpeer networks can be useful [6, 28, 34] and many of the currently deployed peer-to-peer file-sharing systems generate such networks [12, 29]. We evaluate some of these topologies including one extracted from a Gnutella trace in Section 3.2 and now we find that the logarithmic relation between the hop distance and the number of replicas is a good approximation unless most nodes have the file (i.e. the number of replicas of a file is of the same order as the total number of nodes).

3.1 Random Graph Topology

We simulated a number of random graph topologies with a varying number of nodes and varying per-node average degree (i.e. the average number of neighbors each node has in the topology). The results are shown in Figure 1 and 2. The topology construction follows the standard Erdos-Renyi random graph construction method [4] with the extra step that we drop nodes that do not belong to the largest connected component[2]. After obtaining this topology, we uniformly populate a certain fraction of nodes with a file (i.e. we randomly pick a node and if the node does not already have the file, we mark it as having that file now; this process is repeated until the desired number of replicas have been populated). After the replica population, each node does a breadth-first search for the file. On the y-axis, we plot the average hop distance to the first replica found from the searching node (average taken across all nodes) while the x-axis in the figures shows the fraction of nodes that have the file. These figures[3] establish the following three properties of the relation between the hop distance to a replica and the number of replicas of the file:

1. $\tau_i(n_i)$, the average hop distance to a file, is negative-logarithmically related to n_i/M, the fraction of nodes that have the file.
2. $\tau_i(n_i)$ is independent of the total number of nodes.
3. Increasing per-node average degree decreases the slope of the relation between $\tau_i(n_i)$ and $-ln(n_i/M)$.

Thus, we can write the following relation:

$$\tau_i(n_i) = -\beta\, ln(n_i/M) \qquad (1)$$

where we know that β is related to the per-node average degree. In Figure 3, we plot β vs. average degree. We also plot $1/\beta$ on the secondary axis as the β vs. average degree curve suggests an inverse proportionality. As shown in the figure, $1/\beta$ is logarithmic in average degree. Substituting $\beta = \alpha/ln(d)$ where d is the per-node average degree and α, based on these simulation results, is 1.03, we obtain:

$$\tau_i(n_i) = -\alpha\, ln(n_i/M)\,/ln(d) \; = -1.03\, log_d(n_i/M)$$

Neglecting the 1.03 multiple, we can write:

$$\tau_i(n_i) \; = \; log_d(M/n_i) \qquad (2)$$

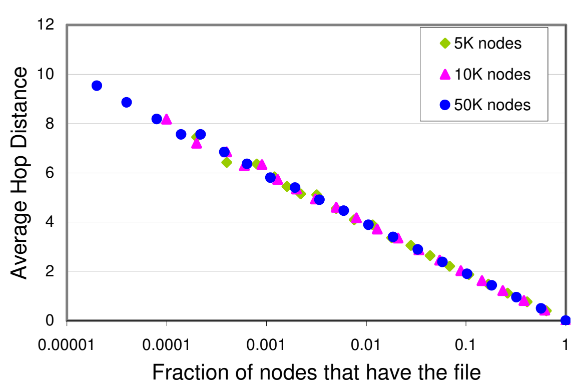


Figure 1. Random Graph: Effect of Number of Nodes (Avg Degree ~3.2)

---

[2] The number of nodes listed in the figures is the number of nodes in the largest connected component rounded-off to the nearest thousand while the average degree reported is the average per-node degree in this largest connected component.

[3] The equations shown in Figures 2-6 are the standard regression equations. R^2 values near 1 indicate a very good fit.

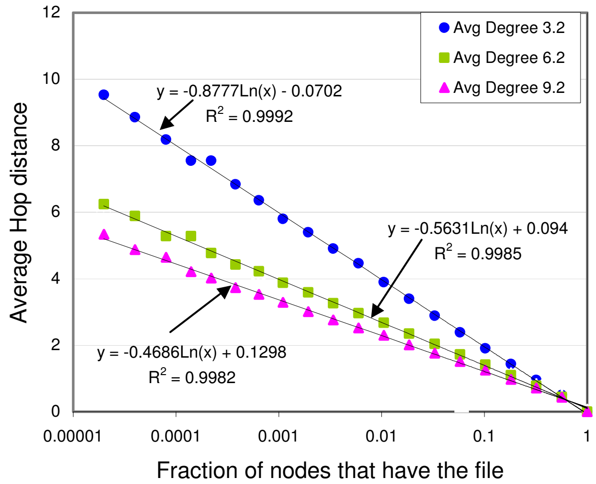


Figure 2. Random Graphs: Effect of Degree (~50K nodes)

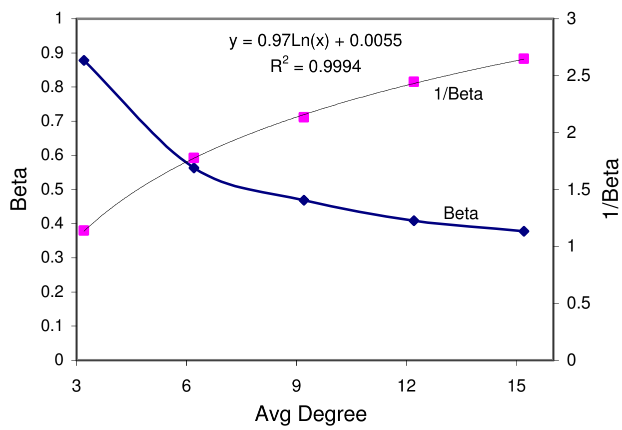


Figure 3. Slope as a function of Degree

This expression is consistent with the expected hop distance for the case of one replica; when there is only one replica of a file in the network, the average hop distance to the file will be proportional to the network diameter. The network diameter of a random graph with average degree d is $O(log_d M)$ [4] which is what Eq. (2) gives for $n_i = 1$. It is also possible to approximately prove Eq. (2) analytically for finite n_i in the asymptotic case of $M \rightarrow \infty$ as we show next.

We wish to calculate the expected number of hops to the nearest replica of the file being searched given the number of replicas of that file in the network. Assuming that the replicas of a file are uniformly distributed in the network, the probability of finding the file at a randomly selected node is n_i/M when there are n_i replicas of the file in the network. In addition to notation defined earlier, define:

P_h = probability that a search for file i takes exactly h hops

F_h = probability that file i is not found in h hops

Therefore, $P_h = F_{h-1} - F_h$. The average hop distance is: $\tau_i(n_i) = \sum_{h=0}^{H_M} hP_h$ where H_M is the hop-distance by which all the nodes M have been probed. Hence,

$$\tau_i(n_i) = \sum_{h=1}^{H_M} h[F_{h-1} - F_h]$$

$$= \sum_{h=0}^{H_M - 1} F_h - H_M F_{H_M}$$

Using the approximation that the probability of finding file i at a node is independent of the probability of finding that file at any other node, we can write $F_h = (1-n_i/M)^{S_h}$ where S_h is the number of nodes probed up to h hops including the node initiating the search. For Erdos-Renyi random graphs, S_h is approximately = d^h [4]. Therefore,

$$\tau_i(n_i) \approx \sum_{h=0}^{H_M-1}(1-n_i/M)^{d^h} - H_M(1-n_i/M)^M$$

since $d^{H_M} = M$. Note that, as a result of our assumption of the probability of finding file i at a node being independent of whether the file i has been found or nor found at other nodes, we obtain a non-zero value for F_{H_M}, the probability that the file is not found even after probing all nodes which, clearly, is not possible. Hence, we drop this last term from subsequent derivation.

Using the Euler-Maclaurin summation formula:

$$\tau_i(n_i) \approx \int_0^{H_M}(1-n_i/M)^{d^h}\,dh + \sum_{k=1}^{n}\frac{B_k}{k!}[f^{(k-1)}(H_M) - f^{(k-1)}(0)] + R_{f,n}$$

where B_k are the Bernoulli numbers, $f(h) = (1-n_i/M)^{d^h}$, and $R_{f,n}$ is the remainder term in the summation for function $f(h)$.

Substitute $t = d^h$: $h = log_d t \Rightarrow dh = (1/lnd)(dt/t)$. At $h = H_M$, $t = M$ and at $h=0$, $t=1$ (as the querying node is already probed). Therefore:

$$\tau_i(n_i) \approx \frac{1}{\ln d}\int_1^M \frac{(1-n_i/M)^t}{t}dt + \sum_{k=1}^{n}\frac{B_k}{k!}[f^{(k-1)}(H_M) - f^{(k-1)}(0)] + R_{f,n}$$

Using the Euler-Maclaurin summation formula again, we get:

$$\tau_i(n_i) \approx \frac{1}{\ln d}\left[\sum_{t=1}^{M-1}\frac{(1-n_i/M)^t}{t} - \sum_{k=1}^{n}\frac{B_k}{k!}[g^{(k-1)}(M) - g^{(k-1)}(1)] - R_{g,n}\right] + \sum_{k=1}^{n}\frac{B_k}{k!}[f^{(k-1)}(H_M) - f^{(k-1)}(0)] + R_{f,n} - H_M(1-n_i/M)^M$$

where $g(x) = (1-n_i/M)^t/t$, and $R_{g,n}$ is the remainder term in the summation for function $g(x)$.

Since, $B_k = 0$ for odd k (other than 1) and $B_4 = 1/720$, we can neglect drop the higher-order terms beyond $k = 3$ and the remainder term in the Euler-Maclaurin summation formulas. The required $g(x), f(x)$ and $g'(x), f'(x)$ terms at the limits can be evaluate as:

$g(1) = (1-n_i/M)$,

$g(M) = (1-n_i/M)^M/M$

$g'(x) = (1-n_i/M)^t/t[ln(1-n_i/M) - (1/t)]$

$\Rightarrow g'(1) = (1-n_i/M)[ln(1-n_i/M) - 1]$

$g'(M) = \{(1-n_i/M)^M/M\}[ln(1-n_i/M) - 1]$

$f(0) = (1-n_i/M)$

$f(H_M) = (1-n_i/M)^M$

$f'(x) = ln(1-n_i/M) f(x) d^x lnd$

$\Rightarrow f'(0) = (1-n_i/M) ln(1-n_i/M) lnd$

$f'(H_M) = M\, lnd\, (1-n_i/M)^M ln(1-n_i/M)$

As discussed earlier, $F_{H_M} = (1-n_i/M)^M \approx 0$. Hence, $g(M), g'(M), f(H_M), f'(H_M)$ are all 0. Therefore,

$$\tau_i(n_i) \approx \frac{1}{\ln d}\left[\sum_{t=1}^{M-1}\frac{(1-n_i/M)^t}{t} + \frac{1}{2}(1-n_i/M) + \frac{1}{12}(1-n_i/M)[1-\ln(1-n_i/M)]\right] - \frac{1}{2}(1-n_i/M) + \frac{1}{12}(1-n_i/M)\ln(1-n_i/M)\ln(d)$$

$$= \frac{1}{\ln d}\sum_{t=1}^{M-1}\frac{(1-n_i/M)^t}{t} + (1-n_i/M)\left[\left(\frac{1}{2}+\frac{1}{12}\right)\frac{1}{\ln d} - \frac{1}{2}\right] + \frac{1}{12}(1-n_i/M)\ln(1-n_i/M)\left[\ln d - \frac{1}{\ln d}\right]$$

As $M \to \infty$, applying the series summation $\sum_{k=1}^{\infty}\frac{(1-1/x)^k}{k} = \ln x$, we can write $\sum_{t=1}^{M-1}\frac{(1-n_i/M)^t}{t} = \ln(M/n_i)$. As $M \to \infty$, the $ln(M/n_i)$ term will dominate the other $(1-n_i/M)$ terms for finite n_i and, hence, we can neglect the other terms to finally show:

$$\tau_i(n_i) \approx (1/\ln d)[\ln(M/n_i)] = \log_d(M/n_i)$$

This result can also be shown by an alternate approach that is more intuitive. As already discussed, under our assumption that the n_i replicas are uniformly distributed in the network, a randomly selected node has a probability n_i/M of having the file. In the absence of any information on the possible location of a file, the search is *blind* as to where to look and, therefore, must probe different nodes (i.e., *sample* *the node space*) and the different search methods differ only in how they go about querying these nodes: in flooding, each additional hop simultaneously probes exponentially more nodes than the previous hop, whereas a random walk probes one additional node with every additional hop. When the search overlay network is a random graph, each edge is equally likely and, hence, the search could revisit nodes that have already been queried. Thus, a blind search for file i is a sequence of Bernoulli trials with n_i/M as the probability of success. Therefore, the average number of nodes probed until *success* (i.e. the file is found) is (*probability of success*)$^{-1}$ or M/n_i.

Since a random walk queries one additional node per round, it takes M/n_i rounds to find the file. In contrast, since the number of nodes reached in h hops in an Erdos-Renyi random graph with average per node degree d is d^h [4], flooding can query M/n_i nodes in just $\log_d(M/n_i)$ rounds in an Erdos-Renyi random graph search overlay network with average per node degree d.

Given the result that a blind search will need to probe M/n_i to find an object that has n_i replicas uniformly distributed in a network of M nodes, we can estimate the search time for any (blind) search techniques over any search overlay network topology. For example, in any search overlay network that has exponential expansion in number of nodes reached by one additional hop, the logarithmic relation between the hop distance to the nearest replica and the number of replicas of the file will be a very good approximation for the search performance. In the next section where we study other commonly suggested topologies in unstructured peer-to-peer networks, we find this logarithmic relation between the hop distance to the nearest replica and the number of replicas of the file to be a very good approximation as our analysis suggests.

3.2 Other Topologies

Measurement studies of the deployed peer-to-peer systems indicate heterogeneity in the peers [27]. In the presence of node heterogeneity, it is possible to construct better search network topologies than a random graph by correlating the node degree and the node capacity. Two popular alternative search network models are power-law random graphs and, what we call, superpeer networks. Power-law random graph is a well-studied topology in which the degree distribution follows Zipf's law [2]. Many real-world networks seem to follow this model [2] and the initial Gnutella1 topology seemed to fit this model as well [25]. When the search mechanism involves random walks, power-law random graphs are the preferred topology [1, 28] and decentralized methods have been proposed to construct power-law random graphs where the higher capacity nodes are the high-degree nodes [6]. The second class of topologies, which we call, the superpeer network topology classify peers into "superpeers" and "ordinary nodes"; the "ordinary nodes" connect to a superpeer, typically a higher-capacity and more stable node; the superpeers are interconnected to each other. KaZaa [29] and Gnutella2 [12] follow this architecture (see [12, 19, 21] for measured/recommended values). While the interconnection topology between KaZaa supernodes is not known as it uses a proprietary protocol, the Gnutella2 specification suggests that each ultrapeer should connect to 5-30 other ultrapeers [12]. For our simulation of the superpeer network topology, we assume the superpeers to be interconnected in a random graph topology.

In Figures 4-6, we show the simulation results for power-law random graphs, a Gnutella2 topology and a superpeer network topology. The Gnutella2 trace had over 100K nodes (see [21] for the details of the measurement methodology). Other topologies had 50K nodes.

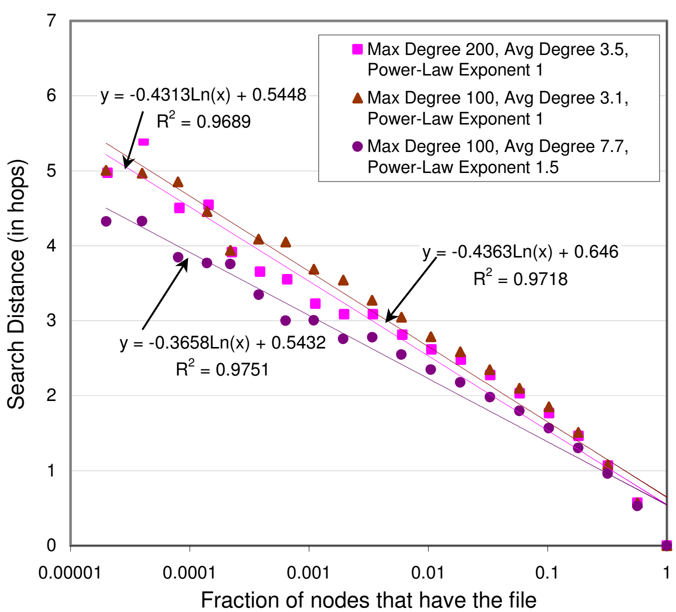


Figure 4. Power-Law Random Graphs (~50K nodes)

These results indicate that the logarithmic relation between the hop distance and the number of replicas is a good approximation for all these topologies when a file is at less than 10% of the nodes. The faster than linear decrease in hop distance when a large fraction of nodes have the file may be explained by the "percolation effect" [28]. While the logarithmic approximation is worst for power-law random graphs, we emphasize that even the Gnutella1 did not have a pure power-law random graph topology except in its initial days [25]. The measured Gnutella2 and the superpeer network topologies, on the other hand, show a good fit to the logarithmic approximation. Since more than 10% of the nodes having a file is not very likely, we believe that our logarithmic relation derived for Erdos-Renyi random graphs is applicable to practical peer-to-peer networks.

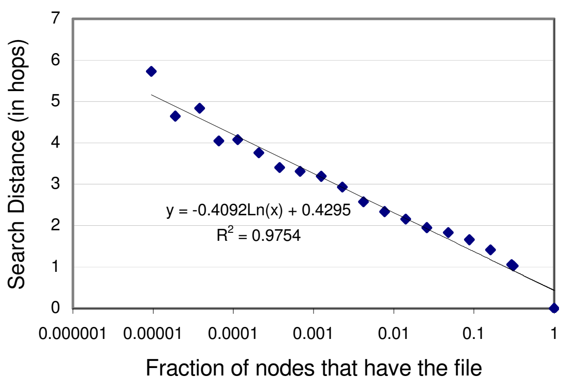


Figure 5. Gnutella2 Measured Topology (~100K nodes, Trace Date: 2003 [21])

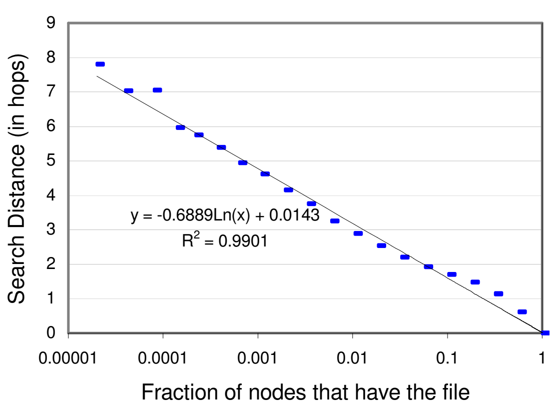


Figure 6. Superpeer Network Topology (~50K nodes, 32 nodes / superpeer)

4. SEARCH PERFORMANCE OPTIMIZATION

Search mechanisms for a file in an unstructured network where no information is available on file location are broadly classified into *flooding* (and its variations) and *random walk* (and its variations). In a random walk search, the querying node sends the query for the file to one of its neighbors (selected randomly) and if that neighbor does not have the file, it will forward the query to one of its neighbors (selected randomly) and the process repeats until the file is found. In flooding, the querying node sends the query for a file to all its neighbors and if their neighbors do not have the file either, each of them forwards the query to all their neighbors and the process repeats. We explore flooding search in this paper. Our assumption is that the query flood stops as soon as a replica of a file is found[4]. This controlled flooding is essentially a breadth-first search for the file on the search network as described in Section 3 where we established the relation between the search distance to the nearest replica and the number of replicas. In this section, we find the optimal replica distribution for the search time in controlled flooding search.

The performance metrics relevant to search are the time taken to find the file and the amount of overhead due to the file search process. In the absence of detailed information on the underlying physical topology, the average number of hops taken to find the file can be considered to reflect the search time. Therefore, one performance metric is τ, *the average search distance* per query. The average number of nodes queried per search is a good metric for the amount of overhead associated with file searches. A higher number for the average number of nodes probed per search implies that each node has to process more queries. We call this second search performance metric the

[4] Section 6.2 addresses the practical issues regarding this assumption.

query-processing load Q. Since each node probed requires a separate message, this metric also incorporates the network traffic overhead due to file searches.

In the random walk search, the number of hops taken for the search (referred to as "search size" in [8]) is equal to the number of search queries sent. Thus, optimizing the search size (as in [8]) optimizes the overall search performance in random walk search.

4.1 Problem Definition

To the peer-to-peer system model described in the beginning of Section 3, we add the following description. There are N unique files in the system, each with an associated request rate λ_i for file i per node[5]. We assume that each file is of equal size. Nodes have finite local storage space to store file replicas. We assume that the storage space at each node is equal and has the capacity to store K files. The search mechanism is controlled flooding and a node will always satisfy a request for a file present in its local storage. The notation for the various system parameters discussed is:

M = number of nodes in the system

N = number of unique files in the system

K = per-node storage size in number of files

λ_i = request rate of file i per node

$\lambda = \sum_{i=1}^{N} \lambda_i$

n_i = number of replicas of file i in the system

$\tau_i(n_i)$ = average search distance for file i when there are n_i replicas of the file in the system.

Our objective is to find the optimum value for n_i, the number of replicas of file i (for all i), which minimizes the average search distance

$$\tau = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \tau_i(n_i) \qquad (3)$$

Thus, our optimization problem may be stated as:

$$\underset{\{n_i\}_{i=1}^{N}}{Min}\left[\tau = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \tau_i(n_i)\right] \qquad (4)$$

Subject to[6]:

$$\sum_{i=1}^{N} n_i \le KM \qquad (5)$$

Search distance will be minimized if all the files could be stored at all the nodes, but the total storage space available in the network is limited to the combined storage space of all nodes. Thus, the optimization is subject to the constraint in Eq. (5), namely, that the total number of replicas of all the files should not exceed the total storage available. Having multiple replicas of a file at a node does not help the file search and we prohibit it. Therefore, the number of replicas of any file can never be more than the number of nodes. Further, there is at least one replica of each file. These two conditions give $2N$ other inequality constraints –

$$n_i \le M \quad \text{for all } i = 1 \text{ to } N \qquad (6)$$

$$n_i \ge 1 \quad \text{for all } i = 1 \text{ to } N \qquad (7)$$

Note that, only if the per-node storage capacity is more than the number of unique files (i.e. $N < K$), will Eq. (5) be an inequality and in that case, all the sub-equations in Eq. (6) will be equalities. So, for example, if there were only two files and each node had the storage space for 5 files, all files will be at all the nodes ($n_i = M$) but not all of the available storage space will be filled ($\sum_{i=1}^{N} n_i < KM$).

4.2 Optimal Replica Distribution

Theorem 1:

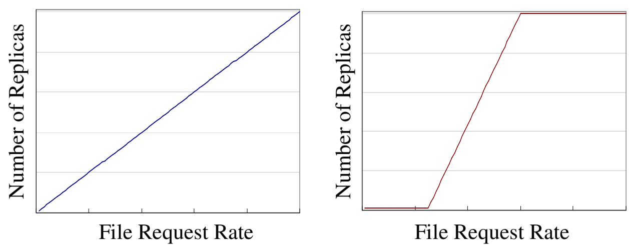
Given an unstructured peer-to-peer network where the relation between the hop distance $\tau_i(n_i)$ to the nearest replica and the number of replicas n_i is of the form $\tau_i(n_i) = \alpha \log_d(M/n_i)$ and the storage capacity at each node is equal, then the average search distance in controlled flooding query searches is minimized when the number of replicas, n_i, of file i is piece-wise linear with respect to the file request rate λ_i, (i = 1,2, …, N) i.e.

---

5 λ_i is the request rate for file i in the system averaged over all M nodes i.e. $\lambda_i = \frac{\sum_{j=1}^{M} \lambda_{ij}}{M}$.

6 Our assumption that the replicas are uniformly distributed in the network in conjunction with the constraint in Eq. (5) supports our constraint that no node stores more than K files.

$$
n_i = \begin{cases} \dfrac{\lambda_i}{\lambda} KM & \text{if } \dfrac{1}{KM} \le \dfrac{\lambda_i}{\lambda} \le \dfrac{1}{K} \;\; \forall i, \qquad (8) \\[2ex] Max(1, Min(\dfrac{\lambda_i \beta}{\gamma_0}, M)) & \text{in the general case.} \qquad (9) \\[1ex] \quad \text{where } \gamma_0 \text{ is s.t. } \sum_{i=1}^{N} n_i = KM & \end{cases}
$$



(i) *(ii)*

Figure 7. Optimal Replica Distribution: (i) under constraints on file request rates (ii) in the general case

Proof:

As discussed in Section 4.1, minimization of the average search distance is a constrained optimization problem. The classical approach to solving constrained optimization problems is the method of Lagrange multipliers. First we will show the result for part *(i)*. We will ignore the constraints specified by Eqs. (6) and (7) for now and instead show that our optimal solution satisfies these constraints under conditions on λ_i specified in part *(i)*. The Lagrangian of our constrained optimization problem is:

$$H = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \tau_i(n_i) + \gamma(\sum_{i=1}^{N} n_i - K\,M)$$

Minimizing H over all n_i for $i = 1$ to N :

$$\frac{\partial H}{\partial n_i} = \frac{\lambda_i}{\lambda} \frac{\partial \tau_i}{\partial n_i} + \gamma = 0 \qquad i = 1 \text{ to } N \qquad (10)$$

Using $\tau_i(n_i) = \alpha \log_{d}(M/n_i) = -\beta \ln(n_i) + c$,

$$\frac{\partial \tau_i}{\partial n_i} = -\frac{\beta}{n_i}$$

Substituting this in Eq. (10), we obtain,

$$n_i = \frac{\lambda_i}{\lambda} \frac{\beta}{\gamma}$$

Applying the constraint $\sum_{i=1}^{N} n_i = KM$ to remove the unknown constant β/γ, we obtain the optimum number of replicas as:

$$n_i = \frac{\lambda_i}{\lambda} KM \qquad i = 1 \text{ to } N \qquad (11)$$

The constraints in Eq. (6), (7) are clearly satisfied if $\frac{1}{KM} \le \frac{\lambda_i}{\lambda} \le \frac{1}{K}$ $\forall i$. This proves part *(i)* of the theorem.

Without this condition on λ_i, one can rewrite the problem as a maximization problem using a modified Lagrangian. Using $\tau_i(n_i) = -\beta \ln(n_i) + c$, and including both the constraints in Eq. (6), (7) and rewriting the $n_i \ge 1$ constraint as $-n_i \le -1$, the modified Lagrangian is:

$$G = \beta [\sum_{i=1}^{N} \lambda_i \ln(n_i)] - \gamma_0 [\sum_{i=1}^{N} n_i - KM] - \sum_{i=1}^{N} \gamma_i (n_i - M) - \sum_{i=1}^{N} \alpha_i (-n_i + 1)$$

The Kuhn Tucker Conditions for the modified Lagrangian are:

$$\frac{\lambda_i \beta}{n_i} - \gamma_i - \gamma_0 + \alpha_i = 0 \quad \text{for } i = 1 \text{ to } N \qquad (12)$$

$$\sum_{i=1}^{N} n_i \le KM, \quad \gamma_0 \ge 0, \quad \text{and} \quad \gamma_0[\sum_{i=1}^{N} n_i - KM] = 0 \qquad (13a)$$

$$n_i \le M, \quad \gamma_i \ge 0, \quad \text{and} \quad \gamma_i (n_i - M) = 0 \quad \text{for } i = 1 \text{ to } N \qquad (13b)$$

$$-n_i \le -1, \quad \alpha_i \ge 0, \quad \text{and} \quad \alpha_i (-n_i + 1) = 0 \quad \text{for } i = 1 \text{ to } N \qquad (13c)$$

From Eq. (12): $n_i = \dfrac{\lambda_i \beta}{\gamma_i + \gamma_0 - \alpha_i}$

Eq. (13b), (13c) imply that: either $\gamma_i = 0$ or $n_i = M$, and also either $\alpha_i = 0$ or $n_i = 1$, respectively. Therefore, the optimum solution is:

$$n_i = Max(1, Min(\frac{\lambda_i \beta}{\gamma_0}, M))$$

where, from Eq. (13a), γ_0 is such that $\sum_{i=1}^{N} n_i = KM$ when the storage size is not large enough to store all the files (i.e. $N \ge K$). This proves part *(ii)* of the theorem.

■ Q.E.D.

In much of the remaining discussion, we assume $\frac{1}{KM} \le \frac{\lambda_i}{\lambda} \le \frac{1}{K}$ $\forall i$ and use Eq. (11) as the optimal replica distribution. If KM is sufficiently large (which is the likely case), $\frac{\lambda_i}{\lambda} \ge \frac{1}{KM}$ should be satisfied. If $\frac{\lambda_i}{\lambda} \ge \frac{1}{K}$, the optimal distribution is for the higher request rate files to be located at all M nodes; but if a file is located at all nodes, we can simply assume that they are not part of the peer-to-peer system as no one ever lacks them and, hence, no search is ever made for these files. Without these high request rate files in the system, $\frac{\lambda_i}{\lambda} \le \frac{1}{K}$ should be satisfied.

4.2.1 Importance of Linear Proportionality Replica Distribution

As we showed in [35], the linear proportionality replica distribution implies that the load presented to a server to download any file it stores is independent of which file it is. This invariant has significant implications.

Such a load distribution implies that each peer serves the same download load regardless of the files it stores locally (i.e. a *fairness in download load distribution*) if each peer stores the same number of files and has the same link bandwidth (i.e. homogenous service capacities which we have assumed in our system model of this paper also).

Further, this invariant also implies that since each file replica presents the same load to the server storing that file, no peer has any incentive to store any particular file over another. If this were not the case and, say, that the high request rate files had fewer replicas than suggested by linear proportionality replica distribution (and the low request rate files had more replicas than suggested by linear proportionality replica distribution), then a peer that stores a high request rate file will have to serve more download load than a peer that stores only low request rate files. This asymmetry may prompt a peer to replace the high request rate file by a low request rate file to reduce its load. However, while this step will decrease its load, it will increase the load on other nodes that were sharing that high request rate file and they may also decide to replace the high request rate file by a low request rate file. Thus, having unequal download load per file replica introduces the possibility that the most desired files may eventually not be shared by any peers in the system. The linear proportionality replica distribution, by ensuring equal download load per file replica prevents this advantage of greedy behavior and ensures *system stability*.

Finally, since uniform utilization of each node minimizes the download time if the queueing delay is convex in the load on the serving node, the linear proportionality replica distribution *minimizes the download time*. We summarize these results in the following theorem:

Theorem 2:

In a peer-to-peer system where the download requests for a file are uniformly distributed over all the replicas of the file in the system and each file is of the same size, if the number of replicas of file i, n_i, is proportional to the average request rate for file i, λ_i, for all files i.e. $n_i = \alpha\lambda_i$ $\forall i$ where α is the proportionality constant:

1. *The download request rate for any file that a node must serve is independent of which file it is and, hence, a greedy behavior whereby a node selects the files it will share to share to minimize the download requests it serves offers no advantage,*

and if each node has the same storage capacity and the upload link bandwidth,

2. *Each node operates at the same utilization factor independent of the files it has in its storage, and*
3. *If the queueing delay is convex in the load on the serving node, then the download time is minimized.*

■

We extend these results to the unequal file size and the unequal service capacity cases in section 6.3.3 and section 6.3.4 respectively.

4.3 Search Performance with Linear Proportionality Replica Distribution

The *minimum search distance* can be derived by substituting Eq. (11) in Eq. (3) and computing τ. Thus,

$$\tau_{\text{opt}} = -\alpha\sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\log_{\text{d}}\left(\frac{\lambda_i}{\lambda}K\right) = -\alpha\sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\log_{\text{d}}\frac{\lambda_i}{\lambda} - \alpha\log_{\text{d}}K \qquad (14)$$

It is interesting to note that the $-\sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\log_{\text{d}}\frac{\lambda_i}{\lambda}$ term is the entropy of $\{\frac{\lambda_i}{\lambda}\}$, the probability that a query in this peer-to-peer system will be for file *i*. Thus, under the optimal replica distribution, the entropy measures how much we reduce the search time as the skew in the file request rates $\{\lambda_i\}$ increases (the entropy is maximized when $\{\lambda_i\}$ are uniform and reduces as the skew increases). The second term in the τ_{opt} expression accounts for the effect of higher storage space on the search distance. In the extreme example where $K = N$ and, hence, $n_i = M$ (i.e. we are in the domain of Figure 7(ii) and no longer in the domain of Fig. 7(i) as is the case for the rest of this section), all files can be stored at all nodes so no search is required and search distance is zero no matter how high the uncertainty in which files will be requested next.

The second metric of search performance is the *query-processing load*. As we discussed in Section 3.1, a search with no advance knowledge of where the desired object may be located (i.e. when the search starts, as far as this search is concerned, all nodes in the network are equally-likely to have the file) will probe M/n_i nodes, on average, to find a file which has n_i replicas in the network regardless of how the search goes about querying the nodes in the network. Therefore, the average query-processing load per query is the same with controlled flooding and random walk and can be written as:

$$Q = \sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\frac{M}{n_i}$$

Hence, the query-processing load for linear proportionality replica distribution Q^{linear} is:

$$Q^{\text{linear}} = \sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\frac{\lambda}{\lambda_i K} = \frac{N}{K} \qquad (15)$$

This result is of enormous significance. K/N is the fraction of the total number of files each node can store. Thus, Eq. (15) shows that with the linear proportionality replica distribution, the query-processing load per node is inversely proportional to the fraction of the total number of files each node can store. Thus, there is a trade-off for the peers between the storage resources allocated to the peer-to-peer application versus the processing time expended in processing the search queries for the peer-to-peer application. Beyond this trade-off, it is also important to note that the query-processing load per node is independent of the number of nodes in the peer-to-peer system under our optimal replica distribution. Thus, in an application where additional nodes do not bring additional files, we have a fully scalable system.

Let us now compare this to the optimal query-processing load. [8] had found the $n_i \propto \sqrt{\lambda_i}$ replica distribution to be optimal for query-processing load. This can be seen by minimizing $\sum_{i=1}^{N}\frac{\lambda_i M}{\lambda n_i}$ with the total storage constraint $\sum_{i=1}^{N} n_i = KM$, which gives the result

$$n_i = \frac{KM\sqrt{\lambda_i}}{\sum_{i=1}^{N}\sqrt{\lambda_i}}$$

and the optimal query-processing load Q^{opt} as:

$$Q^{\text{opt}} = \sum_{i=1}^{N}\frac{M\lambda_i}{\lambda}\frac{\sum_{i=1}^{N}\sqrt{\lambda_i}}{KM\sqrt{\lambda_i}} = \frac{(\sum_{i=1}^{N}\sqrt{\lambda_i})^2}{\lambda K} \qquad (16)$$

The value of Q^{opt} depends on the distribution of $\{\lambda_i\}$. To obtain the upper bound on Q^{opt}, set the partial derivatives of $\sum_{i=1}^{N}\sqrt{\lambda_i}$ w.r.t. λ_i to 0 which gives $\lambda_i = c$ for $i = 0$ to N where c is a constant. Thus, Q^{opt} is maximized when all the file request rates are equal, i.e. $\lambda_i = \lambda_j\ \forall i,j$. Substituting, $\lambda_i = \lambda/N$, we obtain that Q^{opt} is upper-bounded by $\frac{N}{K}$, the query-processing load Q^{linear} for the $n_i \propto \lambda_i$ replica distribution.

Since, the $n_i \propto \lambda_i$ replica distribution which minimizes the search time also ensures fairness in download load distribution and minimizes the download time [35], this replica distribution will be preferable if the additional query-processing load at this replica distribution is not very large compared to the optimal query-processing load. While [8] does provide the ratio $Q^{\text{linear}}/Q^{\text{opt}}$ of the query-processing load at the $n_i \propto \lambda_i$ replica distribution and the optimal query-processing load when the underlying file popularity distribution is zipf-like, it overlooks the physical constraint that no file can have more copies than the number of nodes and, as discussed in Section 4.2, when the file popularities are very skewed, the high request rate files are populated at all nodes and, hence, are effectively removed from the search process. The removal of high request rate files from the system with the $n_i \propto \lambda_i$ replica distribution decreases the disadvantage in query-processing load incurred by this replica distribution. In order to extend the results in [8] to include this effect, first we re-derive this ratio and then include the effect of large file skew.

As in [8], we assume the file request rate distribution to be zipf-like with the zipf-exponent providing an estimate of the skew in file popularities. For zipf-distribution,

$$\lambda_i = i^{-\alpha} \qquad i = 1,\dots, N$$

$$\lambda = \sum_{i=1}^{N} \lambda_i \approx \int_1^N x^{-\alpha} dx = \begin{cases} \ln N & \text{for } \alpha = 1 \\ \dfrac{N^{1-\alpha}-1}{1-\alpha} & \text{for } \alpha \neq 1 \end{cases} \tag{17}$$

When $\alpha = 0$, all the files have the same request rates and as $\alpha \to \infty$, all the file requests are for only one file. We further obtain:

$$\sum_{i=1}^{N} \sqrt{\lambda_i} \approx \int_1^N x^{-\alpha/2} dx = \begin{cases} \ln N & \text{for } \alpha = 2 \\ \dfrac{N^{1-\alpha/2}-1}{1-\alpha/2} & \text{for } \alpha \neq 2 \end{cases} \tag{18}$$

Thus, ignoring the possibility of files being populated at all nodes, from Eqs. (16), (17) and (18), we obtain:

$$KQ^{\text{opt}} = \begin{cases} 4\left(N^{1/2}-1\right)^2 \Big/ \left(\ln N\right) & \text{for } \alpha = 1 \\ \left(\ln N\right)^2 \Big/ \left(1-N^{-1}\right) & \text{for } \alpha = 2 \\ \left(\dfrac{N^{1-\alpha/2}-1}{1-\alpha/2}\right)^2 \Bigg/ \left(\dfrac{N^{1-\alpha}-1}{1-\alpha}\right) & \text{otherwise} \end{cases} \tag{19}$$

In remainder of the analysis in this section, we assume N to be large (so we can treat N^{γ} to be 0 when $\gamma < 0$, and approximate $N^{\beta} + N^{\gamma}$ by N^{β} when $\beta > \gamma$). Therefore, KQ^{opt} can be written as $4N/\ln N$ when $\alpha = 1$ and as $(\ln N)^2$ when $\alpha = 2$. Moreover, for other values of α, KQ^{opt} can be written as :

when $\alpha < 1$, $\dfrac{(N^{2-\alpha}-2N^{1-\alpha/2}+1)}{(N^{1-\alpha}-1)} \dfrac{1-\alpha}{(1-\alpha/2)^2} \approx N\dfrac{1-\alpha}{(1-\alpha/2)^2}$,

when $1 < \alpha < 2$, $\dfrac{(N^{2-\alpha}-2N^{1-\alpha/2}+1)}{(N^{1-\alpha}-1)} \dfrac{1-\alpha}{(1-\alpha/2)^2} = \dfrac{(N^{2-\alpha}-2N^{1-\alpha/2}+1)}{(1-1/N^{\alpha-1})} \dfrac{\alpha-1}{(1-\alpha/2)^2} \approx N^{2-\alpha}\dfrac{\alpha-1}{(1-\alpha/2)^2}$,

when $\alpha > 2$, $\dfrac{(N^{2-\alpha}-2N^{1-\alpha/2}+1)}{(N^{1-\alpha}-1)} \dfrac{1-\alpha}{(1-\alpha/2)^2} = \dfrac{(1/N^{\alpha-2}-2/N^{(\alpha/2)-1}+1)}{(1-1/N^{\alpha-1})} \dfrac{\alpha-1}{[(\alpha/2)-1]^2} \approx \dfrac{\alpha-1}{[(\alpha/2)-1]^2}$.

Using these expressions and Eq. (15), we obtain:

$$Q^{\text{linear}}/Q^{\text{opt}} = \begin{cases} (\ln N)/4 & \text{for } \alpha = 1 \\ N/(\ln N)^2 & \text{for } \alpha = 2 \\ \dfrac{(1-\alpha/2)^2}{1-\alpha} & \text{when } \alpha < 1 \\ N^{\alpha-1}\dfrac{(1-\alpha/2)^2}{\alpha-1} & \text{when } 1 < \alpha < 2 \\ N\dfrac{(\alpha/2-1)^2}{\alpha-1} & \text{when } \alpha > 2. \end{cases} \tag{20}$$

These are the ratios given in [8].

These ratios hold assuming that $n_i \propto \lambda_i\ \forall i$ for Q^{linear} and $n_i \propto \sqrt{\lambda_i}\ \forall i$ for Q^{opt}. From Theorem 1, we know that $n_i \propto \lambda_i\ \forall i$ only if $\dfrac{1}{KM} \leq \dfrac{\lambda_i}{\lambda} \leq \dfrac{1}{K}\ \forall i$ and, in the general case, n_i is described by the piece-wise linear expression of Eq. 9. Thus, for zipf-distributed file request rates, the conditions under which the replica distribution has linear proportionality as assumed in [8] can be derived as follows.

For zipf-distributed file request rates, the maximum file request rate $\lambda_1 = 1$ and the minimum file request rate is $\lambda_N = N^{-\alpha}$. Using these maximum and minimum file request rates and Eq. (17), we get the following conditions for Eq. (20) to hold:

$$\frac{1-\alpha}{N^{1-\alpha}-1} \leq \frac{1}{K} \text{ and } \frac{(1-\alpha)N^{-\alpha}}{N^{1-\alpha}-1} \geq \frac{1}{KM}.$$

As we show below, this limits the value that α, the skew in file request rate distribution, can take to $\leq 1+\frac{1}{K}$ for the derived $Q^{\text{linear}}/Q^{\text{opt}}$ ratio to hold. This limit on α implies that the near-linear increase in the $Q^{\text{linear}}/Q^{\text{opt}}$ ratio with increase in N as [8] provides for highly skewed file request rate distribution needs to be re-evaluated.

Case 1: α < 1 and large N,

$$\frac{1-\alpha}{N^{1-\alpha}-1} \leq \frac{1}{K}$$

This should be trivially satisfied for large N.

$$\frac{(1-\alpha)N^{-\alpha}}{N^{1-\alpha}-1} \geq \frac{1}{KM} \Rightarrow \frac{(1-\alpha)N^{1-\alpha}}{N^{1-\alpha}-1} \geq \frac{N}{KM} \Rightarrow (1-\alpha) \geq \frac{N}{KM} \Rightarrow \alpha \leq (1-\frac{N}{KM})$$

For a large peer-to-peer system (i.e. large M) where each node can store only a small fraction of total number of files (i.e. N/K = $1/f$ where f is a small constant fraction less than 1), this condition should also be satisfied.

Case 2: α > 1 and large N,

$$\frac{1-\alpha}{N^{1-\alpha}-1} \leq \frac{1}{K} \Rightarrow \frac{(\alpha-1)N^{\alpha-1}}{N^{\alpha-1}-1} \leq \frac{1}{K} \Rightarrow (\alpha-1) \leq \frac{1}{K} \Rightarrow \alpha \leq 1+\frac{1}{K}$$

Thus, when the file request rate distribution is extremely skewed (i.e. $\alpha > 1+\frac{1}{K}$) some of the files will be populated at all nodes and, hence, the $Q^{\text{linear}}/Q^{\text{opt}}$ ratio is no longer given by Eq. (20) and needs to be re-calculated.

$$\frac{(1-\alpha)N^{-\alpha}}{N^{1-\alpha}-1} \geq \frac{1}{KM} \Rightarrow \frac{(\alpha-1)}{N^{\alpha}-N} \geq \frac{1}{KM} \Rightarrow \approx \frac{(\alpha-1)}{N^{\alpha-1}-1} \geq \frac{N}{KM}$$

For arbitrarily large N, the denominator on the l.h.s will be too large and, hence, this second condition requires that α be not very large. However, a small α decreases the numerator on the l.h.s. so this condition bounds α in a range. Since, calculating this range is difficult and we already have the $\alpha \leq 1+\frac{1}{K}$ upper bound on α, let us evaluate the second condition in this restricted range of α. For $\alpha = 1+\frac{1}{K}$, the second condition becomes $\frac{1}{K} \geq \frac{N(N^{1/K}-1)}{KM} \Rightarrow N(N^{1/K}-1) \leq M$. For a large peer-to-peer system (i.e. large M), this induced limit on N may not be unduly restrictive and, hence, we can consider

$$\alpha \leq 1+\frac{1}{K} \qquad (21)$$

to be the range in which the $Q^{\text{linear}}/Q^{\text{opt}}$ ratio we obtained earlier should hold.

Let us now check if the remaining case of α = 1 adds any further conditions on N.

Case 3: α = 1 and large N,

Since, for α = 1, $\lambda = \sum_{i=1}^{N} \lambda_i \approx lnN$, we obtain:

$$\frac{1}{\ln N} \leq \frac{1}{K} \Rightarrow \ln N \geq K$$

This condition should hold for large N.

$$\frac{N^{-1}}{\ln N} \geq \frac{1}{KM} \Rightarrow N \ln N \leq KM$$

For a large peer-to-peer system (i.e. large M) where each node can store only a small fraction of total number of files (i.e. $K/N = f$ where f is a small constant fraction less than 1), this condition reduces to $\ln N < M$ and should also be satisfied.

Thus, we need to re-calculate the $Q^{\text{linear}}/Q^{\text{opt}}$ ratio for the case where $\alpha > 1+\frac{1}{K}$.

Notice that for the $n_i \propto \sqrt{\lambda_i}$ replica distribution to hold for all i, $\frac{1}{KM} \le \frac{\sqrt{\lambda_i}}{\sum_{i=1}^{N}\sqrt{\lambda_i}} \le \frac{1}{K}$ $\forall i$, and hence, conditions similar to Eq. 21 can now be derived for the square-root proportionality replica distribution as well. Since $\lambda_i = i^{-\alpha}$, in deriving the conditions for the $n_i \propto \sqrt{\lambda_i}$ replica distribution, we would be replacing λ_i with $\sqrt{\lambda_i}$ in the derivation of conditions for $n_i \propto \lambda_i$. Hence, we just need to replace α by $\alpha/2$ in the derivation for $n_i \propto \lambda_i$. Thus, the limits on α for $n_i \propto \sqrt{\lambda_i}$ are twice the corresponding values for the $n_i \propto \lambda_i$ replica distribution. Specifically, for $\alpha > 2 + 2/K$, some files are populated at all nodes for the $n_i \propto \sqrt{\lambda_i}$ replica distribution as well.

Let us restrict our attention to the $1 < \alpha < 2$ range and re-calculate the $Q^{\text{linear}}/Q^{\text{opt}}$ ratio in this range first.

Since $\alpha < 2$, the Q^{opt} expressions in Eq. 19 still hold. When $\alpha > 1 + \frac{1}{K}$, some of the files will be populated at all the nodes for the linearly proportional replica distribution. Let us say that $j-1$ files get populated at all nodes. The query-processing load when the number of replicas for all files follows $n_i \propto \lambda_i$ was N/K where N was the number of files involved in the search and K was the per-node storage available to store these files. Therefore, the query-processing load for the remaining files in our case is $(N-j+1)/K-j+1)$ since only $K-j+1$ locations remain available in the per-node caches for the $N-j+1$ files that still remain involved in the search process. Since the requests for these files account for only a fraction $\frac{\sum_{i=j}^{N}\lambda_i}{\sum_{i=1}^{N}\lambda_i}$ of the total number of file requests, the query-processing load for the $n_i \propto \lambda_i$ replica distribution $Q^{\text{linear}} = \frac{\sum_{i=j}^{N}\lambda_i}{\sum_{i=1}^{N}\lambda_i}\frac{N-j+1}{K-j+1}$. Therefore, to calculate Q^{linear} when $\alpha > 1 + \frac{1}{K}$, we need to determine $j-1$, the number of files that get populated at all the nodes.

Based on the available per-node storage, the number of files and the number of nodes, we can calculate $j-1$ for a given α by removing as many of the high request rate files, beginning with the file with the highest request rate, as necessary until the $\frac{\lambda_i}{\lambda} \le \frac{1}{K}$ condition (computed for the remaining files in the search process) that was violated when $\alpha > 1 + \frac{1}{K}$ is no longer violated i.e. the files are removed from the search process (and populated at all nodes) as long as the condition (with the re-computed values of $\frac{\lambda_i'}{\lambda'}$ and $\frac{1}{K'}$) is violated. The process stops as soon as the condition is no longer violated. Therefore, if $j-1$ files are populated at all nodes, then the maximum file request rate is $\lambda_j = j^{-\alpha}$ and only $K-j+1$ locations remain available in the per-node caches. Hence, the value of j can be derived from the $\frac{\lambda_i'}{\lambda'} \approx \frac{1}{K'}$ condition. Therefore,

$$\frac{j^{-\alpha}}{\sum_{i=j}^{N}\lambda_i} \approx \frac{1}{K-j+1} \qquad (22)$$

Since $\sum_{i=j}^{N}\lambda_i \approx \int_j^N x^{-\alpha}dx = \frac{N^{1-\alpha}-j^{1-\alpha}}{(1-\alpha)} = \frac{1-(j/N)^{\alpha-1}}{j^{\alpha-1}(\alpha-1)}$, $\frac{j^{-\alpha}}{\sum_{i=j}^{N}\lambda_i} \approx \frac{(\alpha-1)}{j-N(j/N)^{\alpha}}$. Hence, $\frac{(\alpha-1)}{j-N(j/N)^{\alpha}} \approx \frac{1}{K-j+1}$. While this relation is not helpful in analytically computing the value of j, we can use Eq. 22 to compute the desired Q^{linear} expression.

$$Q^{\text{linear}} = \left(\frac{\sum_{i=j}^{N}\lambda_i}{\sum_{i=1}^{N}\lambda_i}\right)\left(\frac{N-j+1}{K-j+1}\right) = \left(\frac{\sum_{i=j}^{N}\lambda_i}{K-j+1}\right)\left(\frac{N-j+1}{\sum_{i=1}^{N}\lambda_i}\right).$$

From Eq. 22, $j^{-\alpha} \approx \frac{\sum_{i=j}^{N}\lambda_i}{K-j+1}$. Therefore

$$Q^{\text{linear}} \approx j^{-\alpha}\left(\frac{N-j+1}{\sum_{i=1}^{N}\lambda_i}\right) = \frac{N-j+1}{j^{\alpha}\lambda}$$

Therefore, using Eqs. 16 and 18, for $(1+1/K) < \alpha < 2$, we obtain:

$$Q^{\text{linear}}/Q^{\text{opt}} \approx \left(\frac{N-j+1}{j^{\alpha}\lambda}\right)\left(\frac{\lambda K}{(\sum_{i=1}^{N}\sqrt{\lambda_i})^2}\right) \approx \frac{(N-j+1)K}{j^{\alpha}\left(\dfrac{N^{1-\alpha/2}-1}{1-\alpha/2}\right)^2} = \frac{(N-j+1)K}{j^{\alpha}(N^{2-\alpha}-2N^{1-\alpha/2}+1)}(1-\alpha/2)^2.$$

Therefore, for large N, we can approximate $Q^{\text{linear}}/Q^{\text{opt}}$ as:

$$Q^{\text{linear}}/Q^{\text{opt}} \approx \frac{(N-j+1)K}{j^{\alpha}N^{2-\alpha}}(1-\alpha/2)^2 = \frac{N-j+1}{N}\left(\frac{N}{j}\right)^{\alpha}\frac{K}{N}(1-\alpha/2)^2 \qquad (23)$$

Since j is much smaller than N for large N, $N{-}j{+}1 \approx N$ and, $Q^{\text{linear}}/Q^{\text{opt}} \approx \dfrac{(1-\alpha/2)^2 K}{[\alpha j-(\alpha-1)(K+1)]} = \dfrac{K}{N}\left(\dfrac{N}{j}\right)^{\alpha}(1-\alpha/2)^2$. While, K/N, the fraction of total number of files each node can store, is a fraction much smaller than 1, $\left(\dfrac{N}{j}\right)^{\alpha}$ can be fairly large as j will be much smaller than N. Therefore, $Q^{\text{linear}}/Q^{\text{opt}}$, i.e. the gain in using the $n_i \propto \sqrt{\lambda_i}$ replica distribution instead of the $n_i \propto \lambda_i$ replica distribution can still be large even after we account for saturation.

Since the inclusion of the possibility of the high request rate files being populated at all nodes does not imply a significant decrease in the $Q^{\text{linear}}/Q^{\text{opt}}$ ratio in the $(1+1/K) < \alpha < 2$ range compared to Eq. (20), it is unnecessary to conduct such an analysis for the $\alpha \geq 2$ range and, we can conclude that the gain factor for the $n_i \propto \sqrt{\lambda_i}$ replica distribution is fairly large when the file popularities are very skewed.

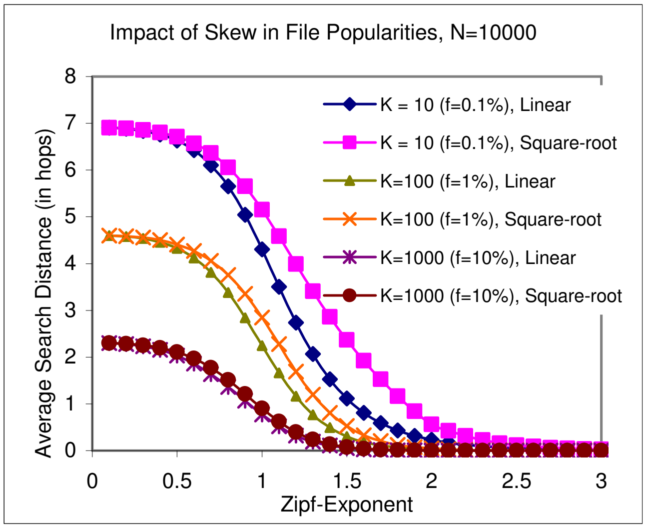
However, when the file request rates are not very skewed (i.e. when $\alpha < 1$), the gain factor with the $n_i \propto \sqrt{\lambda_i}$ replica distribution shows a very different behavior than this high query-processing load disadvantage with the $n_i \propto \lambda_i$ replica distribution for large α. When, $\alpha < 1$, as given in Eq. 20, the gain factor is independent of N and limited to $\dfrac{(1-\alpha/2)^2}{1-\alpha}$ which is fairly reasonable. A number of measurement studies of real peer-to-peer file sharing systems and web traffic estimate the value of α to be between 0.8 and 1 [5, 27] and for $\alpha = 0.8$, the gain factor will be only 1.8 while for $\alpha = 0.9$, it will be only 3.025.

Thus, when the file popularities are not very skewed, one may use the $n_i \propto \lambda_i$ replica distribution. However, before we can say that it would be better to use the $n_i \propto \sqrt{\lambda_i}$ replica distribution when the file popularities are extremely skewed, we must convince ourselves that the additional search time incurred with the $n_i \propto \sqrt{\lambda_i}$ replica distribution is not too high.

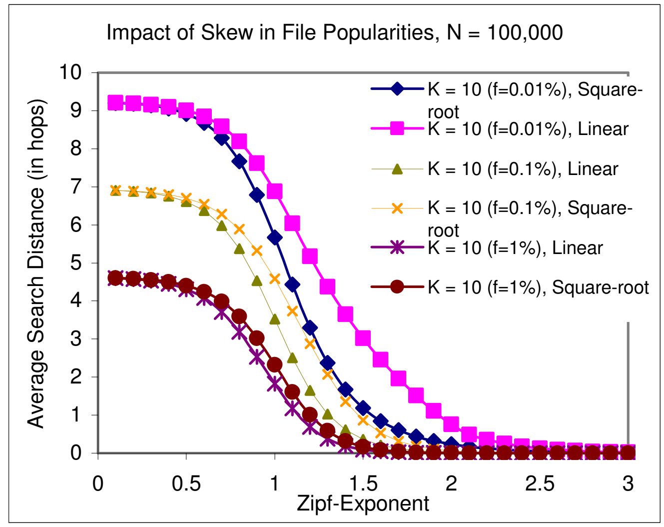
For $n_i = \dfrac{KM\sqrt{\lambda_i}}{\sum_{i=1}^{N}\sqrt{\lambda_i}}$, $\tau_{\text{sqrt}} = -\alpha\sum_{i=1}^{N}\dfrac{\lambda_i}{\lambda}\, log_{\text{d}}(\dfrac{\sqrt{\lambda_i}}{\sum_{i=1}^{N}\sqrt{\lambda_i}}K) = -0.5\alpha\sum_{i=1}^{N}\dfrac{\lambda_i}{\lambda}\, log_{\text{d}}(\lambda_i) + \alpha\, log_{\text{d}}(\sum_{i=1}^{N}\sqrt{\lambda_i}) - \alpha log_{\text{d}}(K)$. In contrast, the $n_i \propto \lambda_i$ replica distribution gives $-\alpha\sum_{i=1}^{N}\dfrac{\lambda_i}{\lambda}\, log_{\text{d}}(\lambda_i) + \alpha log_{\text{d}}\lambda - \alpha log_{\text{d}}K$. Thus, the extra cost is $0.5\alpha\sum_{i=1}^{N}\dfrac{\lambda_i}{\lambda}\, log_{\text{d}}(\lambda_i) + \alpha\, log_{\text{d}}(\sum_{i=1}^{N}\sqrt{\lambda_i}) - \alpha log_{\text{d}}\lambda = 0.5\alpha[\sum_{i=1}^{N}\dfrac{\lambda_i}{\lambda}\, log_{\text{d}}\dfrac{\lambda_i}{\lambda} + log_{\text{d}}(\dfrac{(\sum_{i=1}^{N}\sqrt{\lambda_i})^2}{\lambda})]$. Therefore, when the entropy is highest (i.e. all files have same request rates), the two terms cancel each other and there is no extra cost in using the $n_i \propto \sqrt{\lambda_i}$ replica distribution (which is obvious since all files have an equal number of replicas with either of the two distributions). Further, when the entropy goes to 0, the extra cost again goes to 0 (since, now we have only one file in the system and both distributions populate it at all nodes). Thus, the extra cost in search time with the $n_i \propto \sqrt{\lambda_i}$ replica distribution is maximized between these two extremes. As with our analysis of the query-processing loads at these distributions, we need to account for the possibility of the high-request rate files getting populated at all nodes.

Since obtaining closed-form analytical expressions is difficult, we choose to numerically compute the ratio of the search time with the $n_i \propto \sqrt{\lambda_i}$ replica distribution to the search time with the $n_i \propto \lambda_i$ replica distribution for different cache sizes and varying levels of skew in file popularity distributions. Figure 8 shows the average search distance with the $n_i \propto \lambda_i$ and the $n_i \propto \sqrt{\lambda_i}$ replica distributions for different cache sizes and different zipf-exponent values – sub-figure (a) shows the results for 10,000 files while sub-figure (b) shows it for 100,000 files.

Figure 9 shows the ratio of the average search distance with the $n_i \propto \sqrt{\lambda_i}$ replica distribution to the average search distance with the $n_i \propto \lambda_i$ replica distribution (on primary y-axis) for different cache sizes and different zipf-exponent values – sub-figure (a) shows the results for 10,000 files while sub-figure (b) shows it for 100,000 files. Since our analysis of the ratio of the query-processing load with the $n_i \propto \lambda_i$ replica distribution to the query-processing load with the $n_i \propto \sqrt{\lambda_i}$ replica distribution did not yield explicit expressions when some of the files get populated at all nodes, we also computed the ratio of the query-processing load with the $n_i \propto \lambda_i$ replica distribution to the query-processing load with the $n_i \propto \sqrt{\lambda_i}$ replica distribution and show it alongside the search time ratio on the secondary axis in Figure 9.

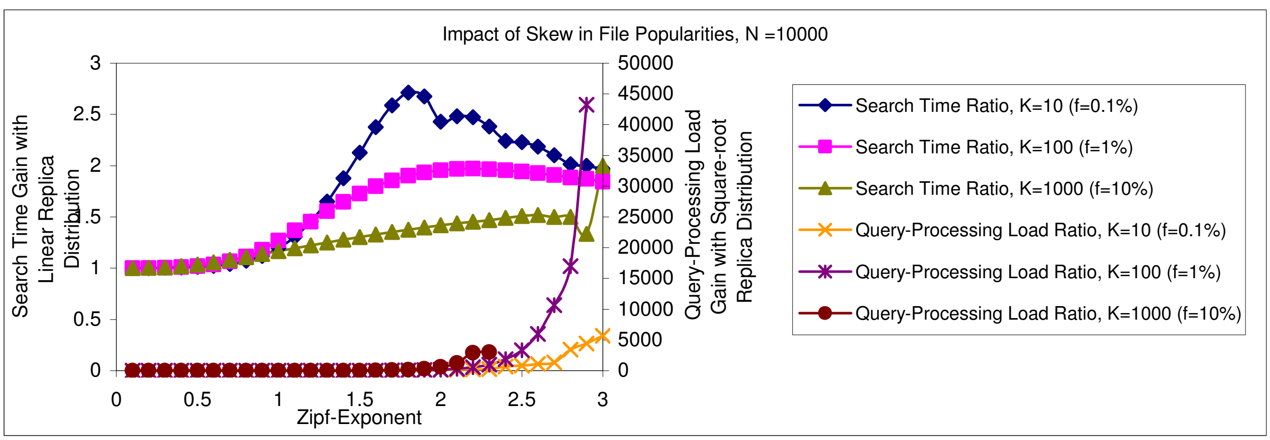


(a) 10,000 unique files

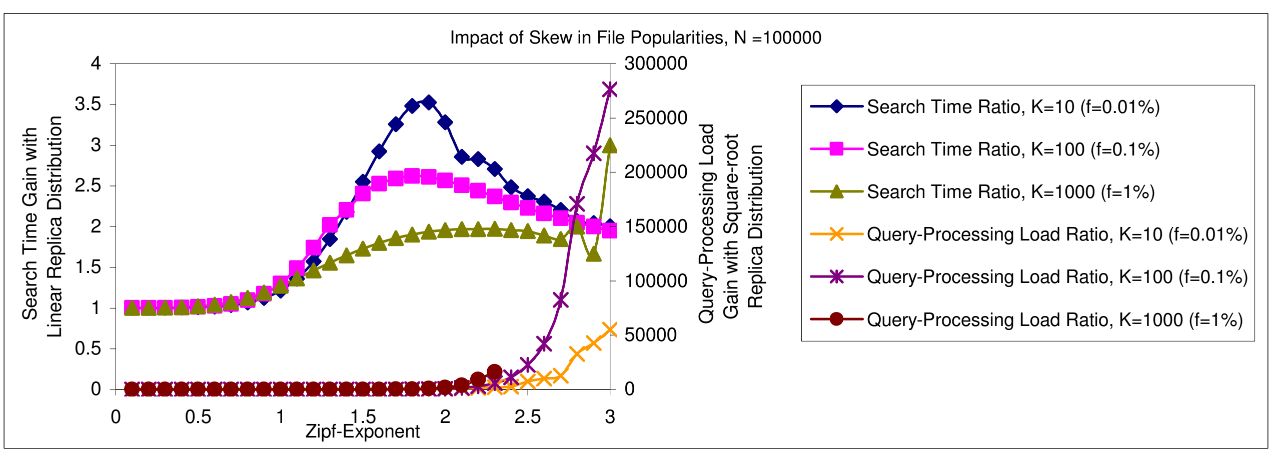


(b) 100,000 unique files

Figure 8. Average Search Distance for the $n_i \propto \lambda_i$ and the $n_i \propto \sqrt{\lambda_i}$ replica distributions with varying skew in file popularities and cache sizes



(a) 10,000 unique files



(b) 100,000 unique files

Figure 9. Search Time and Query-Processing Load ratios of the sub-optimal to the optimal performance with varying skew in file popularities and cache sizes

We can see from Figure 9 that while the average search time only triples or quadruples (depending on the fraction of files each node can store) when using the $n_i \propto \sqrt{\lambda_i}$ replica distribution, the query-processing load becomes extremely high when the $n_i \propto \lambda_i$ replica distribution is used instead if the file popularities are very skewed. It is questionable, though, if real file popularities are indeed as skewed as indicated by the zipf-exponent of 3. Notice that the disadvantage in the query-processing load with the $n_i \propto \lambda_i$ replica distribution increases very rapidly only beyond zipf-exponent of 2.5. There have been a number of studies of peer-to-peer traffic that indicate skew in file popularities. However, the reported skew needs to be adjusted as the measurements only capture the requests that are broadcast (requests for files stored locally are not broadcast to the network and, hence, are absent from the reported measurements). Upon including the requests for files previously requested, [13] estimated the zipf-exponent to be 1. Zipf-exponent of 1 has also been found sufficient to capture the skew in web requests [5]. Therefore, in deciding which replica distribution to use, we should consider the ratio at or around zipf-exponent of 1 instead of 2 or 2.5. Since these numbers are not easily observed in Figure 9, we re-draw the figure with only the relevant range of zip-exponent values in Figure 10.

As shown in Figure 10, even when the cache size is 0.01% of all files, at zipf-exponent of 1, the query-processing load increases by a factor of 3 by using the $n_i \propto \lambda_i$ replica distribution instead of the $n_i \propto \sqrt{\lambda_i}$ replica distribution. Only in cases where the per-node storage is even smaller and the file popularity distribution is even more skewed would one might prefer to have the $n_i \propto \sqrt{\lambda_i}$ replica distribution (and pay the extra cost in the search time and the download time).

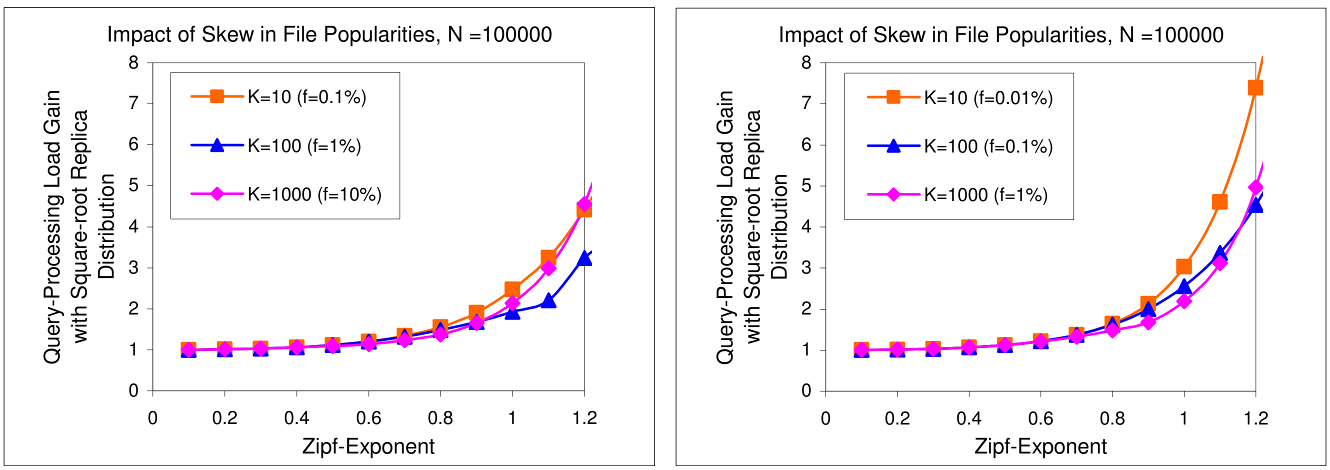


Figure 10. Query-Processing Load ratio for realistic file popularity skew

Another approach to deal with the dichotomy between the optimal replica distribution for the download time and the query-processing load is to note that searches only require information on where a file is stored, not the file itself (unlike downloads where you obviously need the file). Therefore, if a system could be devised whereby one can store pointers to files in square-root proportionality but store the files themselves in linear proportionality, we would have the advantages of both the distributions. Keeping file pointers updated will be a problem with such an approach since peers in typical peer-to-peer systems are known to have short lifetimes [27] unless the pointers are kept only one hop away from the files (which limits the ability to achieve square-root proportionality in file pointers) or are kept at well-defined locations (e.g. in a hierarchical structure) within few hops. Depending on the system dynamics, either of the two strategies can be used. Even though full benefits of square-root proportionality would not be achieved, boosting the number of pointers to the low request-rate files will improve the overall performance.

5. COMPARISON WITH OPTIMAL RANDOM WALK SEARCH

Even though flooding search is the fastest way to locate a file in the absence of any hints on file location, it has been argued that random walk search is superior to flooding search [6, 11, 22, 28, 32] since flooding has a high overhead. The argument contends that the query traffic increases exponentially with increasing hop distance and, at high hop distance, many of these messages are going to nodes that have already been probed. However, as discussed in section 3.1, given the same file replica distribution, the query-processing load for controlled flooding search will be same as the query-processing load for random walk since both methods need to probe the same number of nodes, on average, to find the file. The average search time for controlled flooding search, on the other hand, *is logarithmically smaller* than that for random walk search (as observed in Section 3, flooding search probes the same number of nodes in logarithmic number of rounds) given the same file replica distribution.

Since random walk probes one node at a time, the search time for random walk is same as the query-processing load. Hence, the random walk search time is minimized with the $n_i \propto \sqrt{\lambda_i}$ replica distribution and the optimal search time for random walk is same as the optimal query-processing load i.e.

$$\tau_R^{\text{opt}} = \frac{(\sum_{i=1}^{N}\sqrt{\lambda_i})^2}{\lambda K} \qquad (24)$$

Since the controlled flooding search time is minimized under the $n_i \propto \lambda_i$ replica distribution, if one were to compare random walk with controlled flooding under their respective optimal distributions, the query-processing load for controlled flooding search would be higher than that for random walk search. However, as we found in the previous section, the two distributions result in comparable query-processing loads for typical file popularity distributions.

As we will show next, the average search time for controlled flooding search remains *logarithmically smaller* than that for random walk search even when the two methods are operating under their respective optimal replica distributions.

We wish to show that

$$\tau_F^{\text{opt}} < \alpha log_{\text{d}} \tau_R^{\text{opt}}$$

Since search time cannot be negative, exponentiation will not change the sign of the inequality relation. Hence, it is sufficient to show that

$$d^{\tau_F^{opt}} < (\tau_R^{\text{opt}})^{\alpha}$$

Using Eq. (14) in Section 4.3, we can compute the $d^{\tau_F^{opt}}$ as:

$$d^{\tau_F^{opt}} = d^{-\alpha(\sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\log_d\frac{\lambda_i}{\lambda}) - \alpha\log_d K}$$

$$= (\frac{\text{d}^{-(\sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\log_d\frac{\lambda_i}{\lambda})}}{K})^{\alpha}$$

$$= (\frac{\prod_{i=1}^{N}(\frac{\lambda_i}{\lambda})^{-\frac{\lambda_i}{\lambda}}}{K})^{\alpha} \qquad (25)$$

We now use the theorem of the arithmetic and geometric means [14]:

$$a_1^{p_1} a_2^{p_2} ... a_n^{p_n} < (\frac{p_1 a_1 + ... + p_n a_n}{p_1 + ... + p_n})^{p_1 + ... + p_n}$$

Setting $a_i = \frac{1}{\sqrt{p_i}}$, we get: $\prod_{i=1}^{n}(\frac{1}{\sqrt{p_i}})^{p_i} < \left(\sum_{i=1}^{n} p_i \frac{1}{\sqrt{p_i}} \Big/ \sum_{i=1}^{n} p_i\right)^{\sum_{i=1}^{n} p_i}$. Applying $\sum_{i=1}^{n} p_i = 1$ and $p_i \frac{1}{\sqrt{p_i}} = \sqrt{p_i}$, we get:

$$\prod_{i=1}^{n}(\frac{1}{p_i})^{\frac{p_i}{2}} < (\sum_{i=1}^{n}\sqrt{p_i})^1.$$

Since both sides are positive, taking square of both sides does not change the sign of the inequality and setting $p_i = \lambda_i/\lambda$, we get:

$$\prod_{i=1}^{N}(\frac{\lambda_i}{\lambda})^{-\frac{\lambda_i}{\lambda}} < (\sum_{i=1}^{N}\sqrt{\frac{\lambda_i}{\lambda}})^2$$

Dividing both sides by K, taking the λ out from under the square-root on the r.h.s and raising both sides to the power α, we get:

$$(\frac{\prod_{i=1}^{N}(\frac{\lambda_i}{\lambda})^{-\frac{\lambda_i}{\lambda}}}{K})^{\alpha} < (\frac{(\sum_{i=1}^{N}\sqrt{\lambda_i})^2}{\lambda K})^{\alpha}$$

or, using Eqs. (24) and (25),

$$d^{\tau_F^{opt}} < (\tau_R^{\text{opt}})^{\alpha}$$

Taking log on both sides, we have:

$$\tau_F^{\text{opt}} < \alpha log_{\text{d}} \tau_R^{\text{opt}} \qquad (26)$$

Thus, flooding search takes exponentially less time than the random walk search with each operating under their respective optimal replica distributions.

Even if flooding search was operated under the $n_i \propto \sqrt{\lambda_i}$ replica distribution (so that the query-processing load were equal with either method), the flooding search will take logarithmically smaller than the random walk search (since we found that using the $n_i \propto \sqrt{\lambda_i}$ replica distribution with controlled flooding only increases the search time by a factor of 3 or 4 depending on the per-node cache size).

As we already noted in the previous section, the worst-case query-processing load with the $n_i \propto \sqrt{\lambda_i}$ replica distribution is N/K, both controlled flooding and random walk, each operating under their respective optimal replica distributions have the same worst-case query-processing load.

Table 1 summarizes the different performance metrics for Controlled Flooding search and Random Walk search. While it was known that flooding is faster, random walk search was proposed to control the search traffic overhead so our query-processing load observations are significant. Even though the random walk traffic grows slowly, it seems that in the absence of any hint of where to go next, the walk can take many hops before it finds the file, possibly even querying a previously queried node again. These problems are a consequence of independent sampling that the random walk search approximates. Augmented random walks [28, 32] should perform better although, as noted earlier in this paper and elsewhere [6], storing location pointers farther than one hop away is a risky strategy in dynamic networks.

Table 1. Search Performance Summary at Search Time Optimal Replica Distribution (all quantities are per-query)

Metric	Controlled Flooding	Random Walk	Controlled Flooding is
Average Search Distance*	$\alpha \log_d(M/n_i)$	$(M/n_i) - 1$	Logarithmically better
Optimal Replica Distribution	$n_i \propto \lambda_i$	$n_i \propto \sqrt{\lambda_i}$	-
Average Search Distance	$-\alpha\sum_{i=1}^{N}\frac{\lambda_i}{\lambda}\log_d\frac{\lambda_i}{\lambda} - \alpha\log_d K$	$\frac{(\sum_{i=1}^{N}\sqrt{\lambda_i})^2}{\lambda K}$	Logarithmically better Eq. (23)
Query-Processing Load	$\frac{N}{K}$	$\frac{(\sum_{i=1}^{N}\sqrt{\lambda_i})^2}{\lambda K}$	Comparable if file popularity distribution not very skewed
Worst-case Query-Processing Load	$\frac{1}{K/N}$	$\frac{1}{K/N}$	Equal

** These expressions are valid for arbitrary replica distributions*

6. DISCUSSION

We made some simplifying assumptions in our model to facilitate the analysis. In the latter part of this section, we discuss how our analysis may be affected when these simplifying assumptions are substituted by phenomenon measured and/or expected in real peer-to-peer systems. A critical assumption in our analysis is our controlled-flooding search model. This is discussed in Section 6.1. In Sections 4.3 and 5, we assumed that the replica distribution is optimal; in Section 6.2, we discuss whether the number of replicas can be driven to the optimal (or near optimal) distribution in a distributed manner.

6.1 Practical Controlled Flooding

Our analysis assumes that the query flood stops at the hop where the first replica is found. However, a query flood with no control mechanism may not terminate even if the file is found since the nodes other than the "query hit" node keep forwarding the query oblivious of the "query hit". Usually, a TTL field in the query message is used to limit the query flood to a certain number of hops. Unfortunately, using this technique to stop the flood at the hop distance where the nearest replica is located requires one to know this hop distance in advance of making the query. However, variations of this approach can achieve the desired termination behavior.

One such practical method that has desired termination behavior (albeit at the cost of an increase in the search time) is the *Expanding Rings* method [22]. In an Expanding Rings search, the querying node makes repeated queries with increasing TTL values. Even though this approach incurs additional search time and repeated probing of the nodes within the "last ring", the performance advantage over random walk search still exists. The query-processing load with this method is $\sum_{i=1}^{h} d^i = [d^{h+1} - 1]/[d-1]$, which for large d, is still approximated as d^h. Thus, the relation between the query-processing loads for controlled flooding and random walk will also apply to the Expanding Rings method. However, the search time with this method can be estimated as $\sum_{i=1}^{h} i = h(h+1)/2$ and, hence, the search time relation between controlled flooding search and random walk search will change to $\tau_F^{\text{opt}} \sim O(\log^2 \tau_R^{\text{opt}})$ instead of $O(\log \tau_R^{\text{opt}})$ which is still significantly better for very large networks.

Our model differs from some of the deployed peer-to-peer file sharing systems in two other aspects. The deployed systems tend to return a query hit over the path taken by the query while we assume the query hit returns directly to the querying node. However, even if the query hit is not returned directly, the effect on the query-processing load calculation will be negligible (a linear term is being added to an exponential term). The other difference is that many of these deployed systems prefer to obtain multiple file sources whereas, in our model, a search stops as soon as one source is found. Notice, however, that if the search proceeds to more than a few hops, flooding is likely to return multiple replicas as the number of nodes queried increases exponentially with every additional hop. Besides, while downloading from multiple sources may be better from the point of view of a single user, it is not ideal for the overall system performance [10, 17].

6.2 Achieving the Optimal Replica Distribution

Measuring file request rates and then populating the network with a proportional (i.e., optimal) number of replicas by a central mechanism is not practical. The optimal (or near optimal) replica distribution should rather be attained in a distributed manner. To check the viability of such an approach, we simulated a very simple peer-to-peer system where, in addition to the properties defined for our peer-to-peer system model of Section 3, the finite nodal storage space implies that a previously obtained file must be deleted if space is needed for a newly requested file. The file replacement policy was LRU (except that the last replica of a file is never deleted). Our simulation model started with a single replica of every file and file requests and searches are simulated for an extended period of time. For simplicity, each node had identical request rates for each file. We found that, after an initial transition period, the number of replicas of a file stabilizes (even though the file locations may be changing) to a replica distribution linearly proportional to the file request rates when the request rates for different files are uniformly distributed between the lowest and the highest file request rates (see Figure 11). Only when the request rate distribution is highly skewed (e.g. Zipf with exponent 1, 1.5[7]), the files with higher request rates have fewer than the linearly proportional number of replicas. Notice that even when the skew in file request rates is so large that the optimal replica distribution is now defined by Eq. (9) instead of Eq. (8), LRU file replacement still achieves near optimal replica distribution.

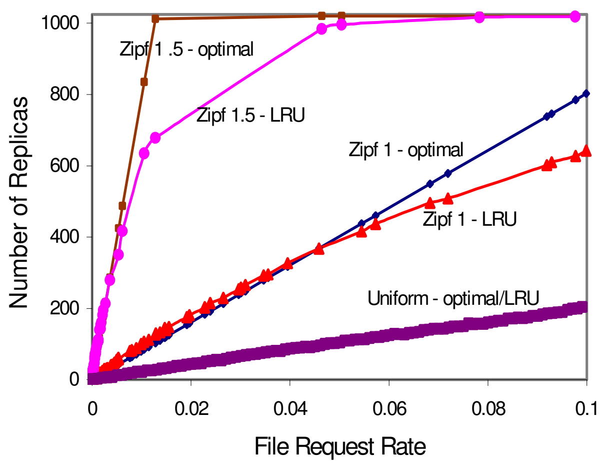


Figure 11. Replica distribution in an example peer-to-peer system with LRU file replacements

Given that such a simple system "naturally" resulted in a near optimal distribution, we are optimistic that this system could be driven to the optimal replica distribution with distributed algorithms and we try several of them next. The results for these local storage management algorithms are shown in Figure 12.

Since a peer serves file requests from other nodes, the traffic from other nodes affects the LRU-ranking. A simple practical modification to the basic LRU algorithm is to ignore the traffic from other nodes for LRU ranking. We call this algorithm *modified-LRU*. Thus, in Modified-LRU, the requests from other nodes are fulfilled but those request do not alter the position of any file in the LRU stack. As we can see from Figure 12, the modified-LRU does better than the basic LRU. However, it still does not give linear proportionality.

Next, we try using the *frequency* of file accesses instead of the *recency* of file accesses used in LRU. Perfect-LFU (Least Frequently Used), where the file with the smallest probability of being requested is removed when the system needs to create a space in the storage, is known to be the optimal policy under the Independence Reference Model (i.e. independent requests) and uniform cost of accessing each object [36]. Since the probability of a file being requested is not known in real systems, Perfect-LFU is not a practical algorithm. Least Frequently Used (LFU) algorithm is an approximation of perfect-LFU that keeps a count of the number of references to each file in the storage and removes the file with the lowest count whenever it needs to create a space in the storage. As shown in Fig. 12, LFU gets the number of replicas distribution closer to the optimal distribution than the modified-LRU or the LRU algorithm. However, the number of replicas of the high request rate files is still below the desired linear proportionality distribution. Since, in simulation, we know the probability of each file being requested (it is equal to the file request rate), we also try the Perfect-LFU algorithm. As shown in Figure 12, the Perfect-LFU does not perform well. It results in only the most popular files being stored which would cause a large increase in the search distances for the less popular files.

[7] As discussed earlier, a zipf-exponent of 1 is sufficient to capture the skew in file request rate distribution.

Since our LRU modification improves performance, we can try the same modification on LFU. The modified-LFU is an LFU where the requests from the other nodes do not increment the frequency count of a file. As shown in Fig. 12, the modification does create more replicas of the high request rate files but the lower request rate files have much fewer than the optimal number of replicas.

Since both the recency-based algorithms (LRU and its variation) and the frequency-based algorithms (LFU and its variation) give close to, but not exactly, the linear distribution, we also try the FBR (Frequency-based Replacement) algorithm [37] which incorporates both the frequency and the recency in deciding the file to be replaced. As shown in Fig. 12, the FBR does not improve upon LFU.

For completeness, we also try the FIFO (First-In First-Out) and Random-Delete (when space needs to be made in the cache, one of the files is randomly selected for deletion) algorithms. As shown in Fig. 12, these algorithms perform worse than the other schemes, which choose the file to be replaced more "intelligently".

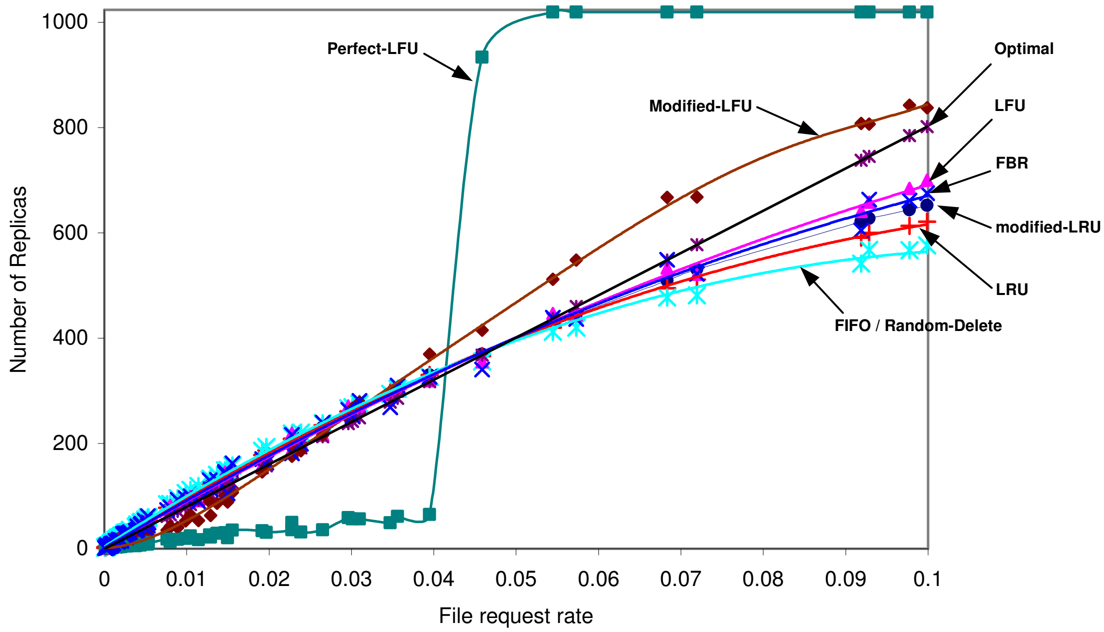


Figure 12. Number of Replicas with different storage management policies (File request rates Zipf-distributed with zipf-exponent 1)

Thus, none of the known local storage management algorithms (and their minor variations) give us the desired linear proportionality. However, an equilibrium analysis of Random-Delete provides insights into possible ways to find approaches that may lead to linear proportionality in the number of replicas.

Each file request creates an additional replica of the file if was not available in the local cache (since that file is now brought into the cache). Thus, the replicas for file *i* are created at a rate $\lambda_i(1-n_i/M)$. With random deletion, the file removal rate is proportional to the number of replicas of the file in the system. At equilibrium, the replica creation rate should equal the replica deletion rate. Thus, at the equilibrium distribution, we have:

$$\lambda_i(1-n_i/M) = \mathrm{C}n_i$$

where C is the proportionality constant. Thus, it is not surprising that Random-Delete does not give linear proportionality. However, we also note that if the replica creation rate was λ_i, linear proportionality can be achieved. Recall that the less-than λ_i creation rate is on account of an additional replica not being created when the file is available in local cache. To "correct" this deviation from the desired replica creation rate, we can create the additional replica at a different peer to compensate for the existence of the file in the local cache. This *Create-Extra+Random-Delete* algorithm was simulated and the results are shown in Figure 13. As expected, the populate-extra algorithm does give us the linear proportionality in number of replicas.

However, writing on other peers' caches is not an ideal approach and it would be better if we could adjust the file deletion rate to be proportional to $n_i(1-n_i/M)$. This can be achieved if the file to be deleted is not selected with uniform probability from the cache but is weighted according to $(1-n_i/M)$. Since, each node does not know the total number of replicas of a file in the network locally, we need to devise other methods to achieve the $(1-n_i/M)$ biasing in the file selection for deletion. We know that if the desired linear proportionality was achieved, $n_i/M = K\lambda_i/\lambda$. Therefore, if the global file request rates were known locally, we could devise an algorithm that selects the file to be deleted with probability proportional to $K\lambda_i/\lambda$, we may get the desired linear proportionality. The simulation results for this *DeleteProb_RequestRatesKnown* algorithm are shown in Figure 13. As expected, we do get the linear proportionality in number of replicas. Unfortunately, in real systems, it is difficult to obtain the global file request rates locally so this algorithm is not practical and the *Create-Extra+Random-Delete* algorithm is the only practical algorithm that we have for achieving the linear proportionality in number of replicas.

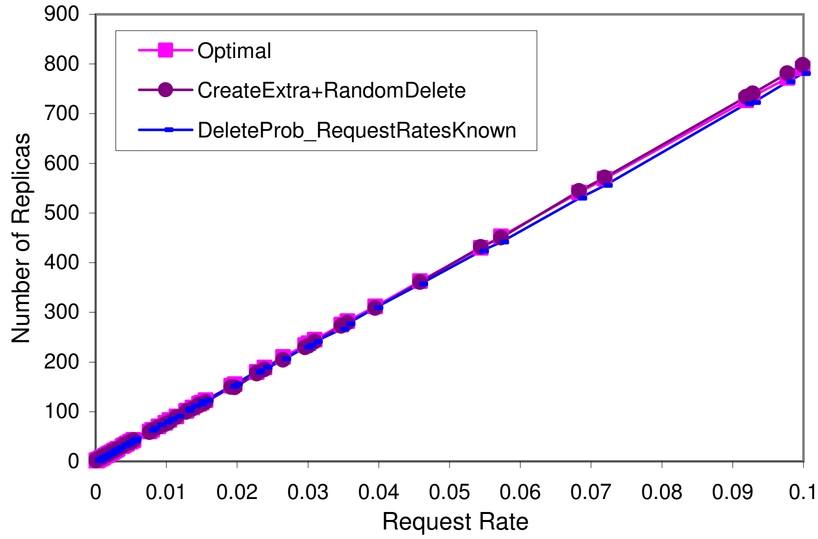


Figure 13. Distributed Algorithms for achieving linear proportionality (File request rates Zipf-distributed with zipf-exponent 1)

6.3 Discussion of Assumptions

6.3.1 Uniform Distribution of Replicas

Intuitively, one would expect this assumption to hold if the file request rates are the same at each node (i.e. $\lambda_{ij} = \lambda_i \; \forall j$). In our simulation model of Section 6.2, we also evaluated where the multiple replicas of a file were located. Starting at the original file source, we measured the fraction of nodes at each hop distance that have the file. This fraction stayed the same at each hop distance which suggests that the file replicas were uniformly distributed. To test the independence of this distribution to initial conditions, we also simulated starting with ten randomly placed replicas of each file, and found that after the initial transition period, the replica distribution was identical to the single initial source case, both in terms of number of replicas as well as the distribution of replicas.

In the real world, however, file interests are likely to vary between different geographic regions; this will lead to geographic clustering of replicas which is reported in the measurement studies of peer-to-peer file sharing networks [27]. While our model of uniform file distribution is not accurate for such a scenario, it should provide us the worst-case performance bounds as geographic clustering of file request rates and the file replicas will only improve performance. Depending on the degree of clustering in file interests, it may even be appropriate to treat different regional clusters as independent sub-networks with search/download traffic crossing over into a different region only rarely. Within the sub-networks, our model will apply (and if there was no cross-cluster traffic and the number of files requested within each cluster were about the same, the query-processing load will be a factor of L smaller than the query-processing load in the uniform distribution case). In the rare events of a cross-cluster file search, flooding might incur a high cost unless the search network topology had only a few cross-cluster links. While we believe that the effect of high cost of flooding in the rare events of cross-cluster search on the average query-processing load will not be substantial enough to offset the factor of L advantage for in-cluster search, to better understand the precise effect of clustering on query-processing load and search time, we would need to conduct analysis similar to the uniform distribution case. We hope to address this issue in our future work.

6.3.2 Other Topologies

Even though we discuss our results for a search network with a random graph topology, our results are applicable for a wider class of topologies. Our results on optimal replica distribution and search distance are based on logarithmic search distance while the query-processing load results also require an exponential expansion in number of nodes as the hop distance increases. As discussed in Section 3.2, the logarithmic search distance applies to other topologies including one obtained from an actual Gnutella2 trace as long as fewer than 10% of the nodes have the file. The power-law random graph, the super-peer network and the measured Gnutella2 topology also display the exponential expansion in number of nodes as the hop distance increases. It is unlikely that more than one-tenth of all requests will be for any one file (which is when the linear proportionality to request rates would require more than 10% of the nodes to have the file)[8]. Thus, our results should apply for these topologies as well.

Finally, even for topologies beyond those already discussed, we believe that our results are likely to apply to any search network topology where the number of nodes increases exponentially with the hop distance.

6.3.3 Unequal File Sizes

A search for a particular file is concerned with finding where the file is located. Thus, for search purposes, having a pointer to the file location is just as good as having the file if these file pointers could stay accurate. Even when we have files of different sizes, the file pointer sizes would still be the same. Thus, if we have accurate file pointers, all of our analysis using constant file size is applicable to variable file sizes.

However, if the node churn-rate is very high, ensuring accuracy of the file pointers may not be feasible and so let us explicitly incorporate the different file sizes in the optimization model.

Let c_i = the normalized[9] size of file i. Since the c_i*'s* are not equal, the total available storage size constraint (Eq. 5) becomes $\sum_{i=1}^{N} c_i n_i \le KM$. Our Lagrangian optimization now yields the optimal replica distribution for flooding search as:

$$n_i = (\lambda_i/\lambda c_i)KM \qquad (33)$$

At this *file-size-adjusted-linear* replica distribution, the minimum average search distance is:

$$\tau_F^{\text{opt}} = -\alpha \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \log_{\text{d}}\left(\frac{\lambda_i}{\lambda}\right) - \alpha\log_{\text{d}}K + \alpha \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \log_{\text{d}} c_i \qquad (34)$$

Comparing this to Eq. 14, we find that unequal file sizes introduce an extra $\alpha \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \log_{\text{d}} c_i$ term in the minimum average search distance. This term can take positive[10] as well as negative[11] values depending on the values of $\{c_i\}$ and $\{\lambda_i\}$. Thus we see that unequal file sizes may increase or decrease the minimum average search distance. If the larger-size files constitute the bulk of the requests, τ_F^{opt} increases since the replicas of the popular files cannot be stored at as many nodes; on the other hand, if the smaller-size files constitute the bulk of the requests then τ_F^{opt} will decrease since the replicas of the popular files can be stored at more nodes.

The query-processing load at this distribution, $Q^{\text{file-size-adjusted-linear}}$ is:

$$Q^{\text{file-size-adjusted-linear}} = \left(\sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \frac{M}{n_i}\right) = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \frac{M\lambda c_i}{\lambda_i KM} = \frac{1}{K} \sum_{i=1}^{N} c_i = \frac{N}{K} \qquad (35)$$

Thus, the query-processing load remains same as that in the equal file size case if the replica distribution is chosen to minimize the search time. We summarize this result in the following theorem:

---

[8] If it does happen that more than one-tenth of all requests are for one file, the faster than linear decrease in hop distance in the $(n_i/M) > 0.1$ range suggests that, in these topologies, the optimal solution is likely to have fewer than a linearly proportional number of replicas of the very popular files. For example, in a super-peer network topology, for search purposes, for even the most popular file, it is sufficient to have enough replicas that, with high probability, each super-peer has at least one leaf node with a replica of the file. Thus, for these topologies, the replica distribution attained by the LRU file replacement policy in our model of Section 6.2 is closer to the optimal replica distribution than we suggested them to be for our random graph topology.

[9] We normalize the file i.e. $(\sum_{i=1}^{N} c_i)/N = 1$. Since the average file size is unity, the per-node storage size can be described in units of the number of files each node can store and, ignoring the bin-packing problem, we can still say that the per-node storage size is K files.

[10] Maximizing $\alpha \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \log_{\text{d}} c_i$ subject to $\sum_{i=1}^{N} c_i = N$, we find that the term takes its maximum value when $c_i = N\lambda_i/\lambda$; this maximum value of $\alpha \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \log_{\text{d}}\left(\frac{\lambda_i}{\lambda}\right) + \alpha\log_{\text{d}}N$ yields $\tau_F^{\text{opt}}{}_{\max} = \alpha\log_{\text{d}}(N/K)$. This value is same as the maximum value of τ_F^{opt} in the equal file size case (Eq. 14) obtained by maximizing τ_F^{opt} w.r.t. λ_i.

[11] The term can take arbitrarily large negative values depending on the values of $\{c_i\}$ and $\{\lambda_i\}$. For example, let there be 1000 files in the system of which one file is of size 999 while the 999 others are of size 1/999 each. When all files have equal request rates, the file of size 999 contributes 0.001 fraction of the total request rate with the smaller 999 files contributing the remaining 0.999 fraction of the total request rate. In this case, $\alpha \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \log_{\text{d}} c_i = \alpha[0.001\log_{\text{d}}999 + 0.999\log_{\text{d}}(1/999)] = -0.998\alpha\log_{\text{d}}999$. If the request rates were skewed towards the smaller size files, this term would decrease the average search time even more.

Theorem 3:

Given an unstructured peer-to-peer network where the dependence of τ_i, the hop distance to the nearest replica, on n_i, the number of replicas, is of the form $\tau_i(n_i) = \alpha \log_d(M/n_i)$ and the storage capacity at each node is equal, then the average search distance in controlled flooding query searches is minimized when the number of replicas, n_i, of file i is proportional to the ratio of the file request rate λ_i to the file size c_i, (i = 1,2, …, N) i.e.

$$n_i = \alpha(\lambda_i/c_i)$$

and at this replica distribution, the query-processing load is N/K i.e. inverse of the fraction of the total number of files a node can store.

■

We can also recalculate the optimal query-processing load. Minimizing the query-processing load $Q = (\sum_{i=1}^{N} \frac{\lambda_i M}{\lambda n_i})$ with the total storage constraint $\sum_{i=1}^{N} c_i n_i = KM$, gives the result

$$n_i = \frac{KM\sqrt{\lambda_i/c_i}}{\sum_{i=1}^{N}\sqrt{\lambda_i/c_i}} \qquad (36)$$

which gives us:

$$Q^{\text{opt-unequal-file-size}} = \sum_{i=1}^{N} \frac{M\lambda_i}{\lambda} \frac{\sum_{i=1}^{N}\sqrt{\lambda_i/c_i}}{KM\sqrt{\lambda_i/c_i}} = \frac{(\sum_{i=1}^{N}\sqrt{\lambda_i c_i})(\sum_{i=1}^{N}\sqrt{\lambda_i/c_i})}{\lambda K} \qquad (37)$$

Using the relation between Arithmetic and Harmonic Means [14]:

$$\left(\frac{p_1 a_1 + \ldots + p_n a_n}{p_1 + \ldots + p_n}\right)^{p_1+\ldots+p_n} \geq \left[\left(\frac{p_1 (a_1)^{-1} + \ldots + p_n (a_n)^{-1}}{p_1 + \ldots + p_n}\right)^{p_1+\ldots+p_n}\right]^{-1}$$

where $\{p_i\}$ is the set of weights to positive real numbers $a_1, a_2, \ldots, a_n$. Setting $a_i = \sqrt{c_i}$ and $p_i = \sqrt{\lambda_i}/(\sum\sqrt{\lambda_i})$, we obtain:

$$(\sum_{i=1}^{N}\sqrt{\lambda_i c_i})(\sum_{i=1}^{N}\sqrt{\frac{\lambda_i}{c_i}}) \geq \left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2 \qquad (38)$$

Using Eqs (16), (37) and (38), we thus have

$$Q^{\text{opt-unequal-file-size}} \geq Q^{\text{opt}} \qquad (39)$$

Thus, we find that unequal file size increases the optimal query-processing load. Since the query-processing load with the replica distribution that minimizes the search time (Eq. 35) did not change, this also implies that when the file sizes are unequal, the disadvantage in query-processing load when replica distribution is chosen to minimize the search time is even smaller than that we observe in Section 4.3 (see Figure 10).

Recall that, for random walk search, the optimal search distance $\tau_R^{\text{opt-unequal-file-size}}$ will have the same expression as $Q^{\text{opt-unequal-file-size}}$. Thus, Eq. 39 also implies that the minimum average search time for random walk (Eq. 37) increases when file sizes are unequal. In contrast, the minimum average search time for flooding search when file sizes are unequal (Eq. 34) could be larger or smaller (depending on the distribution of $\{c_i\}$ and $\{\lambda_i\}$) than when file sizes are equal. However, since the maximum value of the minimum average search time remains unchanged for flooding search, the advantage in search time of flooding search over random walk search will still hold even as flooding incurs comparable query-processing load for typical file popularity distributions.

As discussed in Section 4.2.1 (and shown in [35]) the linear proportionality replica distribution minimizes the download time and ensures fairness if the file sizes were equal. We now show that the result remains valid in the unequal file size case.

Let b_i be the size of file i in bytes (proportional to c_i, the normalized file size). The storage size contributed by each node is the same and equal to, say, B bytes (proportional to K, the per-node storage size in number of files). For download time optimization, each node stores locally as many files as it can (we will ignore the bin-packing problem). Let node j have files $R_j = \{r_1, r_2, r_3, \ldots, r_k\}$ where $r_i \in \{1, 2, 3, \ldots, N\}$ $\forall i$ in its storage where:

$$\sum_{i \in R_j} c_i = K \qquad (40)$$

Since the $M\lambda_i$ requests (per unit time) for file i are still uniformly distributed over the n_i replicas of file i, each replica of file i still serves $M\lambda_i/n_i$ requests per unit time which amounts to $M\lambda_i c_i/n_i$ bytes per unit time. Thus, Λ_j, the bytes served per unit time by node j is:

$$\Lambda_j = \sum_{i \in R_j} \frac{M \lambda_i c_i}{n_i}.$$

Recall from Section 4.2.1 that when $n_i = \alpha\lambda_i$ where α is the proportionality constant, the download load at each node was equal in the equal file size. We can show that this replica distribution gives uniform node utilization with unequal file sizes also:

$$\Lambda_j = \sum_{i \in R_j} \frac{M \lambda_i c_i}{n_i} = \sum_{i \in R_j} \frac{M \lambda_i c_i}{\alpha\lambda_i} = \frac{M}{\alpha} \sum_{i \in R_j} c_i = \frac{MK}{\alpha}$$

(the last step in the above equation follows from Eq. 40). The proportionality constant $\alpha = KM / (\sum_{i=1}^{N} \lambda_i c_i)$ and, hence, $\Lambda_j = \sum_{i=1}^{N} \lambda_i c_i$ which is independent of j and is equal to the download request rate generated by each node on average, in bytes per unit time.

Thus, given that each node has the same download service capacity, we get the result that each node has the same utilization. Hence, the results of [35] on fairness and download time optimality remain valid in the unequal file size case also. We summarize this result in the following theorem:

Theorem 4:
In a peer-to-peer system where the download requests for a file are uniformly distributed over all the replicas of the file in the system, if the number of replicas of file i, n_i, is proportional to the average request rate for file i, λ_i, for all files i.e. $n_i = \alpha\lambda_i$ $\forall i$ where α is the proportionality constant, and each node has the same storage capacity and the same upload link bandwidth,

1. *Each node operates at the same utilization factor independent of the files it has in its storage, and*
2. *If the queueing delay is convex in the load on the serving node, then the average download time is minimized.* ■

6.3.4 Heterogeneity in Peers

While it may not be very unreasonable to assume homogenous peers in smaller communities like, say, a peer-to-peer network of schools using a shared library, nodes on the Internet exhibit a high degree of heterogeneity in the connection bandwidth and the amount of storage shared [27]. In such a scenario, treating all peers as equal results in a performance equivalent to our model where each node has the capacity equal to the lowest capacity node of the original system. One could, however, try to use the higher capacity nodes to improve performance. An often suggested approach [6] is to assign a higher number of links to higher capacity nodes which pushes the topology closer to power-law random graphs. Among the deployed applications, both Kazaa and Gnutella2 use this idea but classify nodes into just two categories, high-capacity peers ("super-peers") and ordinary peers, and assign more links to the higher capacity nodes. Our discussion of these topologies in Section 6.3.2, suggests that these topologies result in better performance. The extent of heterogeneity in storage capacities in not as well established but one can view the higher storage capacity nodes as being almost like a higher degree node: reaching a higher storage capacity node leads to a larger than usual jump in the probability of finding a file just like reaching a higher degree node increases the probability of finding the file (albeit in one or two additional hops).

Finally, we also note that if the storage capacity is proportional to the connection bandwidth of the peers, the results in [35] on the optimality of linear proportionality replica distribution for download time remain valid. This can be seen as follows. Let K_j and μ_j, respectively, be the storage capacity and the download service capacity of node j. When each node has a storage capacity proportional to its download service capacity,

$$K_j = \beta\mu_j \ \forall j \tag{41}$$

where β is the proportionality constant.

Let node j have files $R_j = \{r_1, r_2, r_3, \ldots, r_k\}$ where $r_i \in \{1, 2, 3, \ldots, N\}$ $\forall i$ in its storage. Eq. 40 in Section 6.3.3 is now rewritten as:

$$\sum_{i \in R_j} c_i = K_j$$

Thus, the bytes served per unit time by node j, with $n_i = \alpha\lambda_i$, is:

$$\Lambda_j = \sum_{i \in R_j} \frac{M \lambda_i c_i}{n_i} = \sum_{i \in R_j} \frac{M \lambda_i c_i}{\alpha\lambda_i} = \frac{M}{\alpha} \sum_{i \in R_j} c_i = \frac{MK_j}{\alpha}$$

where α is the proportionality constant. Therefore, the utilization factor ρ_j is:

$$\rho_j = \frac{\Lambda_j}{\mu_j} = \frac{MK_j}{\alpha\mu_j} = \frac{M\beta}{\alpha}$$

Thus, when each node has a storage capacity proportional to its download service capacity, linear proportionality of replica distribution ensures uniform utilization rate and, hence, the download time is minimized at the linear proportionality replica distribution.

6.3.5 Free-riding

Measurement studies report that a significant fraction of nodes in actual peer-to-peer systems do not share any files [27] while we assumed in our model that each node shares equal amount of storage. However, this does not affect our analysis as long as these "free-riders" are uniformly distributed. The existence of these free-riders decreases the overall available storage (if a fraction η of the nodes are free-riders, we now have ηKM total storage space instead of KM total storage space) but this storage should still be allocated to files in proportion to their request rates in the optimal replica distribution.

7. CONCLUSIONS

This paper addresses the effect of distributing multiple replicas on search performance in unstructured peer-to-peer systems. We provide a formal relation between the hop distance to the nearest replica of a file and the number of replicas of that file for search networks modeled by random graph topologies. This logarithmic relation between the hop distance to the nearest replica and the number of replicas is also shown to hold for other popular search network models like the power-law random graphs and the super-peer networks. Using this relation which defines the search distance in a controlled flooding search, we show that the average search distance is minimized when the number of replicas for a file is proportional to the file request rate. We had previously found ([35]) that the linear proportionality replica distribution leads to uniform node utilization and optimal download performance in peer-to-peer network models that were similar to the model we use in this paper. In the current paper, we also showed that with this replica distribution, greedy behavior offers no advantages. While the linear proportionality replica distribution optimizes the search time (in controlled flooding search) and the download time, it is not optimal for the query-processing load. Since, query-processing traffic and download traffic use the same links, a high query-processing load can adversely affect the download performance. Therefore, we derive expressions for the average search distance and the query-processing load per query for controlled flooding search under the linear proportionality replica distribution and compare these to the analogous expressions for the square-root proportionality replica distribution that is optimal for query-processing load. We find that, for realistic file request rate distributions (where file popularity distribution is not very skewed e.g., for zipf-distributed file request rates, the zipf-exponent is less than or around 1), the additional query-processing load incurred in using the linear proportionality replica distribution instead of the square-root proportionality replica distribution is small.

Random walk search has been proposed as an alternative to flooding. We derive the average search time and query-processing load per query expressions for random walk search under the square-root proportionality replica distribution known to be optimal for random walks and show that controlled flooding has logarithmically better search time than random walk while the optimized random walk and the optimized controlled flooding generate similar query-processing loads when the file popularity distribution is not very skewed (e.g., for zipf-distributed file request rates, the zipf-exponent is less than or around 1). We also find, for both search methods under their respective optimal replica distributions, that the query-processing load is upper-bounded by the inverse of the fraction of the total number of files a node can store, and is independent of the number of nodes in the network.

We consider an example peer-to-peer system and show that simple distributed methods such as LRU can result in near linear proportionality replica distributions. Finally, in addressing our assumptions, we find that if the file popularities and the underlying search network were not uniform across the network but rather fully clustered, the query-processing load will be substantially reduced. In our discussion of unequal file sizes and peer capacities, we find that the results in [35] can be extended to show that linearly proportional file replica distribution maintains its advantages for download performance even when the file sizes and peer capacities are not equal.

8. ACKNOWLEDGMENTS

We thank Boon-Thau Loo for providing us the Gnutella2 traces used to extract the search network topology of Figure 6.

9. REFERENCES

[1] Adamic, L. A., Lukose, R. M., Puniyani, A. R., and Huberman, B. A. Search In Power-Law Networks. *Physical Review E 64* (2001).

[2] Albert, R. and Barabasi, A. L. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, vol. 74, pp. 47--97, 2002.

[3] Baryshnikov, Y., Coffman, E., Jelenkovic, P., Momcilovic, P., Rubenstein, D. Flood Search Under the California Split Rule. *Operations Research Letters*, Volume 32, Number 3, May 2004.

[4] Bollobas. B, *Random Graphs,* Academic Press, London, 1985.

[5] Breslau, L., Cao, P., Phillips, G., and Shenker, S. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM 1999*.

[6] Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N. and Shenker, S. Making Gnutella-like P2P Systems Scalable. In *Proceedings of ACM SIGCOMM 2003*.

[7] Cohen, B. Incentives Build Robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, 2003.

[8] Cohen, E. and Shenker, S. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of ACM SIGCOMM 2002*.

[9] Ge, Z., Figueiredo, D. R., Jaiswal, S., Kurose, J., and Towsley, D. Modeling peer-peer file sharing systems. In *Proceedings of IEEE Infocom 2003*.

[10] Gkantsidis, C., Ammar, M., Zegura, E. On the Effect of Large-Scale Deployment of Parallel Downloading. In *IEEE Workshop on Internet Applications (WIAPP'03),* 2003.

[11] Gkantsidis, C., Mihail, M., Saberi, A. Random Walks in Peer-to-Peer Networks. In *Proceedings of ACM INFOCOM 2004*.

[12] Gnutella2 Specification, 2003, http://www.gnutella2.com.

[13] Gummadi, K. P., Dunn, R. J., Saroiu, S., Gribble, S. D., Levy, H. M., Zahorjan, J. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of ACM SOSP, 2003*.

[14] Hardy, G. H., Littlewood, J. E., Pólya, G. *Inequalities*, Cambridge University Press, London, 1952.

[15] Kemp, R. Binary search trees constructed from non-distinct keys with/without specified probabilities. Theoretical Computer Science 156 (1996) 39-70.

[16] Knuth, D. E. *The Art of Computer Programming, Vol 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 2nd Ed., 1975.

[17] Koo, S. G. M., Rosenberg, C., Xu, D. Analysis of Parallel Downloading for Large File Distribution. In *Proceedings of the 9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*, San Juan, Puerto Rico, May 2003.

[18] Le Fessant, F., Handurukande, S., Kermarrec, A. M. and Massouli, L. Clustering in Peer-to-Peer File Sharing Workloads. In *IPTPS 2004*.

[19] Leibowitz, N., Ripeanu M., and Wierzbicki, A. Deconstructing the Kazaa Network. In *3rd IEEE Workshop on Internet Applications (WIAPP'03)*, June 2003, San Jose, CA.

[20] Liben-Nowell, D., Balakrishnan, H., Karger, D. Analysis of the Evolution of Peer-to-Peer Systems. In *PODC 2002*.

[21] Loo, B. T., Huebsch, R., Ion Stoica, I. and Hellerstein, J. M.,
The Case for a Hybrid P2P Search Infrastructure. In *IPTPS 2004*.

[22] Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of 16th ACM International Conference on Supercomputing (ICS'02),* New York, NY, June *2002*.

[23] Qiu, D., Srikant, R. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proceedings of ACM SIGCOMM 2004*.

[24] Pouwelse, J. A., Garbacki, P., Epema, D. H. J. and Sips, H. J. A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System. Technical report PDS-2004-003, TU Delft, April 2004 http://www.pds.ewi.tudelft.nl/reports/2004/PDS-2004-003/

[25] Ripeanu M., Iamnitchi, A., Foster, I. Mapping the Gnutella Network. *IEEE Internet Computing,* Jan-Feb 2002.

[26] Rowstron A. I. T., and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001, pp. 329–350.

[27] Saroiu, S., Gummadi, K. P., Gribble, S. D. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*.

[28] Sarshar, N., Oscar Boykin, P., Roychowdhury, V. P. "Percolation Search in Power Law Networks: Making Unstructured Peer-To-Peer Networks Scalable," In *3rd IEEE International Conference on Peer-to-Peer Computing, 2003*.

[29] Sharman Networks Ltd. KaZaA Media Desktop, 2001. http://www.kazaa.com/.

[30] Sen S. and Wang, J. Analyzing Peer-to-Peer Traffic Across Large Networks," In *Internet Measurement Workshop (IMW 2002)*, Marseille, France, 2002.

[31] Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*.

[32] Tsoumakos, D., Roussopoulos, N. Adaptive Probabilistic Search for Peer-to-Peer Networks. In *3rd IEEE International Conference on Peer-to-Peer Computing, 2003*.

[33] Yang, B. and Garcia-Molina, H. Designing a Super-Peer Network. In *Proceedings of ICDE*, 2003.

[34] Yang, X. and de Veciana, G. Service Capacity of Peer to Peer Networks. In *Proceedings of ACM INFOCOM 2004*.

[35] Tewari, S., Kleinrock, L. "On Fairness, Optimal Download Performance and Proportional Replication in Peer-to-Peer Networks," to appear in *Proceedings of IFIP Networking*, May 2005.

[36] A. Aho, P. Denning, D. Ullman. Principles of optimal page replacement. *Journal of the ACM 18* (1971), pp. 80-93.

[37] J. T. Robinson, M. V. Devarakonda. Data cache management using frequency-based replacement. In *Proc. of ACM SIGCOMM,* 1990.