Proportional Replication in Peer-to-Peer Networks

Saurabh Tewari, Leonard Kleinrock
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095, U.S.A.
{stewari, lk}@cs.ucla.edu

Abstract— **We recently showed for peer-to-peer networks, that having the number of replicas of each object proportional to the request rate for these objects has many per-node advantages. In this paper we complement those results to show that this distribution has network-wide advantages as well. Given these benefits of proportional replication, the next issue is achieving proportional replication in a decentralized manner. We show that local storage management algorithms like LRU automatically achieve near-proportional replication and that the system performance with the replica distribution achieved by LRU is very close to optimal. We also show that the LRU responds to a change in user access pattern quickly (the number of accesses taken to reach the new steady-state replica distribution with LRU is close to the minimum possible with any cache replacement algorithm). Analytical models are provided for computing the steady-state network-wide replica distribution and the transient period for LRU.**

Keywords- Peer-to-Peer, File Replication, Cache Management, LRU, Network Bandwidth, Proportional Replication

I. INTRODUCTION

Peer-to-peer networks offer the 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 to 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. So, for example in peer-to-peer file sharing networks, users share files and if one wants to download a file and another user is sharing that file, one would download it directly from that user. Upon obtaining the desired file, one may also begin to share that file allowing other users to download from them. Thus, a file is likely to have multiple replicas in the network with the more popular files having more replicas (i.e. more sources to download the file from). The replication of files provides the robustness while its correlation with popularity provides the automatic scaling according usage patterns. Since music file sharing over the Internet is the most popular peer-to-peer networking application, characteristics associated with music file sharing (e.g. free-riding, short node lifetimes, large variation in user connection bandwidth and user shared storage, no limits on user storage allocated for downloaded content) are usually associated with all peer-to-peer networking. However, many other applications can benefit from peer-to-peer networking. For example, peer-to-peer networking can be used to offload load from the service provider video server in a video-on-demand service. One may also conceive of use of peer-to-peer networking in a shared digital library application (for example, a state-funded library for all K-12 schools in a state) whereby individual schools (or school districts) dedicate fixed amounts of storage for the application; the content is brought in upon a user request and is kept in the storage and is available to other users at other sites. For applications such as these, the presence of an intermediary and/or similarity in the user group simplifies the assumptions for the peer-to-peer system and it is not unreasonable to assume that users are similar and well-behaved (i.e. no free-riding, little variation in interests and resources of each user, long node lifetimes). If the number of users is not very large and the user caches are available for long time durations, a centralized solution for the search problem may be feasible. Hence, in this paper, we ignore the issue of search costs in our peer-to-peer system, concentrating exclusively on the downloading aspects of the peer-to-peer system. [18] showed that when the number of replicas of each object is proportional to the request rate for that object, a user has no advantage in sharing one file over the other (as the download load for each file is equal), the total download load on each node is same and when queueing delays are convex in node utilization, the average queueing delay seen by a downloader is minimized. In this paper, we focus on system performance aspects and show that in addition to these user-centric advantages, the proportional replication distribution also minimizes the network bandwidth used (measured as the average number of links traversed in a download). The system model is discussed in Section 2 and the proof of the optimality of the proportional replication for the network bandwidth used is presented in Section 3.

Given these benefits of proportional replication, we devote much of this paper seeking distributed mechanisms to achieve such a replication. In Section 4, we discuss the performance of some existing cache management algorithms such as LRU, LFU, and FIFO. Our simulations show that these algorithms achieve near-proportional replica distribution. We construct a cache management algorithm that achieves the proportional replication and compare system performance with this algorithm to the system performance when cache management policy is LRU. Our simulations show that the performance with LRU is only slightly inferior to the optimal performance. Therefore, we study the behavior of LRU in peer-to-peer environments in more detail in Section 5 where an analytical model is provided to compute the replica distribution LRU will achieve in different situations which can then be used to estimate the system performance.

As user interests change over time, the replica distribution should adapt to the new request rates. Therefore, the utility of a replica distribution algorithm also depends on its ability to converge to the new steady-state distribution quickly and/or maintain adequate performance during the transient period. We address this issue in Section 6 where our simulations show that, upon a change in the access pattern, LRU converges to the new steady-state replica distribution faster than the cache management algorithm that achieves linearly proportional replication. We also provide a preliminary analytical model for estimating the duration of the warm-up transient period (i.e. the time taken to reach the steady-state replica distribution starting with an empty buffer) and the replica distribution during this transient period for LRU. This model can be used to estimate the transient performance with LRU cache management by making similar extensions as in [3].

Some related work is briefly discussed in Section 7 and Section 8 concludes the paper.

II. System Model

Our abstract peer-to-peer system model is shown in Fig. 1. The broadband network can have any topology. Our only assumption is that the network topology has exponential expansion [15]. As discussed in [15], many commonly used Internet topology models fit this description. The central server is optional and shown only to signify that a file never disappears from the system as a result of cache replacement. Our simulations do not include the central server and at least one replica of each file in the system is maintained by assigning a peer ("origin server") for each file which must always keep the file in its cache.

We assume that there are M peers in the system (the terms peers, users and nodes are used interchangeably in this paper). There are N unique files in the system (the term file represents any generic object that may be downloaded), each with an associated request rate λ_i for file i per node (the request rates are uniform across nodes). 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. A file may have multiple replicas in the system (i.e. $n_i \geq 1$ where n_i is the number of replicas of file i in the system). Thus, a node will always find a file it is looking for. The specifics of the search mechanism are not important as long as the download requests for file i are equally distributed over the n_i replicas of file i in the system. We estimate the network bandwidth used in downloading a file by the average number of links along the shortest path to the nearest replica of the file. The notation for the various system parameters discussed is:

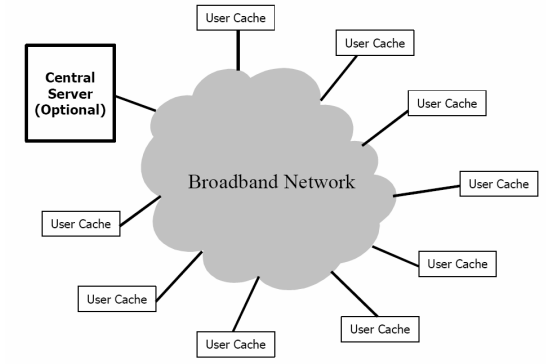

Figure 1. Peer-to-Peer System

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

V = number of nodes in the underlying link-level topology

$\tau_i(n_i)$ = average number of links to nearest replica for file i when there are n_i replicas of the file in the system.

III. Benefits of Proportional Replication

As discussed earlier, [18] showed that selecting $n_i \propto \lambda_i$ as the replica distribution offers significant user-level benefits. In this paper, we show that this distribution has system-level benefits as well. We focus on the average number of links traversed per download as our metric for system performance since it provides us with an estimate of the network bandwidth that each download "consumes". If the objective is to minimize the network bandwidth used, the download source should be the nearest replica if multiple replicas of the file are available. We derive an expression for the relation between the average number of links to the nearest replica of a file to the number of replicas of the file assuming that the replicas are uniformly distributed over the network. Using the derived expression, we formulate and solve our optimization problem to find the optimum replica distribution is, once again, $n_i \propto \lambda_i$. We then briefly address our assumption of uniform distribution of replicas and show (via simulations) that if all peers have the same request rates, cache management automatically results in a uniform distribution of the replicas over the network.

A. Link Distance to the Nearest Replica

Most of the popular topological models of the Internet and several other common topologies have the property of *exponential expansion* (i.e. the number of unique nodes reached within a hop distance h is exponentially related to the hop distance) [15]. Therefore, we assume our link-level network topology to have this exponential expansion as well. Clearly, not every node on the link level topology will be a participating peer. We assume that the participating peers are uniformly distributed over the entire network. The following theorem states our main result that τ_i, the average number of link-level hops to the nearest replica of file i, is logarithmically related to n_i, the number of replicas of file i, when the underlying link-level topology has an exponential expansion.

Theorem 1:

For a peer-to-peer network of size M where the underlying link-level topology has an exponential expansion, i.e. the number of nodes reachable in h hops is kd^h where k and d are

constants based on the link-level topology and the M peers are uniformly distributed over the link-level topology, for large networks (i.e. as $M \to \infty$) $\tau_i(n_i)$, the average number of links traversed in downloading file i from its nearest source, is related to the number of replicas of file i, n_i, as follows:

$$\tau_i(n_i) = \log_d(M/n_i) + C \quad (1)$$

(where C is constant) for finite n_i, assuming that the n_i replicas of a file are uniformly distributed over the participating peers.

Proof:

We wish to calculate the expected number of link-level hops to the nearest replica of the file to be downloaded given the number of replicas of that file in the network.

Assuming that each node of the link-level topology graph is equally-likely to be a participant in the peer-to-peer network, if there are V nodes in the link-level topology graph, the probability that a randomly selected node is a participant in the peer-to-peer network is M/V. Let S_h be the expected number of participating peers reachable in h hops. Since the underlying link-level topology has exponential expansion, kd^h link-level topology graph nodes (where k and d are constants based on the link-level topology) can be reached in h hops. These kd^h nodes are participating peers with probability M/V. Therefore,

$$S_h = (M/V)kd^h \quad (2)$$

Assuming that the replicas of a file are uniformly distributed in the network, the probability of finding file i at a randomly selected node is n_i/M when there are n_i replicas of the file in the network. In addition to the notation defined earlier, define: P_h as the probability that, from a randomly selected requesting node, the nearest replica of file i is available exactly at h link-level hops and F_h as the probability that no peer within h link-level hops of that requesting node has file i.

Therefore, $P_h = F_{h-1} - F_h$. The average link-level hop distance to the nearest replica is: $\tau_i(n_i) = \sum_{h=0}^{H_M} hP_h$ where H_M is the link-level distance within which all the M peers can be reached. 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 assumption 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 $F_h = (1 - n_i/M)^{S_h}$. Since, $n_i \geq 1$, F_{H_M}, the probability that the file is not found even after probing all nodes, is zero. Therefore,

$$\tau_i(n_i) = \sum_{h=0}^{H_M - 1} (1 - n_i/M)^{\frac{M}{V} kd^h}$$

Using the Euler-Maclaurin summation formula: $\tau_i(n_i) = \int_0^{H_M} (1 - n_i/M)^{\frac{M}{V} kd^h} dh + \sum_{k=1}^{n} \frac{B_k}{k!}[f^{(k-1)}(H_M) - f^{(k-1)}(0)] + R_{f,n}$

where B_k are the Bernoulli numbers, $f(h) = (1 - n_i/M)^{\frac{M}{V} kd^h}$, $f^{(k)}(h)$ is the k^{th} derivative of $f(h)$ and $R_{f,n}$ is the remainder term in the summation for function $f(h)$.

Define $t = (M/V)kd^h$. Therefore, $h = \log_d(Vt/kM)$ and, hence, $dh = dt/(t \ln d)$. At $h = H_M$, $t = M$ and at $h = 0$, $t = 1$ (the peer downloading the file is a participating peer). Therefore: $\tau_i(n_i) = \frac{1}{\ln d} \int_1^M \frac{(1 - n_i/M)^t}{t} dt + \sum_{k=1}^{n} \frac{B_k}{k!}[f^{(k-1)}(H_M) - f^{(k-1)}(0)] + R_{f,n}$

Using the Euler-Maclaurin summation formula again, we get: $\tau_i(n_i) =$

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

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

Since, $B_k = 0$ for odd k (other than 1) and $B_4 = 1/720$, we can neglect the higher-order terms beyond $k = 2$ and the remainder term in the Euler-Maclaurin summation formulas. The required $g(t)$, $f(h)$ and $g^{(1)}(t)$, $f^{(1)}(h)$ terms at the limits can be evaluated as:

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

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

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

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

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

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

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

$f^{(1)}(h) = \ln(1 - n_i/M) f(h) d^h \ln d$

Hence, $f^{(1)}(0) = (1 - n_i/M)\ln(1 - n_i/M) \ln d$

$f^{(1)}(H_M) = M \ln d (1 - n_i/M)^M \ln(1 - n_i/M)$

As discussed earlier, the probability that file is not found even after probing all nodes $F_{H_M} = (1 - n_i/M)^M = 0$. Hence, $g(M)$, $g^{(1)}(M)$, $f(H_M)$, $f^{(1)}(H_M)$ are all 0. Therefore,

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

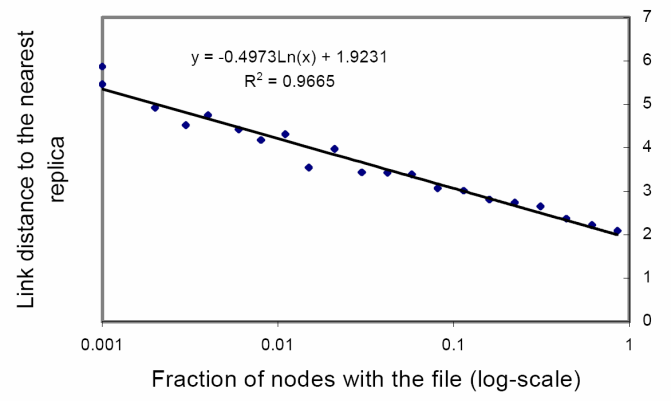

Figure 2. Link distance to the Nearest Replica (1000 peers, link topology: 10,254-node PLRG with rank exponent 2.25)

As $M \to \infty$, applying the series summation $\sum_{k=1}^{\infty} \frac{(1-1/x)^k}{k}$ $= \ln x$, we can write $\lim_{M\to\infty} \sum_{t=1}^{M-1} \frac{(1-n_i/M)^t}{t} = \ln(M/n_i)$. 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:

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

❑ Q.E.D.

The above theorem establishes the logarithmic relation of τ_i, the average number of link-level hops to the nearest replica of file i, to n_i, the number of replicas of file i for $M \to \infty$. Our simulations indicate that the relation holds for smaller values of M also. In Fig. 2, we show the simulation results for a peer-to-peer network of 1000 peers with an underlying 10,254-node PLRG (power law random graph [6]) link-level topology of power-law rank-exponent 2.25 (parameters motivated by [6], [15]) with the participating peers chosen among the link-level topology nodes uniformly at random, and the targeted number of replicas placed at the participating peers selected uniformly at random. The link distances shown are the average distance from each participating peer to the nearest replica. Other topologies with exponential expansion give similar results.

B. *Minimization of Network Bandwidth Used*

Our objective is to find the optimum value for n_i, the number of replicas of file i (for all i), which minimizes τ, the average number of links traversed in satisfying file requests, i.e.,

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

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:

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

The constraint in (5) states that the total number of replicas of all the files should not exceed the total storage available. Since there is no benefit derived from having multiple replicas of a file at a node, 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 –

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

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

This is a straightforward optimization problem and we state the solution as the following theorem:

Theorem 2:

For a peer-to-peer network whose underlying link-level topology is such that the number of links traversed in downloading file i from its nearest source, is related to the number of replicas of file i, n_i, as $\tau_i(n_i) = \alpha \log_d(M/n_i) + C$, and the storage capacity at each node is equal, then the network bandwidth used (measured as the average number of links traversed in a download) 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, ($i = 1,2, \ldots, N$) i.e.

(i) The number of replicas of each file n_i is proportional to the file request rate λ_i i.e.

$$n_i \propto \lambda_i \qquad (8)$$

if $\frac{1}{KM} \le \frac{\lambda_i}{\lambda} \le \frac{1}{K} \quad \forall i$, and,

(ii) The number of replicas

$$n_i = \text{Max}(1, \text{Min}(\frac{\lambda_i \beta}{\gamma_0}, M)) \qquad (9)$$

where γ_0 is s.t. $\sum_{i=1}^{N} n_i = KM$, in the general case.

Proof:

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 (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

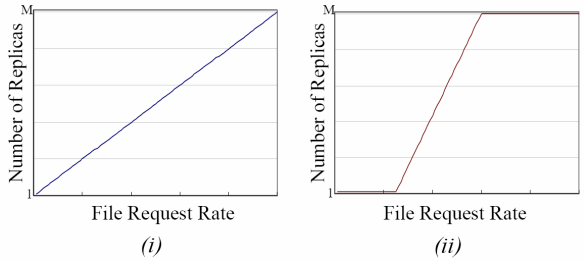

Figure 3. Optimal Replica Distribution: (i) under constraints on file request rates in (8) (ii) in the general case.

optimization problem is:

$$H = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \tau_i(N_i) + \gamma\left(\sum_{i=1}^{N} n_i - KM\right)$$

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 $\tau_i(n_i) = \alpha \log_d(M/n_i) + C = -\beta \ln(n_i) + c'$,

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

Substituting this in (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 (6) and (7) are 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 (6) and (7) and rewriting the $n_i \ge 1$ constraint as $-n_i \le -1$, the modified Lagrangian is:

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

The Kuhn Tucker Conditions for the modified Lagrangian are:

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

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

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

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

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

(13b) and (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:

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

where, from (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 the remaining discussion, we assume $\frac{1}{KM} \le \frac{\lambda_i}{\lambda} \le \frac{1}{K} \forall i$ and use (11) as the optimal replica distribution. For sufficiently large KM, $\frac{\lambda_i}{\lambda} \ge \frac{1}{KM}$ 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 download 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.

C. *Uniform Distribution of Replicas*

The assumption of uniform distribution of file replicas in the network allows us to say that the probability of finding file i at a randomly selected node is n_i/M when there are n_i replicas of the file in the network. In this section, we wish to verify that in our model of a peer-to-peer system where nodes have fixed storage (and, consequently, must use some cache management algorithm), the replicas are indeed uniformly distributed at "equilibrium" when all peers have the same file access pattern. Let us clarify what we mean by "equilibrium". Suppose that each file is at one node (the "origin server" for that file) initially. As nodes make requests and get these files, the number of replicas for each file increases. However, since the storage is limited, nodes will eventually have to delete some of these replicas. Thus, the number of replicas changes from the initial condition of one replica per file over time. We define the system to be in equilibrium when the number of replicas of each file does not change (on average, over some time interval), i.e., the "drift" is zero.

A generic un-annotated network topology has no physical location associations which makes discussing the distribution of replicas in the system difficult. Therefore, we introduce a coordinate system in our network. Relative to any one particular node (the "origin"-node), one can think of the location of the other nodes as their hop distance to the origin-node. Thus, relative to the origin-node, the node location can be specified using polar coordinates. A radially different pattern is not possible if the file request patterns and the per-node storage capacity are uniform across all nodes, and the topological difference among the nodes is minimal. Therefore, the ratio of the number of replicas of a file to the number of nodes at each hop distance from the origin-node is an adequate measure of file distribution. For example, if this ratio is the same at each hop-distance, one can conclude that the file replicas are uniformly distributed in the network. In contrast, if

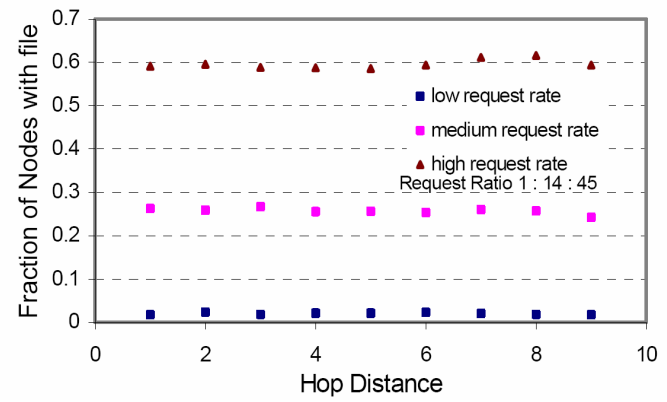

Figure 4. Distribution of files at different hop distances from the Origin Server for files with different request rates

this ratio were to decrease with increasing hop distance when the origin server is chosen to be the origin-node, one would conclude that the probability of finding the file is higher at nodes closer to the origin server. For any simulation run, after the topology instance is generated, the number of nodes located at each hop distance from the origin-node is easily computed. Statistics for the number of nodes that have a file can be collected after the simulation has run for a sufficient length of time (reached "equilibrium").

Fig. 4 shows the file distribution, with a single origin server, for three different files – one with a very high request rate, another with a moderate request rate and the third with a very low request rate. The simulation had 1000 peers with a storage capacity to store 10 files each and there were a total of 100 unique files in the system. The file request rates were Zipf-distributed with a zipf-exponent of 1.0 which has been found to be adequate to capture the skew in request rates for peer-to-peer systems [7] and web environments [2]. The underlying link-level topology was the 10,254-node PLRG discussed earlier. The participating peers were uniformly chosen from the link-level topology nodes. The file replacement policy when the space was needed for a newly requested file was LRU (i.e. the Least Recently Used file is replaced).

For the simulation results shown in Fig. 4, the number of replicas used to compute the fraction of nodes that have the file at each hop distance is the average value over 1000 simulation iterations after the steady-state replica distribution was reached. The deviation from the overall fraction of nodes with the file, at different hop distances, is a measure of how uneven the file distribution is. As seen in the figure, relative to the overall fraction of nodes with the file, the variation in the fraction of nodes with the file at different hop distances is very small. Therefore, we conclude that when all the participating peers have the same file request rates, at equilibrium, the replicas of a file are uniformly distributed over the network.

IV. Cache Replacement Algorithms For Proportional Replica Distribution

Ideally, we wish our peer-to-peer system to be autonomic and operate at optimal or near-optimal performance with no external intervention (as opposed to measuring file request

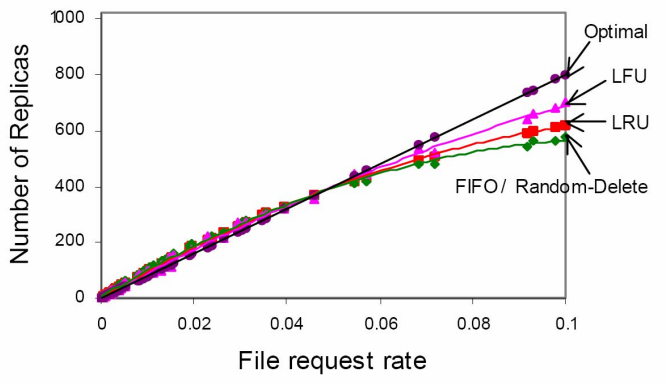

Figure 5. Number of Replicas vs. Request Rate

rates periodically and then populating the network with a proportional number of replicas by a centralized mechanism). In Section 3.C, we discussed a peer-to-peer system where the peers had finite storage space and if space was needed for a newly requested file then a previously obtained file was deleted using the LRU file replacement policy (except that the last replica of a file is never deleted). We plot the steady-state distribution of the number of replicas of each file against the file request rate in Fig. 5 for the same simulation. As we can see, the LRU cache replacement policy obtains near linear proportionality except for high request rate files.

We also simulated other common cache management algorithms such as FIFO (First-In, First-Out: replace the oldest file), LFU (Least Frequently Used: replace the least frequently used file) and Random-Delete (randomly select the file to be replaced) and the results are shown in Fig. 5 alongside the results for LRU. All these algorithms generate a replica distribution similar to LRU: LFU is closer to the optimal distribution than LRU while FIFO and Random-Delete are slightly further from it than LRU.

While none of the known cache management algorithms give us exactly the desired linear proportionality, an equilibrium analysis of Random Delete (when space needs to be made in the cache, one of the files is randomly selected for deletion) provided us insights into possible ways for constructing mechanisms to achieve the linear proportionality in a decentralized manner.

Each file request when the file is not available in the local cache results in that file being 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:

$$\lambda_i(1-n_i/M) = Cn_i$$

where C is a proportionality constant. This explains why Random Delete does not give linear proportionality.

A keen observer will note that a replica creation rate of λ_i will achieve linear proportionality with Random Delete file replacements. The deviation from a λ_i creation rate was due to

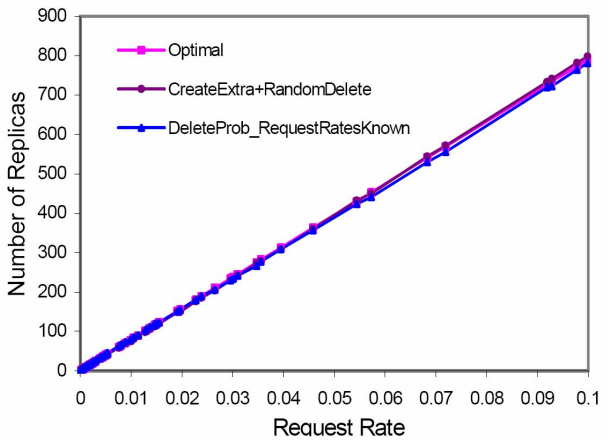

Figure 6. Optimal Cache Management Algorithms

replicas not being created when the file is available in the local cache. Creating a replica when the requested file *is* available in local cache can rectify this. Since the replica cannot be created in the local cache, it must be created at a different peer. This *Create-Extra+Random-Delete* algorithm was simulated and the results are shown in Fig. 6. As expected, the populate-extra algorithm does give us the linear proportionality in number of replicas. Writing on other peers caches, however, defeats our original objective as the extra download incurred on every request that is locally satisfied is likely to negate any performance benefit derived from the replica distribution being closer to optimal. 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 Fig. 6. Unfortunately, in real systems, it is difficult to obtain the global file request rates locally so this algorithm does not seem very practical.

To evaluate the benefits (against the cost) of each of these algorithms, we should know the average number of links traversed per download for these algorithms. Using the $\tau_i(n_i) = \alpha \log_d(M/n_i) + C$ expression for the average number of links traversed in downloading a file that has n_i replicas, we can compute the average number of links traversed per download using (3) to be:

$$\tau = \alpha \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \log_d(M/n_i) + C' = \frac{\alpha}{\ln d} \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \ln\left(\frac{M}{n_i}\right) + C'$$

To remove the effects of the underlying network topology, we extract the replica-distribution-dependent term from τ and define a new system performance metric, *normalized network bandwidth used*:

$$\tau' = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \ln\left(\frac{M}{n_i}\right) \qquad (14)$$

TABLE I. NORMALIZED NETWORK BANDWIDTH USED VS. CACHE REPLACEMENT ALGORITHM

Algorithm	LRU	LFU	FIFO	"Optimal"
τ'	1.1897	1.176	1.203	1.1746

We compute the normalized network bandwidth used from the steady-state replica distribution achieved by the different cache replacement algorithms in these simulations. The numbers are listed in Table 1. Comparing the performance of the different algorithms, we find that no significant advantage is obtained by using LFU (especially taking into account the added complexity of keeping frequency counters) instead of LRU or FIFO and that even the optimal cache replacement algorithm offers no significant advantage. Thus, LRU or FIFO cache replacement is a strong candidate for use in our peer-to-peer system and we study LRU's performance in peer-to-peer environment in more detail in the next section.

V. LRU CACHE REPLACEMENT IN PEER-TO-PEER ENVIRONMENT

So far, our simulations used a zipf-distributed file request rate with zipf-exponent of 1.0. To explore the effect of skew in file request rates, we show the simulation results for the number of replicas against the file request rate in Fig. 7 for two other file request rate distributions: a more skewed file request rate distribution (zipf-distribution with zipf-exponent 1.5), and a less-skewed request rate distribution where the request rates for different files are uniformly distributed between the lowest and the highest file request rates.

These results suggest that for applications where the access patterns do not have much skew, LRU will perform very well. Notice that even when the skew in file request rates is so large that the optimal replica distribution is now defined by (9) instead of (8), LRU file replacement still achieves near optimal replica distribution.

An analytical model of a network with LRU storage can be

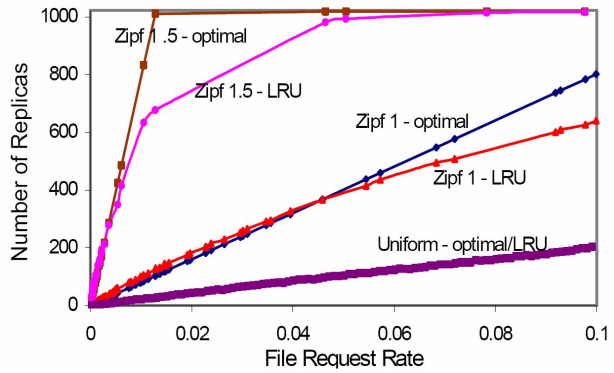

Figure 7. LRU performance with varying levels of skew in the request rate distribution

constructed as follows. Since the probability of finding a file is the same across the network, it is sufficient to find the probability of a file being in storage at any random node. Further, one can construct the model from the perspective of a particular file, say, file i – all requests for file i move the file to the top-most position in the storage; a request for any other files moves file i down to one lower position. We set up a Markov Chain to represent the position of the file in the stack, i.e. the system is in state k when the file is in the k^{th} position from the top of the LRU stack: state 1 implies that the file is at the top-most position in the LRU stack; state N (N is the storage capacity in number of files) means that the file is at the bottom-most position in the LRU stack. An additional state, state 0, is defined to represent the state when the file is not in the storage. A satisfied request for the file always changes the system state to 1 (as the file moves to top-most position in the storage). A satisfied request for any other file changes the system state to next higher state (as that other file now moves to the top-most position pushing all other files one position down). Satisfied requests for the file include the storage owner's requests for this file (since all file requests are eventually satisfied, this contributes to transitions to state 1 from all the other states) and the requests for the file from the other nodes that are satisfied (since nodes always share their files, this contributes to transitions to state 1 from all states other than state 0).

The model is shown in Fig. 8 where λ_i is the file request rate for file i, λ_i' is the total request rate including file i's requests from other nodes satisfied by this node and $\mu_{ij,j+1}$ is the rate at which file i is pushed down from position j to position j+1 as requests for other files are served by the node. While λ_i is given, the file requests from the other nodes (remote requests) complicate the expressions for λ_i'. A node sends out a file request only when it does not have the file. Thus, the rate at which the other M-1 nodes send a file request for this file to the peer-to-peer network is $\lambda_i p_{i0}$, where p_{i0} is the probability that the file i is not available at a node. The nodes that have file i in their cache satisfy these requests for file i sent to the peer-to-peer network. Assuming that the requests are uniformly distributed over the nodes that have the file, the request rate for file i served by a node that has file i on account of requests from other nodes equals $(M-1)\lambda_i p_{i0}/M(1-p_{i0})$. Unfortunately computing $\mu_{ij,j+1}$ is much harder for $j>1$. For $j=1$, the rate at which file i is pushed down is the total rate at which requests for all other files are served by this node (including both local and remote requests). Thus, $\mu_{i12} = \sum_{k\neq i}\lambda_k(1+p_{k0})$ where p_{k0} is the probability of not finding file k. However, for $j>1$, one must adjust these rates for the possibility that the requested file may be in a position $<j$ in which case, an access to a file $k\neq i$ will not affect position of file i in the LRU stack.

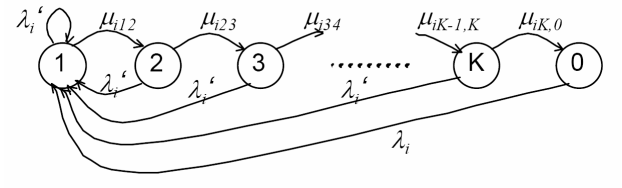

Figure 8. Markov Chain model for the position of file i when per-node storage capacity is K

Reference [5] suggests one technique to circumvent this complexity. The key idea is that at steady-state, the push-down rate for file i from position j to j+1 must equal the rate at which file i is brought into top j positions of the LRU stack (otherwise the probability of finding the file in these top j positions becomes unbounded). This conservation of flow principle allows us to compute $\mu_{ij,j+1}$. File i is brought into the top j positions under two conditions: (i) a local request for file i when file i is not in any of the top j positions: the file may be brought to the top position from positions j+1…K of the local cache if it is available there or it may be brought from a remote node (since we assume that a file that may be requested never disappears from the system, all file requests are satisfied) (ii) a remote request for file i: since the file i is not in any of the top j positions, it must be in the remaining j+1…K positions in the local cache for it to show up in one of the top j positions on a remote request. The local requests contribute $\lambda_i[1-P(i,1...j)]$ to the push-down rate where $P(i,1...j)$ is the probability that file i is in one of the top j positions in the local cache. The remote requests contribute an additional $\lambda_i[(M-1)p_{i0}/M(1-p_{i0})]$ $P(i,j+1...K|j)$ where $P(i,j+1...K|j)$ is the probability that file i is in positions j+1…K of the local cache given that it is not in any of the top j positions in the local cache. Thus,

$$\mu_{ij,j+1} = \lambda_i[1-P(i,1...j)] + \lambda_i[(M-1)\,p_{i0}/M(1-p_{i0})]P(i,j+1...K|j) \quad (15)$$

Even after obtaining all the required rates for the Markov Chain model, calculating individual probabilities is very involved. [5] provides an approximate expression for p_{ij}, the probability that file i is at position j in the LRU stack in terms of the push-down rates at position $j-1$ as follows:

$$p_{ij} \approx \frac{\mu_{ij-1,j}}{\sum_{k=1}^{N}\mu_{kj-1,j}} \quad (16)$$

Other probabilities are defined in terms of p_{ij} as:

$$P(i,1...j) \approx \sum_{k=1}^{j} p_{ik} \quad (17)$$

$$P(i,j+1...K|j) = \frac{P(i,1...K)-P(i,1...j)}{1-P(i,1...j)} \quad (18)$$

$$p_{i0} = 1 - P(i,1...K) \quad (19)$$

Starting with $P(i,1) = \lambda_i$, we can iteratively solve (15)-(19) until the value of p_{i0} converges. The complexity is O(KN) [5] and, in our computations, the value of p_{i0} converged in only a few iterations. In Fig. 9, we plot the number of replicas of each file obtained from simulation and $MP(i,1...K)$, the product of the number of peers in the system and the probability of finding file i in the local cache, obtained from (15)-(19) against the file request rates for M = 5000, N = 500, K = 50 and zipf-distributed $\{\lambda_i\}$ with zipf-exponent 1.0. As shown in the figure, the analytical model agrees very well with the simulation

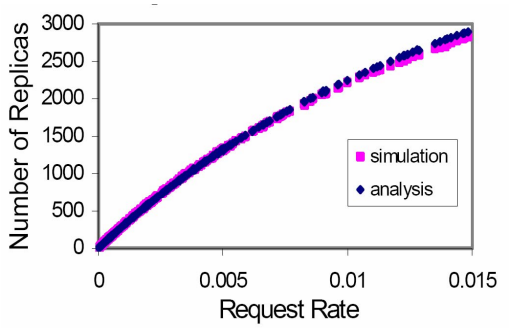

Figure 9. Validation of Steady-state Analytical Model for LRU

results. System designers can use this model to estimate the steady-state replica distribution for different system parameters and the predicted values can be used to estimate the required network bandwidth using (14) for a given cache size or for selecting the appropriate size of the per-node user cache to limit the required network bandwidth within a desired value.

VI. Transient Performance of the LRU Cache Replacement Policy

As we saw in earlier sections, at steady state, LRU and other popular cache management algorithms can achieve near-proportional replication even when the access rates are extremely skewed. All our simulations in Fig. 4 had started with only a single replica of each file (at their respective origin servers) and, the equilibrium distribution was eventually achieved in all the cases. Thus, we know that reasonable cache replacement policies will also adapt the file replication distribution if the user access patterns change. However, we must examine the transient performance of the system to assure ourselves that: either (a) the transient period is short, or (b) the performance during the transient period is acceptable (i.e. the peer-to-peer network does not crash for example). We conducted a number of simulation runs to study the transient performance with the different cache replacement algorithms. The basic simulation setup here was identical to that in Section 4 except that at a certain pre-determined simulation iteration, the request rate distribution was changed from zipf-distributed with exponent 1 to an exponent of 0 where each file now had the same request rates at all nodes. Fig. 10 shows the number of replicas of each file[1] with LRU cache replacement with increasing simulation iterations starting from a few iterations prior to the change. To get a sense of the scale of the x-axis, we note that in each simulation iteration, on average, each node makes one file request. In the zipf-exponent-1 request rate distribution, the lowest file request rate is 0.0001 requests/node/iteration. Thus, on average, in 10 iterations, the least popular file is requested only once.

From Fig. 10, we note that it takes about 12-13 iterations to reach the new steady-state in the presented simulation scenario. For zipf-exponent-0 request rate distribution, on average, there are 10 requests for each file per iteration. The file with the lowest request rate in the original distribution had 1 replica at

[1] In Figs. 10, 11 and 13, each curve is for a different file (we show curves for 12 files out of the 100 files in the simulation) and shows the number of replicas of the file in the iteration indicated on the x-axis.

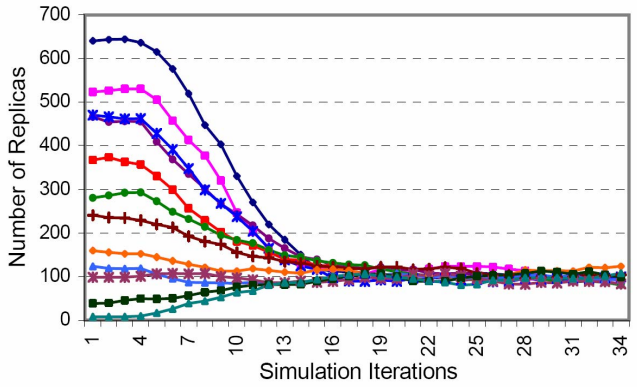

Figure 10. Transient behavior of LRU (the request rate distribution changed at iteration label 5)

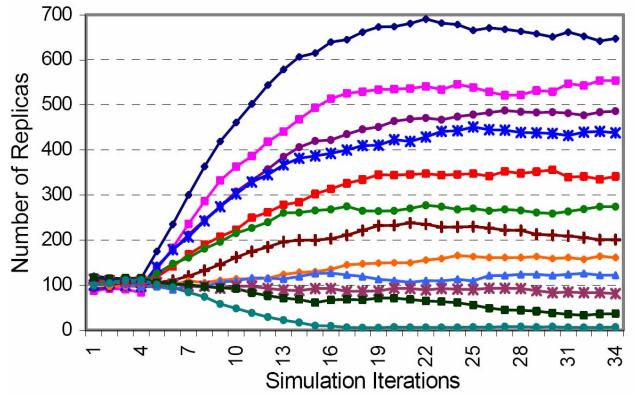

Figure 11. Transient behavior of LRU (the request rate distribution changed at iteration label 5)

iteration labeled 5 and according to the zipf-exponent-0 request rate distribution there should be about 100 replicas in the system. Since new replicas are created only upon requests for the file, it will take at least 10 iterations to create 100 replicas. Thus, the 12-13 iterations LRU took to reach the new equilibrium distribution is very reasonable.

In the aforementioned simulation, we also changed the request rate distribution back to the original zipf-exponent-1 distribution 400 simulation iterations after the change to the zipf-exponent-0 request rate distribution. In Fig. 11, we show the number of replicas of each file with increasing simulation iterations starting a few iterations prior to when the request rates revert back to the original distribution.

Once again, we see that the number of replicas reaches the neighborhood of the steady-state values in about 12-13 iterations after which we can say that the system performance is close to the steady-state performance. We also note here that the rate of convergence to the new steady-state distribution appears to be independent to the individual request rates in both Fig. 10 and 11.

Similar experiments were performed for FIFO, LFU and the "optimal" cache replacement algorithm discussed in Section 4. Instead of presenting the per-file details, as in Figs. 10, 11, we condense the replica distribution information by computing the normalized network bandwidth used, as defined in Section 4, for the replica distribution at each iteration and show only that

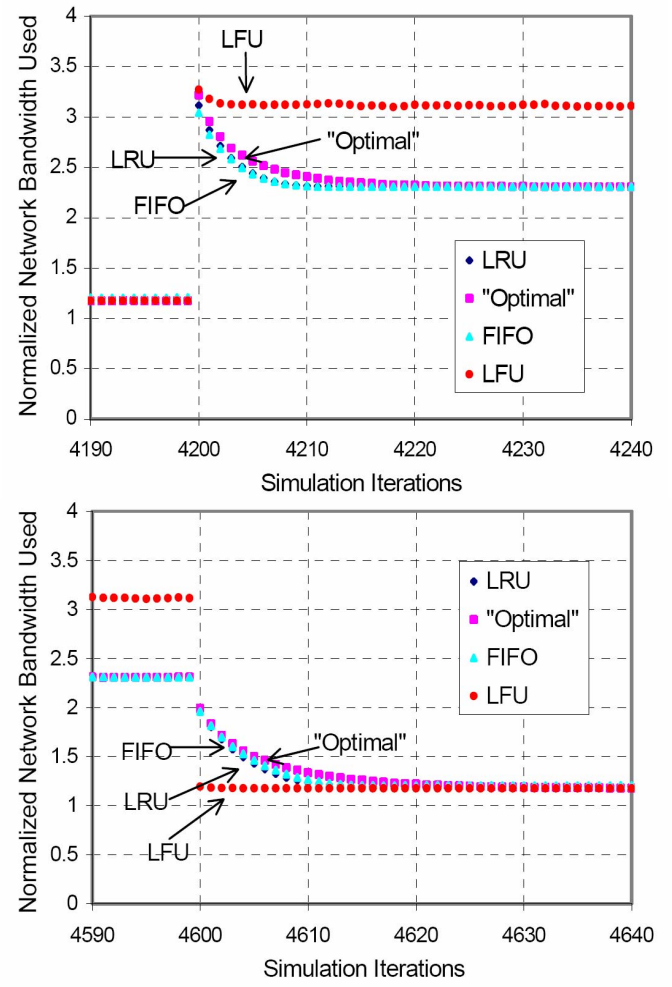

Figure 12. Transient Performance of different cache replacement algorithms

in Fig. 12. At the 4200th iteration, the request rate distribution was changed from zipf-exponent-1 to zipf-exponent-0 and at the 4600th iteration it was changed back to zipf-exponent-0. As shown in the figure, the duration of the transient period with FIFO is the same as that with LRU while the optimal cache replacement algorithm constructed in Section 4 has a slightly longer transient period and a slightly worse performance during the transient period (even though the simulations assumed that the new file request rate distribution is relayed to the algorithm at all nodes instantaneously).

As we can see LFU adapts to the change in request rate very poorly (it has better transient performance than other algorithms upon reverting back to the original request rate distribution only because it had never achieved the steady-state distribution for the equal request rate distribution and when the request rate distribution reverted back to zipf-exponent-1 distribution, the replica distribution was close to the steady-state distribution of LFU for the zipf-exponent-1 distribution). This happened because our simulations used an infinite length window over which the frequency is counted (with an upper bound on the maximum counter value) and so it takes a long time for the algorithm to register the change in access patterns (effectively, the replica distribution never changed). A shorter window should perform better than shown (but not necessarily better than LRU) but maintaining a sliding window is a complex task.

These results suggest that LRU or FIFO may be an adequate choice for cache replacement in peer-to-peer systems. To better understand the transient behavior of LRU, we now attempt to develop an analytical model for the transient behavior of LRU cache replacement.

Following the approach in [3], which analyzed the transient performance of LRU for a database application, we first derive the expressions for LRU performance in the cache warm-up period (i.e. starting with an empty cache to reaching the steady-state replica distribution). As we saw in the previous section, the number of replicas of each file at steady state can be computed as $MP(i,1...K)$ where M is the number of peers in the network and $P(i,1...K)$ is the probability of finding file i in the cache. We wish to compute the time-dependent probability of finding file i in the cache after T accesses, $P^t(i,T)$, starting with an empty cache. Note that once there are enough accesses that LRU replacement policy kicks in, the desired probabilities are defined by (15)-(19) as derived in Section 5. In this section, we are only interested in computing the probabilities while the cache is not full.

We can compute $P^t(i,T)$ if we know the probability that file i is not in the cache after T accesses, $p^t_{i0}(T)$, using (19). After T accesses, file i is not in the cache only if none of the previous T accesses were for file i. Therefore,

$$p^t_{i0}(T) = \left(1 - \frac{\lambda_i}{\lambda}\right)^T \tag{20}$$

where λ_i is the request rate for file i. Note that if the file is not in the cache, the additional term for requests for file i from other nodes satisfied by this node that complicated (15) is not required in the warm-up transient analysis.

Given $p^t_{i0}(T)$, the probability of finding file i in the cache after T accesses is:

$$P^t(i,T) = 1 - \left(1 - \frac{\lambda_i}{\lambda}\right)^T \tag{21}$$

These expressions apply only if the cache is not full yet i.e.

$$\sum_{i=1}^{N} P^t(i,T) \le K \tag{22}$$

One can iteratively compute (21), (22) for increasing values of T beginning with T=1 until (22) is violated. The smallest value of T at which (22) is violated is the transient period, $T_{transient}$. The values of $P^t(i,T)$ computed at each iteration can be used to compute the number of replicas of each file in the transient period which can, then, be used to estimate the system performance as defined in Section 4 for the transient period.

We compare the replica distribution during the starting period of our simulation with LRU cache replacement to the output of our analytical model for the same system parameters (M = 1000, N = 100, K = 10 and zipf-distributed $\{\lambda_i\}$ with zipf-exponent 1.0) in Fig. 13. As we can see in Fig. 13, the transient performance predicted by the analysis matches very well with

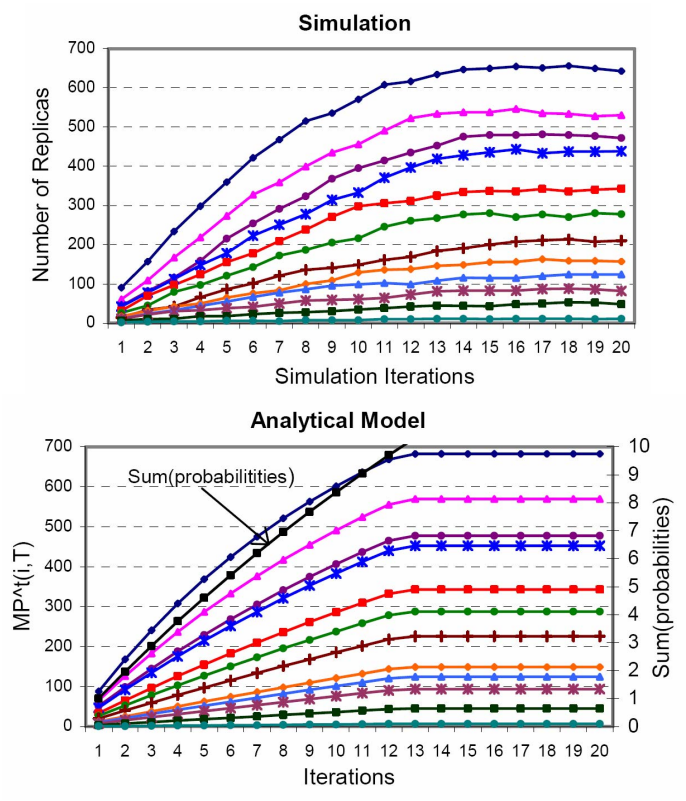

Figure 13. Validation of Analytical Warm-up Transient Model

the simulation results for the cache warm-up period. The analytical model shows (22) being violated at the 13th iteration which is the duration of the warm-up transient period in the simulation as well. $MP^t(i,T)$ also appears to match well with the number of replicas of each file in the warm-up transient duration. Beyond the warm-up transient period, we plot the replica distribution based on the steady-state probabilities as given in Section 5.

We also note that the values of $P^t(i, T_{transient})$ obtained via this iterative process (shown in Fig. 13) are close to the steady-state probabilities $P(i,1...K)$ that we computed in the previous section using (15)-(19) (the differences arise as we ignored the additional term for requests for file i from other nodes satisfied by this node). Thus, this warm-up transient model gives us another method of estimating the steady-state system performance.

Another key observation is that the warm-up transient period of 13 iterations is the same as the transient period in our simulations shown in Fig. 10, 11. Thus, the general transient period is related to the warm-up transient period. We know that if the request pattern changed such that none of the older files were to be requested and a new set of files becomes the active set, our arguments in development of (21), (22) will apply exactly the same. In the general case where the file request rate changes are not as drastic, the same calculations for $P^t(i,T)$ as in (21) may not apply although the duration of the transient period is likely to be shorter or the same as that defined by (21), (22) for which the cache is completely populated according to the new access pattern.

Finally, we add that (22) directly demonstrates the role of cache size in determining the duration of the transient period. If we increase the cache size K to 20 files in our earlier system, our computations show that the duration of the warm-up transient period increases to 35 iterations (compared to 13 iterations for K=10). Intuitively, one can see this since a larger cache means more entries must clear out before the distribution is defined by the new access pattern. When the request rate changes are not as drastic, however, we do expect the transient period to be shorter.

Also, recall from Section 4 that the system performance is not very sensitive to the replica distribution when the replica distribution is close to the optimal distribution. So, for small changes, the system performance should remain reasonable even through the transient period.

VII. Related Work

Aside from dealing with specific issues like free-riding, short node lifetimes etc. associated with music/video file sharing over the Internet, the most popular application of peer-to-peer networking, a substantial part of research into peer-to-peer systems has gone into designing effective mechanisms for searching which peer has the desired file which is a difficult task if one attempts to build an ideal peer-to-peer system where all participants are equal and there are no central servers. Some of our analytical work on search performance in unstructured peer-to-peer networks [16, 17] is relevant to our work here since the average distance to the nearest replica is also the number of hops needed to find a source for the desired file in flooding-based searches. In particular, our derivation of the average distance to the nearest replica over the link-level topology in Section 3 has the same steps as the derivation of the average distance to the nearest replica over the Erdos-Renyi random graph overlay network in [17]. Download performance in peer-to-peer networks has also been addressed by [1, 11, 18] among others. [11] discusses the gains in reducing the download time by splitting a large file into small pieces so as to increase the service capacity of a large file rapidly after its initial introduction into the peer-to-peer network. [1] provides an analytical model for selecting peers so as to minimize the download time while [18], like our current work, focuses on file replication in seeking the same goal. File replication is also discussed by [4, 8, 12] among others. These works are similar to our work in that one of their objectives in replication is improving the download performance. However, [8, 12] study system architecture issues. [4] presents an analytical model for a decentralized caching system but since it is in the context of web caching, the assumptions are different (e.g. unlike our model, cached content has limited lifetime in their scenario). Web caching is addressed by many others (e.g. [14, 20]) and, even though we study a network of caches, some of this work is relevant to ours as our assumptions of uniformity imply that the system performance can be inferred from the behavior of a single cache and, hence, we find similarities in the analyses [14] and conclusions [20]. A content distribution network also replicates content at multiple sites (to decrease access latency seen by end-users) and optimal allocation of system storage is an issue in these networks also (e.g. [9]). While such works may incorporate caching/replacement in their investigation, the

overall model is of centralized control over the multiple sites and issues such as replica placement have been the main focus of the research in this area. Peer-to-peer networking is also being proposed now to support web accesses in cooperative mobile environments and [10] presents an analytical model for the performance of such a system. Our analytical model in Section 5 is very similar to theirs as we both extend the analytical model for a stand-alone LRU cache given in [5]. Finally, [13], also discusses the "natural" scaling achieved by the fact that user requests create additional replicas which improves system performance although they focused on the system's ability to find the newly created sources. The "scaling" effect observed in [13] regarding the search problem in peer-to-peer networks is, in fact, on account of the optimality of proportional replication for search in unstructured peer-to-peer networks which we had shown in [16] for the case of uniform distribution of files as assumed in this paper. For the search problem in unstructured peer-to-peer networks, we have shown the optimality of proportional replication even for the case of clustered demands in [19].

VIII. Conclusion

In this paper, we showed that the average network bandwidth used per download is minimized when the number of replicas of a file in the network is proportional to the request rate for the file i.e. $n_i \propto \lambda_i \; \forall i$, where n_i is the number of replicas of file i, and λ_i is the request rate for file i. This result on network-wide benefits of this proportional replication which complements our earlier results on the per-node advantages of this replica distribution motivated our inquiry into the ability of cache replacement algorithms to automatically achieve the proportional replication. We found that cache replacement algorithms such as LRU are able to achieve near-proportional distribution. Our simulation results indicate that the average network bandwidth used per download with the replica distribution achieved by LRU is very close to the performance achieved with the optimal replica distribution in the cases we simulated. An analytical model was developed for computing the steady-state replica distribution with LRU in the general case. Since the user access patterns may change over time, we also investigated the transient performance of some of the cache replacement algorithms. The time taken by LRU to converge to the new steady-state replica distribution after a change in the user access pattern was found to be very close to the minimum required by any cache replacement algorithm. We also developed an analytical model for the transient behavior of LRU. In conclusion, LRU cache replacement algorithm is a very attractive mechanism for obtaining the network-wide benefits offered by proportional replication in peer-to-peer networks.

References

[1] M. Adler, R. Kumar, K. Ross, D. Rubenstein, T.Suel and D. Yao, "Optimal Peer Selection for P2P Downloading and Streaming," In Proc. of IEEE INFOCOM 2005.

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

[3] A. Bhide, A. Dan, D. M. Dias, "A Simple Analysis of the LRU Buffer Policy and Its Relationship to Buffer Warm-Up Transient," In Proc. of ICDE 1993.

[4] F. Clévenot and P. Nain, "A Simple Model for the Analysis of the Squirrel Peer-to-peer Caching System," In Proc. of IEEE INFOCOM 2004.

[5] A. Dan and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes," In Proc. of ACM SIGMETRICS 1990.

[6] C. Faloutsos, M. Faloutsos, P. Faloutsos, "On power-law relationships of the internet topology," In Proc. of ACM SIGCOMM 1999.

[7] K. P. Gummadi, et al., "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," In Proc. of ACM SOSP 2003.

[8] J. Kubiatowicz et. al., "OceanStore: An Architecture for Global-scale Persistent Storage," In Proc. of ASPLOS 2000.

[9] N. Laoutaris, V. Zissimopoulos, I. Stavrakakis, "On the Optimization of Storage Capacity Allocation for Content Distribution," Computer Networks, Vol. 47, No. 3, pp. 409-428, February 2005.

[10] C. Lindemann and O. P. Waldhorst, "Modeling Epidemic Information Dissemination on Mobile Devices with Finite Buffers," In Proc. of ACM SIGMETRICS 2005.

[11] L. Massoulie, M. Vojnovic, "Coupon Replication Systems," In Proc. of ACM SIGMETRICS 2005.

[12] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", In Proc. of SOSP 2001.

[13] D. Rubenstein and S. Sahu, "Can Unstructured P2P Protocols Survive Flash Crowds?," IEEE/ACM Trans. on Networking, Vol. 13, No. 3, pp. 501-512, April 2005.

[14] D. Starobinski and D. Tse, "Probabilistic Methods for Web Caching," Performance Evaluation, Vol 46, Nos. 2-3, October 2001.

[15] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topology generators: Degree-based vs structural," In Proc. of ACM SIGCOMM 2002.

[16] S. Tewari, and L. Kleinrock, "Analysis of Search and Replication in Unstructured Peer-to-Peer Networks," In Proc. of ACM SIGMETRICS 2005.

[17] S. Tewari, and L. Kleinrock, "Search and Replication in Unstructured Peer-to-Peer Networks," UCLA Computer Science Dept Technical Report UCLA-CSD-TR050006, March 2005.

[18] S. Tewari, and L. Kleinrock, "On Fairness, Optimal Download Performance and Proportional Replication in Peer-to-Peer Networks," In Proc. of IFIP Networking 2005.

[19] S. Tewari, and L. Kleinrock, "Optimal Search Performance in Unstructured Peer-to-Peer Networks With Clustered Demands," to appear in Proc. of ICC 2006.

[20] L. Rizzo and L.Vicisano, "Replacement policies for a proxy cache," UCL-CS Research Note RN/98/13.