Replication Strategies in Unstructured Peer-to-Peer Networks

Paper ID:298
12 pages

Abstract—

The Peer-to-Peer (P2P) architectures that dominate on today's Internet are decentralized and unstructured. Search is blind in that it is independent of the query and is thus not more effective than probing randomly chosen peers. One technique to improve the effectiveness of blind search is to proactively replicate data.

We evaluate and compare different replication strategies and reveal interesting structure: Two very common but very different replication strategies – uniform and proportional – yield the same average performance on successful queries, and are in fact worse than any replication strategy which lies between them. The optimal strategy lies between the two and can be achieved by simple distributed algorithms.

These fundamental results offer a new understanding of replication and show that currently deployed replication strategies are far from optimal and that optimal replication is attainable by protocols that resemble existing ones in simplicity and operation.

I. Introduction

Peer-to-peer (P2P) systems, almost unheard of two years ago, are now one of the most popular Internet applications and a very significant source of Internet traffic. While Napster's recent legal troubles may lead to its demise, there are many other P2P systems which are continuing their meteoric growth and which may soon surpass Napster's peak popularity. Despite their growing importance, the performance of P2P systems is not yet well understood.

P2P systems were classified by [7] into three different categories. Some P2P systems, such as Napster [5], are *centralized* in that they have a central directory server to which users can submit queries (or searches). Other P2P systems are *decentralized* and have no central server; the hosts form an ad hoc network among themselves and send their queries to their peers. Of these decentralized designs, some are *structured* in that they have close coupling between the P2P network topology and the location of data; see [10], [8], [9], [12], [1] for a sampling of these designs. Other decentralized P2P systems, such as Gnutella [2] and FastTrack [11]- based Morpheus [4] and KaZaA [3], are *unstructured* with no coupling between topology and data location. Each of these design styles – centralized, decentralized structured, and decentralized unstructured – have their advantages and disadvantages and it is not our intent to advocate for a particular choice among them. However, the decentralized unstructured systems are the most commonly used in today's Internet. They also raise an important performance issue – how to replicate data in such systems – and that performance issue is the subject of this paper.

In these decentralized unstructured P2P systems, the hosts form a P2P overlay network; each host has a set of "neighbors" that are chosen when it joins the network. A host sends its query (*e.g.*, searching for a particular file) to other hosts in the network; Gnutella uses a flooding algorithm to propagate the query, but many other approaches for are possible. The fundamental point is that in these unstructured systems, because the P2P network topology is unrelated to the location of data, the set of nodes receiving a particular query is unrelated to the content of the query. A host doesn't have any information about which other hosts may best be able to resolve the query. Thus, search probes can not be on average more effective than probing random nodes. Indeed, simulations in [7] suggest that random probing is a reasonable model for search performance in these decentralized unstructured P2P systems.

To improve system performance, one wants to minimize the number of hosts that have to be probed before the query is resolved. One way to do this is to replicate the data on several hosts.[1] That is, either when the data is originally stored or when it is the subject of a later search, the data can be proactively replicated on other hosts. Gnutella does not support proactive replication, but at least part of the current success of FastTrack-based P2P networks can be attributed to replication: FastTrack, designates high-bandwidth nodes as search-hubs (super-nodes). Each supernode replicates the index of several other peers. As a result, each FastTrack search probe emulates several Gnutella probes and thus is much more effective.

It is clear that blind search is more effective when larger index can be viewed per probe, but even though FastTrack increased per-probe capacity, it basically uses the same *replication strategy* as Gnutella: The relative index capacity dedicated to each item is proportional to the number of peers that have copies (and essentially, to the query rate to that item).

The fundamental question we address in our paper is: given fixed constraints on per-probe capacity, what is the optimal way to replicate data ?

We define replication strategies that, given a query frequency distribution, specify for each item the number of copies made. The main metric we consider is the performance on successful queries, which we measure by the resulting average search size. We also consider performance on insoluble queries, which is captured by the maximum search size allowed by the system.

Our analysis reveals several surprising fundamental results. We first consider two natural but very different replication strategies: *uniform* and *proportional*. The uniform strategy, replicating everything equally, appears naive, whereas the proportional strategy, where more popular items are more replicated, is designed to perform better. However, we show that the two replication strategies have the same average search size on successful queries. Furthermore, we show that the uniform and propor-

[1]For the basic questions we address, it does not matter if the actual data is replicated or if only pointers to the data are replicated. The actual use depends on the architecture and is orthogonal to the scope of this paper. Thus, in the sequel, "copies" refers to actual copies or pointers.

tional strategies constitute two extreme points of a large family of strategies which lie "between" the two and that any other strategy in this family has better average search size. We then show that one of the strategies in this family, *Square-root* replication, minimizes the average search size.

Proportional and Uniform replications, however, are not equivalent. Proportional makes popular items easier to find and less popular items harder to find. In particular, Proportional replication requires a much higher limit on the maximum search size while Uniform minimizes this limit, and thus, minimizes resource consumption due to insoluble queries. We show that the maximum search size with Square-root strategy is in-between the two, and closer to Uniform, and continue to define a range of optimal strategies that balance resources consumed on soluble and insoluble queries. Interestingly, these optimal strategies lie "in-between" Uniform and Square-root and are "far" from Proportional.

Last, we address how to implement these replication strategies in the distributed setting of an unstructured P2P network. It is easy to implement the uniform and proportional replication strategies; for uniform the system creates a fixed number of copies when the item first enters the system, for proportional the system creates a fixed number of copies every time the item is queried. What isn't clear is how to implement the square-root replication policy in a distributed fashion. We present a distributed algorithm, called path replication, which produces the optimal replication. Surprisingly, this replication policy is implemented, in a somewhat different form, in the Freenet P2P system.

These results are particularly intriguing given that Proportional replication is close to what is used by current P2P networks: With Gnutella there is no proactive replication and thus the number of locations an item is at is the number of nodes that requested it. Even FastTrack, that uses replication, replicates the complete index of each node to a search hub, and thus, the relative representation of each item remains the same. Our results suggest that Proportional, although intuitively appealing, is far from optimal and a simple distributed algorithm which is consistent in spirit with FastTrack replication is able to obtain optimal replication.

In the next section we introduce our model and metrics and present a precise statement of the problem. In Section III we examine the Uniform and Proportional replication strategies. In Section IV we show that *Square-root* replication minimizes the average search size on soluble queries and provide some comparisons to Proportional and Uniform replication. Section V defines the optimal policy parameterized by varying cost of insoluble queries. In Section VI we present the distributed algorithm that yields square-root replication and present simulation results of its performance.

II. Model and statement of the problem

The network consists of n nodes, each with *capacity* ρ which is the number of copies/keys that the node can hold [2]. Let $R = n\rho$ denote the total capacity of the system. There are m data items in the system. The normalized vector of *query rates* takes the form $\mathbf{q} = q_1 \geq q_2 \geq \cdots \geq q_m$ with $\sum q_i = 1$. The query rate q_i is the fraction of all queries that are issued for the ith item.

[2]Later on in this section we explain how to extend the definitions, results, and metrics to heterogeneous capacities

An *allocation* is a mapping of items to the number of copies of that item (where we assume there is no more than one copy per node). We let r_i denote the number of copies of the i'th item (r_i counts all copies, including the original one), and let $p_i \equiv r_i/R$ be the fraction of the total system capacity alloted to item i: $\sum_{i=1}^{m} r_i = R$. The allocation is represented by the vector $\mathbf{p} = (r_1/R, r_2/R, \ldots, r_m/R)$. A replication or allocation *strategy* is a mapping between the query rate distribution $\mathbf{q}$ and the allocation $\mathbf{p}$.

We assume $R \geq m \geq \rho$ because outside of this region the problem is either trivial (if $m \leq \rho$ the optimal allocation is to have copies of all items on all nodes) or insoluble (if $m > R$ there is no allocation with all items having at least one copy).

Our analysis is geared for large values of n (and $R = \rho n$) with our results generally stated in terms of $\mathbf{p}$ and ρ with n factored out. Thus, we do not concern ourselves with integrality restrictions on the r_i's. However, we do care about the bounds on the quantities p_i. Since $r_i \geq 1$, we have $p_i \geq \ell$ where $\ell = \frac{1}{R}$. Since there is no reason to have more than one copy of an item on a single node, we have $r_i \leq n$ and so $p_i \leq u$ where $u = \frac{n}{R} = \rho^{-1}$. Later in this section we will discuss reasons why these bounds may be made more strict.

We argued in the introduction that performance of blind search is captured well by random probes. This abstraction allows us to evaluate performance without having to consider the specifics of the overlay structure. Simulations in [7] show that random probes are a reasonable model for several conceivable designs including Gnutella-like overlays. Specifically, the search mechanism we consider is *random search*: The search repeatedly draws a node uniformly at random and asks for a copy of the item; the search is stopped when the item is found. The *search size* is the number of nodes drawn until an answer is found. With the random search mechanism, search sizes are random variables drawn from a Geometric distribution with expectation $1/p_i$. Thus, performance is determined by how many nodes have copies of any particular item. For a query distribution $\mathbf{q}$ and an allocation $\mathbf{p}$, we define the *average search size* (ASS) $A_{\mathbf{q}}(\mathbf{p})$ to be the expected number of nodes one needs to visit until an answer to the query is found, averaged over all items. It is not hard to see that

$$A_{\mathbf{q}}(\mathbf{p}) = 1/\rho(\sum_i q_i/p_i)\ . \qquad (1)$$

The set of *legal allocations* P is a polyhedron defined by the $(m-1)$-dimensional simplex intersected with an m-dimensional hypercube:

$$\begin{aligned} \sum_{i=1}^{m} p_i &= 1 & (2)\\ \ell \leq\ p_i &\leq u\ . & (3) \end{aligned}$$

The legal allocation that minimizes the average search size is the solution to the optimization problem

$$\textbf{Minimize} \sum_{i=1}^{m} q_i/p_i \text{ such that } \mathbf{p} \in P\ .$$

One obvious property of the optimal solution is *monotonicity*

$$u \geq p_1 \geq p_2 \geq \cdots \geq p_m \geq \ell \quad (4)$$

(If $\mathbf{p}$ is not monotone then consider two items i, j with $q_i > q_j$ and $p_i < p_j$. The allocation with p_i and p_j swapped is legal, if $\mathbf{p}$ was legal, and has a lower ASS.)

In the next section we explore the performance of two common replication strategies, uniform and proportional. However, we first discuss refinements of our basic model.

A. Bounded search size and cost of insoluble queries

So far we have assumed that searches continue until the item is found, and our subsequent analysis of the ASS metric will be based on that assumption, but we now discuss applying these results when searches are truncated when some maximal search size L is reached. We say an item is *locatable* if a search for it is successful with very high probability. When comparing performance of different allocations, we assume the same set of locatable items. We can apply our results to truncated searches if we restrict ourselves to allocations where the lower bound ℓ is sufficiently large, so that all items are locatable and expression (1) is a good approximation for the average truncated search size. The lower bound ℓ is thus determined according to L. A search for item i is likely to be successful only if $p_i \geq 1/(\rho L)$. The probability of failure is 2^{-C} when $p_i \geq C/(\rho L)$. Setting $C = O(\log m)$ would guarantee failure probability which is polynomially small in m.

For truncated random searches the search sizes are random variables drawn from a Geometric distribution with expectation $1/p_i$ but truncated by L. The expression (1) becomes an upper bound on A, but the error is at most $\min_i \sum_{j\geq 1}(1-\rho p_i)^{L+j} = \min_i (1-\rho p_i)^L/(\rho p_i) \leq \exp(-C)/(\rho\ell) = (L/C)\exp(-C)$. In the sequel, we assume that ℓ is such that $\ell > \ln L/(\rho L)$ and use the untruncated approximation (1) for the average search size (it is within an additive term of $\exp(-\ell\rho L/\ln L) < 1$). The likelihood for an unsuccessful search for a locatable item is at most $\exp(-\ell\rho L) < 1/L$.

The parameter L is meaningful when considering the performance on *insoluble queries*, that is, queries made to items that are not locatable. In actual systems, some fraction of queries are insoluble, and search performed on such queries would continue until the maximum search size is exceeded. The cost of these queries is not captured by the ASS metric, but is proportional to L and to the fraction of queries that are insoluble.

When comparing replication strategies on the same set of locatable items, we need to consider both the ASS, which captures performance on soluble queries and L, which captures performance on insoluble queries. In this situation we assume that the respective $L(\mathbf{p})$ is the solution of $\min_i p_i = \ln L/(\rho L)$. Thus, if f_s is the fraction of queries which are soluble and $(1-f_s)$ is the fraction of insoluble queries, the performance of the allocation $\mathbf{p}$ is

$$f_s A_{\mathbf{q}}(\mathbf{p}) + (1-f_s)L(\mathbf{p})\ . \quad (5)$$

B. Heterogeneous capacities and bandwidth

Prior definitions assumed that all copies have the same size and that all nodes have the same storage and the same likelihood of getting probed. In reality, hosts have different capacities and bandwidth and in fact, the most recent wave of unstructured P2P networks [4], [3] exploits this asymmetry. We argue that our results still apply in these more general settings. Suppose nodes have capacities ρ_i and *visitation* weight v_i (in the Uniform case $v_i = 1$, in general v_i is the factor in which the visitation rate differs from the average). It is not hard to see that the average capacity seen per probe is $\overline{\rho} = \sum_i v_i \rho_i$. The quantity $\overline{\rho}$ simply replaces ρ in Equation 1 and in particular, all allocations are affected the same way.

An issue that arises when the replication is of copies (rather than pointers) is that items often have very different sizes. Our subsequent analysis can be extended to this case by treating the ith item as a group of c_i items with the same query rate, where c_i be the size of item i. As a result we obtain altered definition of the basic allocations:

- Proportional has $p_i = c_i q_i / \sum_j c_j q_j$ (proportion to query rate and size).
- Uniform has $p_i = c_i / \sum_j c_j$ (proportion to item's size).
- Square-root has $p_i = c_i\sqrt{q_i} / \sum_j c_j\sqrt{q_j}$ (proportional to size and to the square-root of the query rate).

III. ALLOCATION STRATEGIES

A. Uniform and Proportional

We now address the performance of two replication strategies. The uniform replication strategy is where all items are equally replicated:

Definition III.1: Uniform allocation is defined when $\ell \leq 1/m \leq u$ and has $p_i = 1/m$ for all $i = 1, \ldots, m$.

This is a very primitive replication strategy, where all items are treated identically even though some items are more popular than others. One wouldn't, initially, think that such a strategy would produce good results.

When there are restriction on the search size (and thus on ℓ), Uniform allocation has the property that it is defined for all $\mathbf{q}$ for which some legal allocation exists: Any allocation $\mathbf{p}$ other than Uniform must have i where $p_i > 1/m$ and j where $p_j < 1/m$; If Uniform results in allocations outside the interval $[\ell, u]$, then either $1/m > u$ (and thus $p_i > u$) or $1/m < \ell$ (and thus $p_j < \ell$).

An important appeal of Uniform allocation is that it minimizes the required maximum search size, and thus, minimizes system resources spent on insoluble queries. It follows from Equation 5 that when a large fraction of queries are insoluble, Uniform is (close to) optimal.

Another natural replication strategy is to have r_i be proportional to the query rate.

Definition III.2: Proportional allocation is defined when $\ell \leq q_i \leq u$. The proportional solution has $p_i = q_i$ for all i.

Some of the intuitive appeal of Proportional allocation is that it minimizes the *maximum utilization rate* [7]. This metric is relevant when the replication is of copies rather than of pointers, that is, when a successful probe is much more expensive to process than an unsuccessful probe. The *utilization rate* of a copy is the average rate of requests it serves; when there are more copies, each individual copy has a lower utilization. To avoid hot-spots it is desirable to have low values for the maximal utilization. Under random search, all copies of the same item i

have the same utilization rate q_i/p_i. The maximum utilization rate is thus $\max_i q_i/p_i$. The average utilization over all copies, $\sum_{i=1}^{m} p_i q_i/p_i = 1$, is independent of $\mathbf{p}$.

When compared to Uniform, Proportional improves the most common searches at the expense of the rare ones, which presumably would improve overall performance. We now analyze some of the properties of these two strategies.

A straightforward calculation reveals the following surprising result:

Lemma III.1: Proportional and Uniform allocations have the same average search size $A = m/\rho$, which is independent of the query distribution.

We note that Lemma III.1 easily generalizes to all allocations that are a mix of Proportional and Uniform, that is, any item i has $p_i \in \{1/m, q_i\}$. We next characterize the space of allocations.

B. Characterizing allocations

As a warmup, we characterize the space of allocations for $m = 2$. We shall see that Proportional and Uniform constitute two points on the space of allocations, with anything "in between" them achieving better performance, and anything else having worse performance. Consider a pair of items with $q_i \geq q_j$. The range of allocations is defined by a single parameter $0 < x < 1$, with $p_i/(p_i+p_j) = x$ and $p_j/(p_i+p_j) = (1-x)$. Proportional corresponds to $x = q_i/(q_i + q_j)$, Uniform to $x = 0.5$. The range $0 < x < 0.5$ contains non-monotone allocations where the less-popular item obtains a larger allocation. The range $1 > x > q_i/(q_i + q_j)$ contains allocations where the relative allocation of the more popular item is larger than its relative query rate. These different allocations are visualized in Figure 1(A), which plots p_i/p_j as a function of q_i/q_j.

The ASS for these two items is proportional to $q_i/x + q_j/(1-x)$. The function has the same value on $x = 0.5$ and $x = q_i/(q_i + q_j)$ and is concave. The minimum is obtained at some middle point. By taking the first derivative, equating it to zero, and solving the resulting quadratic equation, we obtain that the minimum is obtained at $x = \sqrt{q_i}/(\sqrt{q_i} + \sqrt{q_j})$. Figure 1(B) shows the average search size when using these allocations on two items ($m = 2$ and $\rho = 1$). In this case, the maximum gain factor by using the optimal allocation over Uniform or Proportional is 2.

In the sequel we extend some of the observations made here to $m \geq 2$. In particular, we develop a notion of an allocation being "between" Uniform and Proportional and show that these allocations have better ASS, we also define the policy that minimizes the ASS, and we bound the maximum gain as a function on m, u, and ℓ.

C. Allocations between Uniform and Proportional

For $m = 2$, we noticed that all allocations that lie "between" Uniform and Proportional have smaller ASS. For general m we first define the notion of being between Uniform and Proportional:

Definition III.3: An allocation $\mathbf{p}$ lies between Uniform and Proportional if for any pair of items $i < j$ we have $q_i/q_j \geq p_i/p_j \geq 1$, that is, the ratio of allocations p_i/p_j is between 1 ("Uniform") and the ratio of their query rates q_i/q_j ("Proportional").

Note that allocations in this family are monotone and include Uniform and Proportional.

We now establish that all allocations in this family other than Uniform and Proportional have a strictly better average search size:

Theorem III.1: Consider an allocation p between Uniform and Proportional. Then $\mathbf{p}$ has an average search size of at most m/ρ. Moreover, if $\mathbf{p}$ is different from Uniform or Proportional then its average search size is strictly less than m/ρ.

Proof: The limits $q_i/q_j \geq p_i/p_j \geq 1$ hold if and only if they hold for any consecutive pair of items. These allocations are characterized by the $m-1$-dimensional polyhedron obtained by intersecting the $m - 1$-dimensional simplex (the constraints $\sum p_i = 1$ $p_i > 0$ defining the space of all allocations) with additional constraints $p_i \geq p_{i+1}$ and $p_{i+1} \leq p_i q_{i+1}/q_i$. Recall that the average search size function $F(p_1, \ldots, p_m) = \sum_{i=1}^{m} q_i/p_i$ is convex. Thus, the maximum value(s) must be obtained on vertices of this polyhedron. Vertices are allocation where for any $1 \leq i < m$, either $p_i = p_j$ or $p_i = p_j q_i/q_j$.

It remains to show that the maximum of the average search size function over all vertex allocation is obtained on the Uniform or Proportional allocations.

We show that if we are at a vertex other than Proportional or Uniform, we can get to one of these allocations by a series of moves, where each move is to a vertex with a larger ASS than the current one. Consider a "vertex" allocation $\mathbf{p}$ different than Proportional and Uniform. Let $1 < k < m$ be the minimum such that the ratios $p_2/p_1, \ldots, p_{k+1}/p_k$ are not all 1 or not all equal to the respective ratio q_{i+1}/q_i. Note that we must have that $q_{i+1} < q_i$ for at least one $i = 1, \ldots, k-1$; and $q_{k+1} < q_k$. (otherwise, if $q_1 = \cdots = q_k$ then any vertex allocation is such that $p_1, \ldots, p_{k+1}$ are consistent with Uniform or with Proportional; if $q_k = q_{k-1}$ then minimality of k is contradicted).

There are two possibilities for the form of $\mathbf{p}$:

1. We have $p_{i+1}/p_i = 1$ for $i = 1, \ldots, k-1$ and $p_{k+1}/p_k = q_k/q_{k+1}$.
2. We have $p_{i+1}/p_i = q_{i+1}/q_i$ for $i = 1, \ldots, k-1$ and $p_{k+1}/p_k = 1$.

Claim: at least one of the two "candidate" vertices characterized by

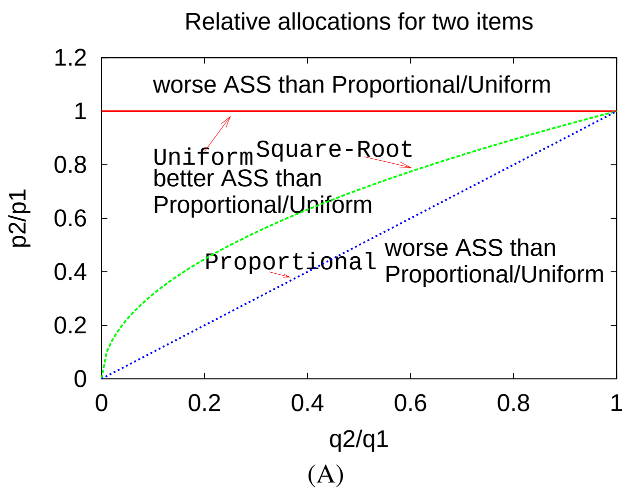
- $\mathbf{p}'$: $p'_{i+1}/p'_i = 1$ for $i = 1, \ldots, k$ and $p'_{i+1}/p'_i = p_{i+1}/p_i$ for $(i > k)$, or
- $\mathbf{p}''$: $p''_{i+1}/p''_i = q_{i+1}/q_i$ for $i = 1, \ldots, k$ and $p''_{i+1}/p''_i = p_{i+1}/p_i$ for $(i > k)$.

has strictly worse ASS than $\mathbf{p}$.

Note that this claim concludes the proof: We can iteratively apply this move, each time selecting a "worse" candidate. After each move, the prefix of the allocation which is consistent with either Uniform or Proportional is longer. Thus, after at most $m - 1$ such moves, the algorithm reaches the Uniform or the Proportional allocation.

The proof of the claim is fairly technical and is deferred to the Appendix.

Simple modification in the proof show that allocations such that $\forall j, p_j < p_{j+1}$ (less popular item gets larger allocation) or



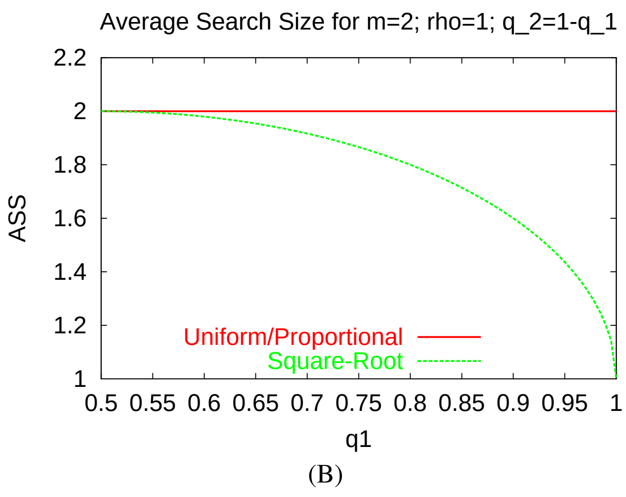


Fig. 1. (A) The space of allocations on two items (B) The average query cost for two items

$\forall j, p_j/p_{j+1} > q_j/q_{j+1}$ (between any two items, the more popular item get more than its share according to query rates) perform strictly worse than Uniform and Proportional. ■

Lemma III.2: All allocations in this family are such that $p_1 \leq q_1$ and $p_m \geq q_m$, thus, they constitute legal allocations whenever the query distribution is such that Proportional is a legal allocation.

Proof: Consider such an allocation $\mathbf{p}$. We have that $p_i \geq (q_i/q_1)p_1$. Thus $1 \geq \sum_{i=1}^m p_i \geq (\sum_{i=1}^m q_i)p_1/q_1 = p_1/q_1$. Hence, $p_1 \leq q_1$. Similarly $p_i \leq (q_i/q_m)p_m$, and we obtain that $p_m \geq q_m$. ■

IV. The Square-Root allocation

We consider an allocation where for any two items, the ratio of allocations is the square root of the ratio of query rates. Note that this allocation lies "between" Uniform and Proportional in the sense of Definition III.3. We show that this allocation minimizes the ASS.

Definition IV.1: Square-root allocation is defined when $\ell \leq \sqrt{q_i}/\sum_i \sqrt{q_i} \leq u$. and has $p_i = \sqrt{q_i}/\sum_i \sqrt{q_i}$ for all i.

Lemma IV.1: Square-root allocation, when defined, minimizes the average search size.

Proof: The goal is to minimize $\sum_{i=1}^m q_i/p_i$. This optimization problem is likely to have arisen in different context. One variant that had been considered is the capacity assignment problem [6]. We include a proof for the sake of completeness.

Substituting $p_m = 1 - \sum_{i=1}^{m-1} p_i$ we have

$$F(p_1, \ldots, p_{m-1}) = \sum_{i=1}^{m-1} q_i/p_i + q_m/(1 - \sum_{i=1}^{m-1} p_i) .$$

We are looking for the minimum of F when $\sum_{i=1}^{m-1} p_i < 1$ and $p_i > 0$. The value of F approaches ∞ when we get closer to the boundary of the simplex. Thus, the minimum must be obtained at an interior point. By solving $dF/dp_i = 0$ we obtain that

$$p_i = (1 - \sum_{j=1}^{m-1} p_j)\sqrt{q_i/q_m} = p_m\sqrt{q_i/q_m} .$$

■

Recall that for all $\mathbf{q}$, the average search size under both Uniform and Proportional allocations is m/ρ. The average search size under Square-Root allocation is

$$(\sum q_i^{1/2})^2/\rho ,$$

and depends on the query distribution. An interesting question is the potential gain of applying Square-Root rather than Uniform or Proportional allocations. The gain can be bounded by m, u, and ℓ (proof is in the Appendix):

Lemma IV.2: Let A_{min} be the average search size using Square-root allocation. Let A_{uniform} be the average search size using Proportional or Uniform allocation. Then

$$A_{\text{uniform}}/A_{\text{min}} \leq m(u + \ell - m\ell u) .$$

Moreover, this is tight for some distributions.

Note that if $\ell = 1/m$ or $u = 1/m$ then the only legal allocation is $1/m$ on all items, and indeed the gain ratio is 1. If $\ell \ll 1/m$, the gain ratio is roughly mu.

A. Comparing strategies on specific query distributions

We illustrate the performance of different strategies using query distribution obtained from Web proxy logs. We looked at the top m Urls, with "query rates" proportional to the number of requests, and top m hostnames, with "query rates" proportional to the number of users that requested the hostname. The value of m corresponds to the number of locatable items. Figure 2 shows the ratio of the ASS of Square-root allocation and Proportional allocation for different values of m (the x axis). For larger values of m, the ASS of Square-root allocation is 30%-50% of that of Proportional or Uniform allocations. Figure 3 shows the maximum search size, which determines the cost of insoluble queries. The figure shows that the maximum search size under Square-root is within a factor of 2 of the smallest possible (Uniform) whereas Proportional requires a much larger search size.

The optimal policy which minimizes resource consumption for a given ratio of soluble and insoluble queries is discussed in the following Section.

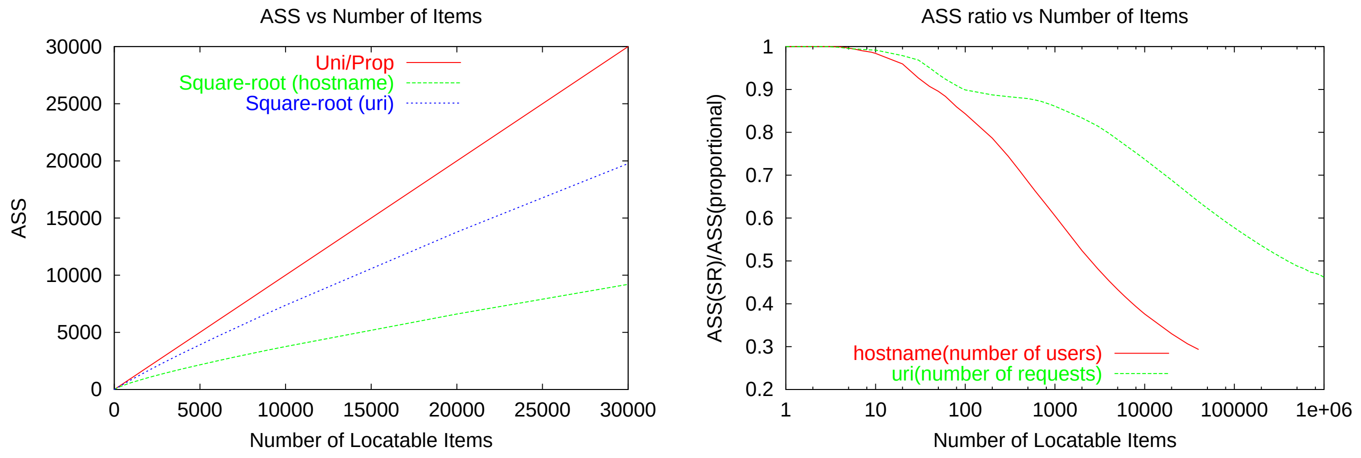


Fig. 2. The ASS of different strategies, and the ratio of ASS under optimal to ASS under proportional, as a function of the number of locatable items.

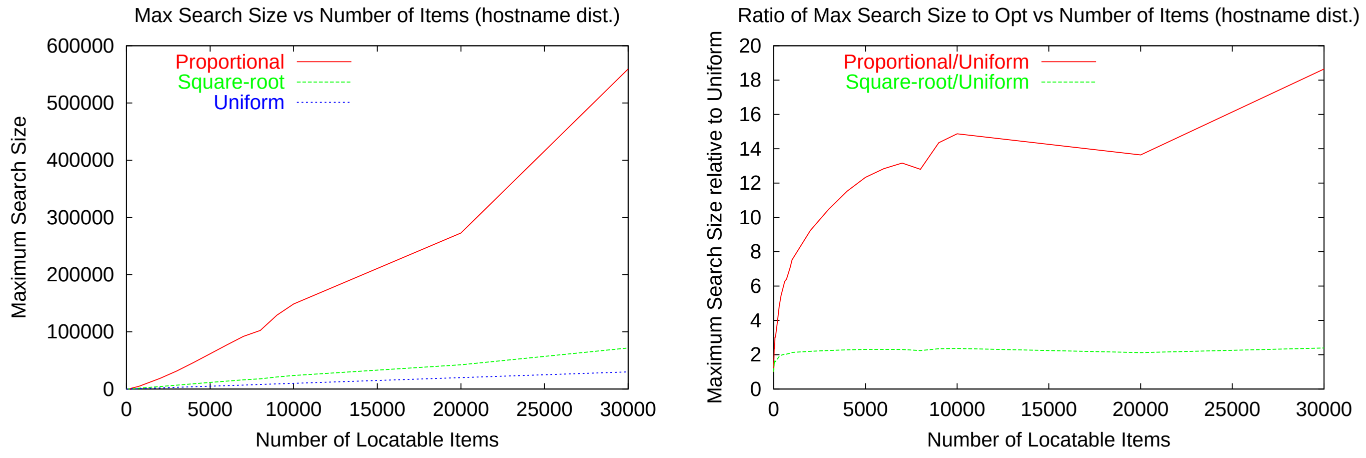


Fig. 3. Maximum search size as a function of number of locatable items, and ratio of maximum search size to the minimum possible (Uniform allocation).

V. Square-root* and Proportional* allocations

Suppose now that we fix the set of locatable items and the bound on the maximum search size (that is, we fix the lower bound ℓ on the allocation of any one item). Phrased differently, this is like fixing the resource use due to insoluble queries.

With ℓ (or u) fixed, Square-root and Proportional allocations may not be defined - since as we had seen, they are defined on a more restricted set of query distributions than Uniform.

We now ask what are the replication strategies which minimize ASS under these constraints ?

We define a natural extension of Square-Root, Square-Root*, which always results in a legal allocation (when there is a legal allocation). We show that Square-Root* minimizes ASS. Square-root* allocation lies in-between Square-root and Uniform. Since it minimizes the ASS while fixing the maximum search size, by sweeping ℓ, we obtain a range of optimal strategies for any given ratio of soluble and insoluble queries. The extremes of these range are Uniform allocation, which is optimal if there is a large number of insoluble queries and Square-root allocation which is optimal is there are relatively few insoluble queries.

A. Square-Root* allocation

Lemma V.1: Consider a query distribution $\mathbf{q}$ where $q_1 \geq \cdots \geq q_m$ and $\ell \leq 1/m \leq u$. There is a unique monotone allocation $\mathbf{p} \in P$ for which the following conditions apply. Furthermore, this allocation minimizes the ASS.
1. if $u > p_i, p_j > \ell$ then $p_i/p_j = \sqrt{(q_i/q_j)}$.
2. if $p_j = \ell \leq p_i$ or if $p_j \leq u = p_i$, then $p_i/p_j \leq \sqrt{(q_i/q_j)}$.
The proof is deferred to the Appendix, and provides a simple iterative procedure to compute the Square-Root* allocation for arbitrary q_i's.

The Square-Root* allocation as a function of ℓ varies between $\ell = \sqrt{q_m}/\sum_i \sqrt{q_i}$ (where Square-Root* coincides with the Square-Root allocation) and $1/m$ (where Square-Root* coincides with the Uniform allocation). On intermediate values of ℓ, a suffix of the items is assigned the minimum allcation value ℓ, but the remaining items still have allocations proportional to the square-root of their query rate.

The ASS as a function of ℓ is a piecewise hyperbolic function that is minimized for $\ell = \sqrt{q_m}/\sum_i \sqrt{q_i}$ and is increasing with ℓ. The breakpoints correspond to a suffix of the items that have minimum allocation. Basic algebraic manipulations show that the breakpoints are $\ell_n = p_n/(1 - \sum_{i=n}^{m} p_i + p_n(m-n+1))$, where $p_i = \sqrt{q_i}/\sum_j \sqrt{q_j}$ is the allocation of the ith item under

Square-Root allocation. Note that the extremes of this range are indeed $\ell_m = p_m$ (Square-Root allocation) and $\ell_1 = 1/m$ (Uniform allocation).

The ASS for $\ell \in [\ell_n, \ell_{n-1})$ is

$$\sum_{i=n}^{m} q_i/\ell + (\sum_{i=1}^{n-1} q_i/p_i)\frac{\sum_{i=1}^{n-1} p_i}{1 - l(m-n+1)}$$

and is increasing with ℓ. The maximum search size needed to support the allocation is approximately $1/\ell$ and is decreasing with ℓ. The overall search size (as defined in Equation 5) is a convex combination of the ASS and the MSS and is minimized inside the interval $[p_m, 1/m]$. Figure 4 illustrates the ASS, the MSS, and various combinations that correspond to different mix of soluble and insoluble queries. When most queries are soluble (f_s is close to 1), the minimum point is closer to p_m, where the ASS is minimized. When most queries are insoluble (f_s is close to 0), the minimum is obtained closer to $1/m$, where the MSS is minimized.

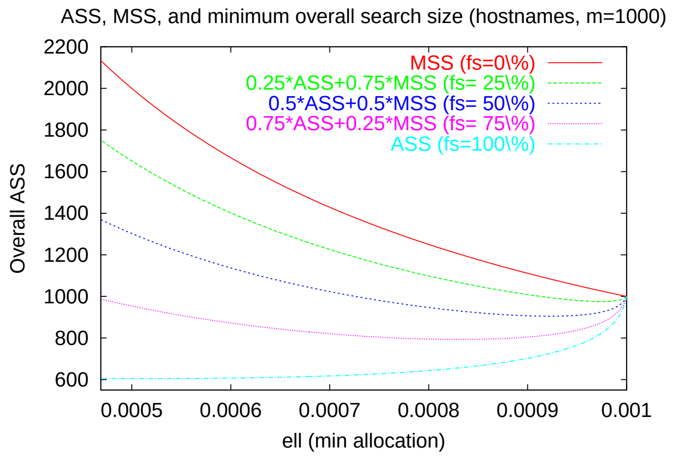


Fig. 4. Maximum search size, Average Search Size of Square-Root*, and Overall Search size as a function of ℓ; Using hostname frequency distributions and m=1000.

B. *Proportional* allocation*

We similarly can ask what is the strategy which minimizes the maximum utilization under these constraints; and similarly define Proportional* allocation which is defined for all query distributions and minimizes the maximum utilization rate.

Similarly, we extend Proportional allocation so it is applicable when $\ell \leq 1/m \leq u$. We define Proportional* allocation as the unique allocation defined by the following conditions:

- if $u > p_i, p_j > \ell$ then $p_i/p_j = q_i/q_j$.
- if $p_j = \ell \leq p_i$ or if $p_j \leq u = p_i$, then $p_i/p_j \leq q_i/q_j$.

When we have $\ell \leq q_i \leq u$ for all i then Proportional* is the same as Proportional. It follows from Theorem III.1 that Proportional* allocation has ASS no higher than Uniform allocation and when Proportional* is different than Proportional (that is, there are $q_i < \ell$ or $q_i > u$) then Proportional* has a lower ASS than Uniform. Proportional* allocations lies “in-between” Proportional and Uniform (in the sense of III.3).

VI. Distributed algorithms for Square-Root Allocation

Up till now, we gained understanding of what is the best replication strategy. In this section we show that it can be achieved, via a simple distributed protocol, in a decentralized unstructured P2P network.

The Uniform allocation can be obtained via a simple scheme which replicates each item in a fixed number of locations when it first enters the system. We had seen that the optimal allocation is a hybrid of Square-Root and Uniform allocations. We first show how to obtain a Square-Root allocation, and then discuss how these algorithms can be twicked to obtain optimal balance of the cost of soluble and insoluble queries. In order to exploration replication algorithms[3] aimed at Proportional and Square-Root allocations, we need to first model a dynamic setting where copies are created and deleted:

- **Creation:** New copies can be created after each query. After a successful random search, the requesting node creates some number of copies, call it C, at randomly-selected nodes. This number C can only depend on quantities locally observable to the requesting node.
- **Deletion:** Copies do not remain in the system forever; they can be deleted through many different mechanisms. All we assume here of copies is that item lifetimes are independent of the identity of the item and the survival probability is non-increasing with age: if two copies were generated at times t_1 and $t_2 > t_1$ then the second copy is more likely to still exist at time $t_3 > t_2$.[4]

The creation and deletion processes are consistent with our model and operation of unstructured networks. We assume the same communication patterns for copy creation and for the search process. Since search performance is modeled well by contacting random peers, we used the same model for evaluating copy creation. In actuality, the network would perform copy creation by visiting nodes in a similar fashion to how it performs search.[5] Our copy deletion process is consistent with the expectation that in deployed networks, copy deletion occurs either by a node going offline or by some replacement procedure internal to a node. We expect node crashes to be unrelated to the content (part of the index) they posses. The replacement procedure within each node should not discriminate copies based on access patterns. Two common policies, Least Recently Used (LRU) or Least Frequently Used (LFU) are inconsistent with our assumption, but other natural policies such as First In First Out (FIFO) or random deletions (when a new copy is created remove a cached copy selected at random), are consistent with it.

Let $\langle C_i \rangle$ be the average value of C that is used for creating copies of item i ($\langle C_i \rangle$ may change over time). We say that the system is in *steady state* when the lifetime distribution of copies

[3]A note on terminology: a replication strategy is a mapping between $\mathbf{q}$ and $\mathbf{p}$ whereas a replication algorithm is a distributed algorithm that realizes the desired replication strategy.

[4]Examples of deletion processes consistent with these assumptions are fixed life durations, random deletions where the deletion rate may vary over time, and FIFO replacement implemented at each location. Replacement policies not consistent with these assumptions are LRU and LFU, which both depend on the identities of the items.

[5]This process automatically adjust to designs where search is focused in a fraction of nodes, or when nodes are not evenly utilized, such as with FastTrack, since hosts recieve copies in the same rate that they receive search probes.

does not change over time. The property of this deletion process that is exploited by our algorithms is

Claim VI.1: If the ratio $\langle C_i\rangle/\langle C_j\rangle$ remains fixed over time and $\langle C_i\rangle, \langle C_j\rangle$ are bounded from below by some constant, then $p_i/p_j \to q_i\langle C_i\rangle/(q_j\langle C_j\rangle)$
Thus, Proportional allocation is obtained if we use the same value of C for all items.

A more challenging task is designing algorithms that result in Square-Root allocation. The challenge in achieving this is that no individual node issues enough queries to estimate the query rate q_i, and we would like to determine C without using additional inter-host communication on top of what is required for the search and replication. Algorithms that use additional communication conflict the nature of current P2P implementations. We can use the above claim to obtain the following condition for Square-Root allocation

Corollary VI.1: If $\langle C_i\rangle \propto 1/\sqrt{q_i}$ then $p_i/p_j \to \sqrt{q_i/q_j}$ (the proportion factor may vary with time but should be the same for all items).

We propose three algorithms that achieve the above property but do not require additional communication or infrastructure. The algorithms require different amounts of bookkeeping, and so which algorithm is more desirable in practice will depend on the details of the deployment setting. The first algorithm, *path replication*, uses no additional bookkeeping; the second, *replication with sibling-number memory*, records with each copy the value C (number of sibling copies) used in its creation; the third *probe memory* has every node record information about every probe. We show that under some reasonable conditions, sibling-number and probe memory have $\langle C_i\rangle$ close to its desired value and path replication has $\langle C_i\rangle$ converge over time to its desired value.

A. Path replication

Recall that the search size for item i is a Geometric random variable. At any given time, the average search size for item i, A_i is inversely proportional to the allocation p_i. If $\langle C_i\rangle$ is steady then $p_i \propto q_i\langle C_i\rangle$ and thus $A_i \propto 1/(q_i\langle C_i\rangle)$. The *Path replication* algorithm sets the number of new copies C to be the size of the search (the number of nodes probed). At the fixed point, when A_i and $\langle C_i\rangle$ are steady and equal, they are proportional to $1/\sqrt{q_i}$; and p_i is proportional to $\sqrt{q_i}$.

We show that in steady state, under reasonable conditions, A_i and $\langle C_i\rangle$ converge to this fixed point. Let $\overline{A}_i$ be the value of A_i at the fixed point. At a given point in time, the rate of generating new copies is $A_i/\overline{A}_i$ times the optimal. This means that the rate is higher (respectively, lower) than at the fixed point when the number of copies is lower (respectively, higher) than at the fixed point. If the time between queries is at least of the order of the time between a search and subsequent copy generation then path replication will converge to its fixed points.

A possible disadvantage of path replication is that the current C_i "overshoots" or "undershoots" the fixed point by a large factor $(A_i/\overline{A_i})$. Thus, if queries arrive in large bursts or if the time between search and subsequent copy generation is large compared to the query rate then the number of copies can fluctuate from too-few to too-many and never reach the fixed point. Note that this convergence issue may occur even for a large number of nodes. We next consider different algorithms that circumvents this issue by using $\langle C_i\rangle$ that is close to this fixed-point value.

B. Replication with sibling-number memory

Observe that the search size is not sufficient for estimating the query rate q_i at any point in time (path replication only reached the appropriate estimates at the fixed point). In order to have $\langle C_i\rangle \propto 1/\sqrt{q_i}$ we must use some additional bookkeeping. We assume that with each copy we record the number of "sibling copies" that were generated when it was generated and its generation time. The algorithm assumes that each node known the lifetime distribution as a function of the age of the copy (thus, according to our assumptions, the "age" of the copy provides the sibling survival rate).

Consider some past query for the item, a_j, let $d_j > 0$ be the number of sibling copies generated after a_j and let λ_j be the expected fraction of surviving copies at the current time.

Suppose that every copy stores (d, λ) as above. Let T be such that copies generated T time units ago have a positive probability of survival. Let P_T be the set of copies with age at most T. Then, $\sum_{c\in P_T} 1/(\lambda_c d_c)$ is an unbiased estimator for the number of requests for the item in the past T time units. Denote by $\langle 1/(\lambda_c d_c)\rangle$ the expectation of $1/(d\lambda)$ over copies.

We thus obtain that

$$q_i \propto \langle 1/(\lambda_c d_c)\rangle p_i \propto \langle 1/(\lambda_c d_c)\rangle (1/A_i)$$

Hence, it suffices to choose C_i with expected value $\langle C_i\rangle \propto (1/\langle 1/(\lambda_c d_c)\rangle)^{0.5} A_i^{0.5}$.

C. Replication with probe memory

The rate in which a host in the network receives probes for item i is $q_i/(\rho p_i)$, and the expected size of the search is $1/(\rho p_i)$, Evidently, q_i can be estimated from estimates on these two quantities. Consider a time duration in which p_i does not significantly change and consider a host v that receives probes for item i. Let $s_1, \ldots, s_{k_v}$ be the number of nodes probed before v on each search, and let $S_v = \sum_{j=1}^{k_v} s_j/k_v$ be their average [6]. Suppose that each node stores (k_v, S_v) for each item. We obtain that k_v/S_v is a (biased) estimator for a quantity proportional to q_i (the bias decreases with k_v). Better estimates be obtained by aggregation, for example, if the querying node collects this information from all nodes on the search path. If $v_1, \ldots, v_k$ are the path nodes then $(\sum k_{v_j})^2/\sum k_{v_j} S_{v_j}$ constitute an estimator for (a value proportional to) q_i. We thus use $(\sum k_{v_j} S_{v_j})^{0.5}/\sum k_{v_j}$ as a (biased) estimator for $1/\sqrt{q_i}$.

D. Obtaining the optimal allocation

The algorithms presented above converge to Square-root allocation, but can be adjusted so that they result in the optimal allocation. One way to obtain an optimal allocation is to tune the maximum search size allowed. Adjustment of the maximum search size alone, however, would result in changing the set of locatable items. In order to arrive at optimal allocation

[6]Note that S_v in fact estimates half of the ASS. If s is the search-size so far then $2s$ is an unbiased estimator for the size of the full search. Since this calculation affects all items equally (same constant) the allocation would still converge to the same one.

while fixing the set of locatable items we need to use a hybrid of Uniform and Square-root algorithms: basically, we use "permanent copies" generated by the Uniform-allocation algorithm and "transient copies" generated by the Square-root-allocation algorithm. The Uniform algorithms allocates permanent copies, where each item is guaranteed some minimum allocation (e.g., by nominating nodes that never delete it and are replaced when they go offline). The square-root algorithm can then be applied on top of this uniform allocation of permanent copies. The maximum search size is determined according to the minimum allocation, so that items with minimum allocation are locatable; The optimal allocation is achieved by tuning the value of the minimum allocation according to the fraction of queries that are insoluble.

E. Simulations

We simulated two of the proposed "Square-Root" replication algorithms: path replication, and replication with sibling-number memory. The simulations track the fraction of nodes containing copies of a single item in a network with 10K nodes. In our simulations, copies had a fixed lifetime duration. Queries are issued in fixed intervals. Every search is repeated until k copies are found ($k \in \{1, 5\}$). For path replication we take the average search size s and for replication with sibling-number memory we used the estimators discussed above. We simulated various delay durations between the time a search is performed and the time the copies are generated.

Some of these simulations are shown in Figures 5 and 6. The figures illustrate the convergence of "Sibling-Memory" and "path replication" algorithms by showing how the fraction of nodes with copies evolves over time. The figures illustrate the convergence issued discussed above. The sibling-memory algorithm arrives more quickly to the desired allocation. It also also not sensitive to delay in the creation of copies.

VII. Appendix

Proof: [remaining proof of Theorem III.1] We now prove the claim. We start with the first case ($\mathbf{p}$ has the form 1). We use the shorthand $T = \sum_{i=k+1}^{m} p_i$ for the total allocation to the "tail" items and $\psi = \sum_{i=k+1}^{m} q_i/p_i$ for the contribution of the tail items to the ASS. Similarly, we use $T' = \sum_{i=k+1}^{m} p'_i$ and similarly define T'', ψ', and ψ''. Since the relative allocation to items in the "tail" is the same for the allocations $\mathbf{p}$, $\mathbf{p}'$, and $\mathbf{p}''$, we have $\psi' = \psi T/T'$ and $\psi'' = \psi T/T''$.

We now express the respective average search sizes s, s', and s'', as a function of p_1 and ψ. For the allocation $\mathbf{p}$ we obtain:

$$s = 1/p_1 \sum_{i=1}^{k} q_i + \psi$$

For s' we obtain

$$s' = (1/p'_1) \sum_{i=1}^{k} q_i + \psi' = (1/p'_1) \sum_{i=1}^{k} q_i + \psi T/T' \quad (6)$$

The sum of allocations must be 1, thus

$$kp_1 + T = kp'_1 + T' = 1 \quad (7)$$

Since the relative allocations on the tail are the same for $\mathbf{p}$, $\mathbf{p}'$ and $\mathbf{p}''$ we obtain

$$T/T' = p_{k+1}/p'_{k+1} = p_1(q_{k+1}/q_k)/p'_1 \ . \quad (8)$$

From Equations 7 and 8 we obtain that

$$1/p'_1 = k + q_k/(q_{k+1}p_1) - kq_k/q_{k_1} \ . \quad (9)$$

and

$$T/T' = (q_{k+1}/q_k)p_1/p'_1 = (1 - p_1 k(1 - q_{k+1}/q_k)) \quad (10)$$

By substituting Equations 9 and 10 in 6 we obtain

$$s' = (k+q_k/(p_1 q_{k+1})-kq_k/q_{k+1}) \sum_{i=1}^{k} q_i + \psi(1-p_1 k(1-q_{k+1}/q_k))$$

thus

$$s' - s = (q_k/q_{k+1} - 1)(1/p_1 - k) \sum_{i=1}^{k} q_i - \psi(p_1 k(1-q_{k+1}/q_k))$$

If $s' - s < 0$ ($\mathbf{p}'$ is strictly better than $\mathbf{p}$) we obtain that

$$\psi > \frac{(q_k/q_{k+1} - 1)(1/p_1 - k)\sum_{i=1}^{k} q_i}{p_1 k(1 - q_{k+1}/q_k)} = \frac{q_k(1/p_1 - k)\sum q_i}{kp_1 q_{k+1}} \quad (11)$$

We now express s'' in terms of p_1 and ψ.

$$s'' = \sum_{i=1}^{k} q_i/p''_i + \psi'' = kq_1/p''_1 + \psi T/T'' \quad (12)$$

We have

$$kp_1 + T = \sum_{i=1}^{k} p''_i + T'' = p''_1 \sum_{i=1}^{k} q_i/q_1 + T'' = 1 \quad (13)$$

and

$$T/T'' = p_{k+1}/p''_{k+1} = p_1(q_{k+1}/q_k)/(p''_1 q_{k+1}/q_1) = (q_1/q_k)p_1/p''_1 \ . \quad (14)$$

From Equations 13 and 14 we obtain

$$1/p''_1 = \sum_{i=1}^{k} q_i/q_1 - kq_k/q_1 + q_k/(q_1 p_1) \quad (15)$$

and

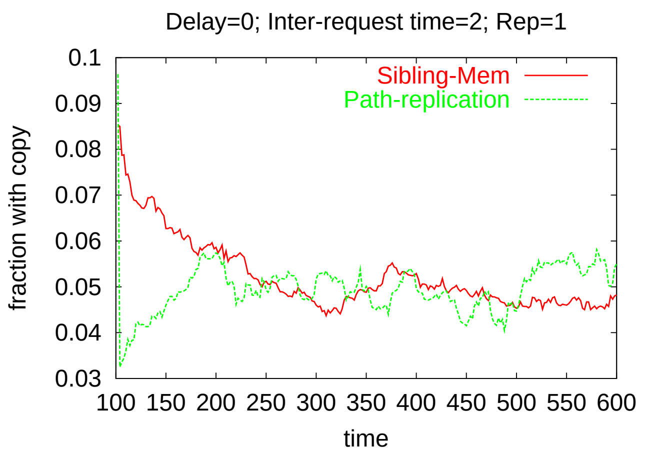
$$T/T'' = 1 - kp_1 + p_1 \sum_{i=1}^{k} q_i/q_k \ . \quad (16)$$

Substituting 15 and 16 in 12 we obtain

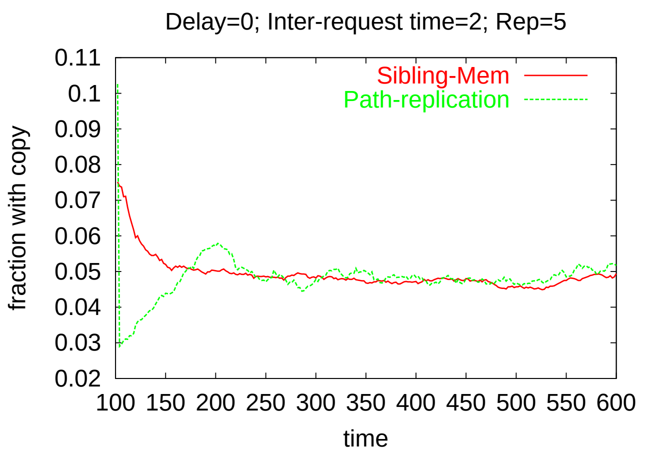
$$s'' = k \sum_{i=1}^{k} q_i - k^2 q_k + kq_k/p_1 + \psi(1 - kp_1 + p_1 \sum_{i=1}^{k} q_i/q_k) \ .$$

Thus,

$$s'' - s = (\sum_{i=1}^{k} q_i - kq_k)(k - 1/p_1) + \psi p_1(\sum_{i=1}^{k} q_i/q_k - k) \ .$$

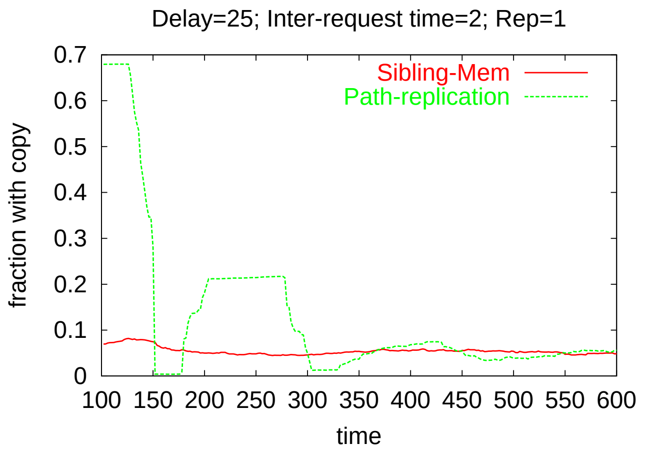


1 sample

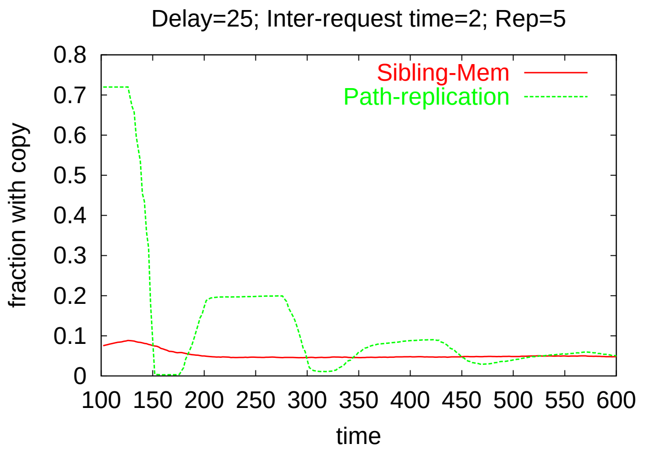


5 samples

Fig. 5. Simulating performance of Path Replication and Sibling-Memory algorithms. In these simulations there is no delay in copy creation; the copy lifetime is 100 time units; and the inter-request-time is 2 time units



1 sample



5 samples

Fig. 6. Performance of Path Replication and Sibling-Memory replication algorithms. In these simulation there is delay of 25 time units in copy creation; the copy lifetime is 100 time units; and the inter-request time is 2.

The second term is always nonnegative. The first term is non-negative if $k > 1/p_1$. If $s'' - s < 0$ ($\mathbf{p}''$ has a shorter ASS than $\mathbf{p}$) we obtain

$$\psi < q_k(1/p_1 - k)/p_1 \; . \tag{17}$$

If both $s'' < s$ and $s' < s$ we obtain from 11 and 17 that

$$\frac{q_k(1/p_1 - k)\sum q_i}{kp_1 q_{k+1}} < \psi < q_k(1/p_1 - k)/p_1 \; .$$

Thus, $q_{k+1} < \sum_{i=1}^{k} q_i/k$, which is a contradiction (recall that $q_{k+1} \leq q_i$ when $i < k+1$).

Note that strict equality occurs only when $s = s' = s''$ and is only possible when $q_1 = \cdots = q_{k+1}$, which contradicts our assumption that q_{k+1}/q_k was inconsistent with previous relations.

We now apply similar arguments to handle the second case (where $\mathbf{p}$ has the form 2). In this case we have

$$s = \sum_{i=1}^{k} q_i/p_i + \psi = kq_1/p_1 + \psi \; . \tag{18}$$

The sum of allocations in $\mathbf{p}$ and $\mathbf{p}'$ must satisfy

$$p_1/q_1 \sum_{i=1}^{k} q_i + T = kp_1' + T' = 1 \; . \tag{19}$$

We have

$$T/T' = p_{k+1}/p_{k+1}' = q_k p_1/(q_1 p_1') \tag{20}$$

From Equations 19 and 20 we obtain

$$(p_1')^{-1} = q_1/(p_1 q_k) - q_k^{-1} \sum_{i=1}^{k} q_i + k \tag{21}$$

and

$$T/T' = 1 - (p_1/q_1) \sum_{i=1}^{k} q_i + kp_1 q_k/q_1 \tag{22}$$

Substituting 21 and 22 in 6 we obtain

$$s' = \frac{q_1}{p_1 q_k} \sum_{i=1}^{k} q_i - \frac{1}{q_k} (\sum_{i=1}^{k} q_i)^2 + \psi \left(1 - \frac{p_1 q_k}{q_1} (q_k^{-1} \sum_{i=1}^{k} q_i - k) \right)$$

Thus,

$$s' - s = (q_k^{-1}\sum_{i=1}^{k} q_i - k)\left((q_1/p_1 - \sum_{i=1}^{k} q_i) - \psi p_1 q_k/q_i\right) .$$

If $s'-s < 0$ we obtain that (note that since q_i are monotone non-increasing and they are not all equal we have $q_k^{-1}\sum_{i=1}^{k} q_i > k$)

$$\psi > \frac{q_1}{p_1 q_k}(\frac{q_1}{p_1} - \sum_{i=1}^{k} q_i) . \tag{23}$$

Repeating the same steps for $\mathbf{p}''$ we have

$$T + \frac{p_1}{q_1}\sum_{i=1}^{k} q_i) = T'' + \frac{p_1''}{q_1}\sum_{i=1}^{k} q_i) = 1 \tag{24}$$

and

$$\frac{T}{T''} = \frac{p_{k+1}}{p''_{k+1}} = \frac{p_1 q_k/q_1}{p''_1 q_{k+1}/q_1} = \frac{p_1 q_k}{p''_1 q_{k+1}} \tag{25}$$

From Equations 24 and 25 we obtain

$$(p''_1)^{-1} = \frac{q_k - q_{k+1}}{q_1 q_k}\sum_{i=1}^{k} q_i + \frac{q_{k+1}}{p_1 q_k} \tag{26}$$

and

$$\frac{T}{T''} = 1 + \frac{p_1(q_k - q_{k+1})}{q_1 q_{k+1}}\sum_{i=1}^{k} q_i \tag{27}$$

Substituting Equations 21 and 27 in 12 we get

$$s'' = \frac{k(q_k - q_{k+1})}{q_k}\sum_{i=1}^{k} q_i + \frac{k q_1 q_{k+1}}{p_1 q_k} + \psi\left(1 + \frac{p_1(q_k - q_{k+1})}{q_1 q_{k+1}}\sum_{i=1}^{k} q_i\right)$$

thus,

$$s''-s = (1-\frac{q_{k+1}}{q_k})(k\sum_{i=1}^{k} q_i - \frac{k q_1}{p_1}) + \psi\frac{p_1 q_k}{q_1 q_{k+1}}\sum_{i=1}^{k} q_i(1-\frac{q_{k+1}}{q_k}) .$$

Hence, if $s'' - s < 0$ we obtain (recall that $q_{k+1} < q_k$ and thus $(1 - \frac{q_{k+1}}{q_k}) > 0$.)

$$\psi < \frac{k q_1 q_{k+1}}{p_1 q_k \sum_{i=1}^{k} q_i}\left(\frac{q_1}{p_1} - \sum_{i=1}^{k} q_i\right) \tag{28}$$

and, since ψ is nonnegative, we have

$$(\frac{q_1}{p_1} - \sum_{i=1}^{k} q_i) \geq 0 . \tag{29}$$

Assume to the contrary that $s'' < s$ and $s' < s$. We obtain a contradiction from Equations 23,28,29. ■

Proof: [Lemma IV.2] The average search size with Uniform allocation is m/ρ for any choice of $\mathbf{q}$. The search size with optimal allocation is at most

$$\max_{\mathbf{q}|\ell \leq \sqrt{q_i}/\sum\sqrt{q_i} \leq u} 1/\rho(\sum_{i=1}^{m}\sqrt{q_i})^2 .$$

We are now interested in the minimum of $\sum\sqrt{q_i}$ over the $(m-1)$-simplex intersected with the cube $\ell \leq q_i \leq u$. The function is concave and maximized at an interior point. Minima are obtained at vertices, which are defined by intersection of $m-1$ dimensional faces of the cube with the simplex. Algebraically, the function is minimized when allocations are either at u or at ℓ. [7] Solving $x\ell + (m-x)u = 1$ we obtain that $x = (um-1)/(u-\ell)$ items have allocation ℓ and the remaining $m - x = (1-m\ell)/(u-\ell)$ items have allocations u. Let q_ℓ (respectively, q_u) be the popularity of items obtaining allocation ℓ (respectively, u). We have $xq_\ell + (m-x)q_u = 1$ and

$$\sqrt{q_\ell}/(x\sqrt{q_\ell} + (m-x)\sqrt{q_u}) = \ell .$$

Substituting x and solving the above we obtain

$$q_\ell = \frac{\ell^2}{u + \ell - m\ell u}$$

$$q_u = \frac{u^2}{u + \ell - m\ell u}$$

Thus, at the minimum point,

$$\sum\sqrt{q_i} = x\sqrt{q_\ell} + (m-x)\sqrt{q_u} = 1/\sqrt{u + \ell - m\ell u} .$$

■

Proof: [Lemma V.1] Consider a monotone allocation $\mathbf{p}$ for which one of these conditions does not apply. A simple case analysis establishes that a legal allocation with smaller ASS can be obtained by "locally" reallocating $p_i + p_j$ among p_i and p_j (increasing one and decreasing the other by the same amount). Thus, these conditions are necessary for an optimal allocation.

We now focus on the set of legal allocations for which condition 1 hold. These allocations have a (possibly empty) prefix of u's and (possibly empty) suffix of ℓ's and the middle part is square-root allocated and has values between u and ℓ. As argued above, the optimal allocation must be of this form.

We first argue that there must be at least one allocation in this set by explicitly defining it: Let $x = \lceil(1-\ell)m/(u-\ell)\rceil$ be the number of u's and $m - x - 1$ be the number of ℓ's in the suffix. There is at most one item in the middle part, and it has allocation between u and ℓ.

We next show that any allocation of this form for which condition 2 does not hold, has a *neighbor* allocation of that form with a lower ASS. By neighbor allocation we refer to one where the suffix or prefix lengths differ by at most one. To prove this, consider the case where $p_j = \ell \leq p_i$ and $p_i/\ell > \sqrt{q_i/q_j}$ (the other case is similar). If this is true for any such j and i then it is true for j being the minimal for which $p_j = \ell$ and $i = j-1$. If we add j to the "middle part" and recalculate square-root allocation then it is not hard to see that we get a better allocation with values between u and ℓ. If $p_{j-1} = u$ (there was no middle part) we place both p_{j-1} and p_j in a newly-created middle part.

It remains to show that there is only one local minima to this process of moving to a better neighbor. Consider an allocation. Let j be the last index for which $p_j = u$. Let $k+1$ be the first

[7]There can be one item which lies in between these two limits (when x is not integral), but this adds a small term which we ignore.

index for which $p_{k+1} = \ell$. The optimality conditions imply that $u\sqrt{q_{j+1}/q_j} \leq p_{j+1} < u$ and $\ell\sqrt{q_k/q_{k+1}} \geq p_k > \ell$. We show that if there are two different solutions that fulfill the optimality conditions then we get a contradiction. Let j, k and j', k' be the corresponding indexes and p, p' the corresponding allocations. We first consider the case where $j = j'$ and $k < k'$. We have $p'_{k+1}, \ldots, p'_{k'} > \ell = p_{k+1}, \ldots, p_{k'}$. As allocations sum to 1 we have $p'_j < p_j$. We must have $p'_j/p'_{k+1} = \sqrt{q_j/q_{k+1}}$. On the other hand, we also must have $p_j/p_{k+1} \leq \sqrt{q_j/q_{k+1}}$. Thus, $p'_j/p'_{k+1} \geq p_j/p_{k+1}$, which is a contradiction. The claims for other cases are similar: If $j' < j < k < k'$ we consider the items j' and k'. We have $p'_{j'} < u$ and $p'_{k'} > \ell$ thus we must have $p'_{j'}/p'_{k'} = \sqrt{q_{j'}/q_{k'}} < u/\ell$. On the other hand, the assumed optimality of the other allocation implies that $\sqrt{q_{j'}/q_{k'}} > u/\ell$. Another case is $j' < j < k' < k$. We consider the total allocation to items $j, \ldots, k'$. If it is larger in p' we get a contradiction with the allocation of p'_i $(i < j)$. If it is larger we get a contradiction by looking at p_i $(i \geq k' + 1)$. ■

References

[1] Open Source Community. The free network project - rewiring the internet. In *http://freenet.sourceforge.net/*, 2001.

[2] Open Source Community. Gnutella. In *http://gnutella.wego.com/*, 2001.

[3] KaZaA file sharing network. KaZaA. In *http://www.kazaa.com/*, 2002.

[4] Morpheus file sharing system. Morpheus. In *http://www.musiccity.com/*, 2002.

[5] Napster Inc. The napster homepage. In *http://www.napster.com/*, 2001.

[6] L. Kleinrock. *Queueing Systems, Volume II: Computer Applications*. Wiley-Interscience, New York, 1976.

[7] C. Lv, P. Cao, E. Cohen, E. Felten, and S. Shenker. Search and replication in unstructured peer-to-peer networks. 2001. Submitted for publication.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, RichardKarp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM'2001*, August 2001.

[9] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of SOSP'01*, 2001.

[10] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM'2001*, August 2001.

[11] FastTrack Peer-to-Peer technology company. FastTrack. In *http://www.fasttrack.nu/*, 2001.

[12] Ben Y. Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.