Optimizing File Availability in Peer-to-Peer Content Distribution

Jussi Kangasharju* Keith W. Ross† David A. Turner‡

*Dept. of Computer Science
TU Darmstadt
Darmstadt, Germany

†Dept. of Computer and Information Science
Polytechnic University
Brooklyn, NY

‡ Dept. of Computer Science
California State University
San Bernardino, CA

***Abstract*— A fundamental paradigm in peer-to-peer (P2P) content distribution is that of a large community of intermittently-connected nodes that cooperate to share files. Because nodes are intermittently connected, the P2P community must replicate and replace files as a function of their popularity to achieve satisfactory performance. In this paper, we develop an analytical optimization theory for benchmarking the performance of replication/replacement algorithms, including algorithms that employ erasure codes. We also consider a content management algorithm, the Top-K Most Frequently Requested algorithm, and show that in most cases this algorithm converges to an optimal replica profile. Finally, we present two approaches for achieving an evenly balanced load over all the peers in the community.**

I. Introduction

In this paper we consider the problem of optimizing file availability in peer-to-peer content distribution communities. Because some files are more popular than others, the question of how to manage the content in order to obtain the optimal performance is important. The essence of the problem is deciding how many copies of each file should be created in the community in order to optimize the given performance metric. As discussed in detail in Section II, the typical goal of a content distribution community is serving as many files as possible from within the community, i.e., maximizing the intra-community hit-rate.

The principal contribution of this paper is deriving an optimal replication policy for content distribution communities. We develop the theory for complete-file replication as well as for segmented-file replication in which erasure codes are used to provide redundancy. For the general case with erasures, we derive an upper bound on the performance of all adaptive schemes. For complete-file replication, we show that an explicit *logarithmic assignment rule* is optimal. The optimization theory provides significant insight into how P2P content distribution communities should be managed.

The second contribution of the paper is an analytical performance evaluationof an adaptive, on-line replication algorithm, called Top-K Most FrequentlyRequested. We show that this algorithm (in most cases) converges to an optimal replication.

Finally, the third contribution of this paper is an evaluation of load balancing algorithms, which shows that even a simple fragmentation approach, where files are divided into small fragments, is sufficient for obtaining an evenly balanced load. This can further be tuned by allowing individual peers to refuse requests they cannot handle; our results show that these refusals have only minimal effect on the other peers.

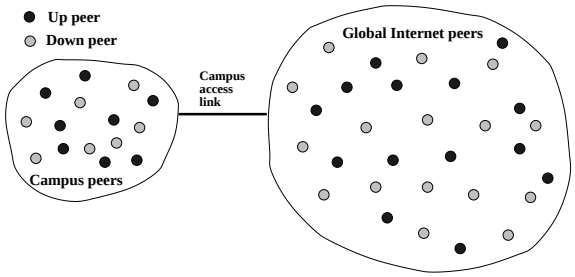
This paper is organized as follows. Section II introduces the concept of community-based content distribution. In Section III, we develop the analytical theory for optimal replication in P2P, including a theory for replication with erasures. Section IV discusses the Top-K Most Frequently Requested algorithm and shows that it almost always achieves optimal performance. We consider load balancing in Section V. Section VI reviews related work. In Section VII we conclude and discuss future directions.

II. Community-Based Content Distribution

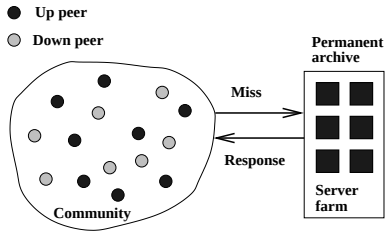
In this paper we examine *content distribution communities*. A P2P content distribution community is a collection of intermittently-connected nodes with each node contributing storage, content and bandwidth to the rest of the community. When a node in the community wants a particular file, it first attempts to retrieve the file from the other nodes in the community. If the desired file is not found in the community, the community retrieves the file from the outside, possibly caches the file, and forwards a copy to the requesting node.

As an example of a P2P content distribution community, consider a university campus network or an ISP network. As shown in Figure 1(a), the peers in a campus are typically interconnected with a high-speed LAN, and the high-speed LAN is connected to the global Internet via a lower-speed access link. Currently, the peers within a campus do not organize themselves as a P2P community: the peers in the campus independently retrieve the same content from peers outside the campus, clogging the access links and wasting peer storage. The university campus could make more efficient use of its resources (WAN bandwidth and peer storage) if the peers were organized in a P2P community. As a P2P community, the peers in the campus would collectively maintain a managed number of copies of files, and would attempt to retrieve files internally before retrieving them from outside the campus.

Note that the main goal of creating a community is to reduce traffic on the network links connecting the community to the outside. Even home users in one ISP could form a community, since that would reduce traffic on the ISP's outgoing link. Although the connection to another home user in the same ISP might not be significantly faster than a connection to any other peer in the Internet, reducing traffic on the outgoing link will



(a) Campus or ISP community



(b) Content distribution booster

Fig. 1. Different types of P2P communities

speed up the downloads which must be retrieved from outside of the community. Hence, the users obtain a better quality of service than they would without the community.

A second example of a P2P content distribution community is a *content-distribution booster*, for example, for the distribution of video training content in a large corporation. As shown in Figure 1(b), videos are permanently archived in a small number of servers, which collectively do not have enough aggregate server and/or transmission capacity to serve all the users in the corporation. By organizing all the corporate nodes into a P2P community, the P2P community serves as a front-end surrogate for video distribution. Popular videos would often be downloaded (or streamed) from other corporate peers, thereby relieving the burden on the archival servers.

In this paper we address the problem of managing content in the community such that the file availability is maximized. We make the assumption that intra-community file transfers are considerably faster than retrieving a file from the outside. This is clearly the case for communities based campus networks and content boosters. Under this assumption, maximizing file availability likely also minimizes the average file download delay, which is the time it takes between the request for a file and the completion of the download. Because the file transfer delay is typically orders of magnitude larger than lookup delays, and because intra-community file transfers occur at relatively fast rates, our problem is, for all practical purposes, equivalent to adaptively managing content to maximize intra-community hit rates.

For ISP-based communities, the assumption about fast intra-community transfers might not always hold, hence the assumption about a high hit-rate minimizing the download time might not be valid. However, the main goal in an ISP community would typically be to minimize the traffic on the outgoing link, i.e., maximize the byte hit-rate within the community. (Wanting to minimize external traffic might be also true for some other communities.) In this paper we consider only maximizing the hit-rate within the community, but all of our theoretical results and algorithms can easily be extended to handle the case where maximizing the byte hit-rate is the goal.

There are three important issues in maximizing the intra-community hit rate.

- **Replication:** Because nodes connect to and disconnect from the community, content needs to be replicated in the community to provide satisfactory hit rates. Naturally, popular content needs more replicas than unpopular content. At the same time, content should not be excessively replicated, wasting bandwidth and storage resources.
- **File Replacement Policies:** Each participating node has a limited amount of storage that it can offer to the community. When this storage fills at some node, the node needs to determine which files it should keep and which it should evict.
- **Load Balancing:** Because some files are more popular, peers storing these files may experience much higher loads than peers storing unpopular files. Our content management algorithms should take this into account and attempt to balance the load as evenly as possible.

III. Optimization Theory

We now develop an analytical theory for optimal content replication in P2P communities. The theory applies to both open and closed communities, where an open community offers the possibility to access files from outside the community. We also derive the theory in a more general context for which each file is erasure coded. Our main focus is on obtaining upper bounds on the performance of replication algorithms.

For this analytical theory, we assume that file popularities and node up probabilities are known *a priori*. Let I denote the number of nodes, and let p_i and S_i denote the up probability and shared storage for node i. In this section we suppose that nodes go up and down independently. Let J denote the number of distinct files, and let b_j and q_j denote the size and request probability of the jth file.

A. No Fragmentation

We first develop the theory for the case when files are not fragmented. Let x_{ij} be a zero-one variable which is equal to one if node i contains a replica of file j and is zero otherwise. It is straightforward to show that the hit probability is given by

$$P_{hit} = 1 - \sum_{j=1}^{J} q_j \prod_{i=1}^{I} (1 - p_i)^{x_{ij}}. \tag{1}$$

The assignments must satisfy the constraints

$$\sum_{j=1}^{J} b_j x_{ij} \leq S_i, \quad i = 1, \ldots, I \tag{2}$$

The solution of the integer programming problem of maximizing (1) subject to (2) provides an upper bound on the hit probability for all content management algorithms. The integer program can thus be used to benchmark algorithms. However, this optimization problem can be shown to be NP-complete by reducing it to the Zero-One Integer Programming problem [1].

We now consider a special case of this problem, namely, when each node is up with the same probability $p_i = p$. Let n_j denote the number of replicas for file j. For the case of homogeneous up probabilities, the problem is to choose non-negative integers $n_1, \ldots, n_J$ such that the following is maximized

$$1 - \sum_{j=1}^{J} q_j (1-p)^{n_j} \tag{3}$$

subject to

$$\sum_{j=1}^{J} b_j n_j \leq S \tag{4}$$

where $S = S_1 + \cdots + S_I$. Note that, because storage has been aggregated into one constraint, the optimal value for (3-4) is actually an upper bound on the optimal hit probability (for homogeneous values of p_i); however, this upper bound turns out to be very tight, and can actually shown to be exact for important special cases. (For example, when all files are the same size.) This optimization problem can be solved efficiently by dynamic programming. Indeed, let $f_j(s)$ be the minimum miss probability when there are s bytes of aggregate storage and files $j, \ldots, J$ to replicate. Then standard dynamic programming arguments give

$$f_j(s) = \min_{n: b_j n \leq s} [f_{j+1}(s - n b_j) + q_j (1-p)^n]$$

The optimal replica profile $n_1, \ldots, n_J$ solves $f_1(S)$.

We note in passing that the same methodology can be used to obtain an upper bound when there are two heterogeneous up probabilities, one for each of the two sets in a two-set partition of the nodes. (For example, 20% of the nodes up with probability .9 and 80% of the nodes up with probability .2.) In this case, we can derive a two-dimensional dynamic programming equation. The computational complexity increases, but the computations remain tractable for large values of J.

B. Upper Bound With and Without Erasures

The number of copies of any of object in the P2P community is an integer at any given time. By removing this integrality restriction, we can develop a methodology for efficiently determining an upper bound on the performance of adaptive management algorithms in large P2P systems. We shall do this for the case of erasures; the case without erasures will be treated as a special case.

We now suppose that each file j is made up of R_j erasures, and that any M_j of the R_j erasures are needed to reconstruct the file. The size of each erasure is b_j/M_j. Throughout this analysis we assume homogeneous up probabilities, that is, $p_i = p$ for all nodes. We also make the natural restriction that the replication algorithms that we are bounding are such that no two erasures from the same file are stored on the same node. Let $c_j = M_j/(b_j R_j)$ and

$$f_j(z) = q_j \sum_{m=M_j}^{R_j} \left(\begin{array}{c} R_j \\ m \end{array} \right) [1-(1-p)^{c_j z}]^m [(1-p)^{c_j z}]^{R_j - m}$$

The main result of this subsection is the following:

Theorem 1: The maximum value of the following optimization problem provides an upper bound on the hit probability for a P2P file-sharing community with erasures.

Maximize

$$\sum_{j=1}^{J} f_j(z_j) \tag{5}$$

subject to

$$\sum_{j=1}^{J} z_j = S \tag{6}$$

$$z_j \geq 0, \quad j = 1, \ldots, J \tag{7}$$

The optimization problem in Theorem 1 is easy to solve numerically, even for large values of J and R_j, $j = 1, \ldots, J$. Specifically, it is straightforward to show that $f_j(z)$ is an increasing concave function of z, so that optimization problem is a separable concave allocation problem. Let $(z_1^*, \ldots, z_J^*)$ be the optimal solution for this concave optimization problem. From Kuhn-Tucker theory, there exists an α such that

$$f_j'(z_j^*) = \alpha \text{ for all } j \text{ such that } z_j^* > 0 \tag{8}$$

The standard procedure to solve this type of problem is to first pick an $\alpha > 0$, solve $f_j'(z_j) = \alpha$ for all j; for those values of j such that $z_j \geq 0$, we sum the z_j's and check if the sum is above or below S. If the sum is above S (below S) we decrease α (increase α) and repeat the procedure. Using a binary search to adjust α, we iterate until the sum of the positive z_j's is within ϵ of S. For the positive z_j's we set $z_j^* = z_j$; for the remaining z_j's we set $z_j^* = 0$.

Theorem 1 provides a powerful means to benchmark the performance of adaptive algorithms with and without erasures. We provide an example at the end of this section that shows that the upper bound is quite tight when entire files are replicated.

Proof: We refer to the rth erasure of file j as erasure jr, $r = 1, \ldots, R_j$. For a fixed assignment of erasure replicas to nodes, let n_{jr} be the number of erasures jr stored in the community of nodes. Clearly

$$\sum_{j=1}^{J} \sum_{r=1}^{R_j} \frac{b_j}{M_j} n_{jr} \leq S \tag{9}$$

Using the same fixed assignment, let Φ_{jr} be the 0-1 random variable which is 1 if any of the n_{jr} erasures jr is in some up node. Clearly,

$$P(\Phi_{jr} = 1) = 1 - (1-p)^{n_{jr}} := p_{jr} \tag{10}$$

Let $P_j(hit) = P(\text{hit}|\text{request for } j)$. Because there is a hit for a request for file j if any M_j of the R_j erasures for file j are available, we have

$$P_j(hit) = \sum_{m=M_j}^{R_j} P(\sum_{r} \Phi_{jr} = m). \tag{11}$$

Let $\mathcal{R}_j(m) = \{A : A \subseteq \{1,\ldots,R_j\} \text{ and } |A| = m\}$. We also have

$$\begin{aligned} P(\sum_r \Phi_{jr} = m) &= \sum_{A\in\mathcal{R}_j(m)} P(\cap_{r\in A}\Phi_r \cap_{r\in A^c} \Phi_r^c) \\ &= \sum_{A\in\mathcal{R}_j(m)} (\prod_{r\in A} p_{jr})(\prod_{r\in A^c}(1-p_{jr})) \end{aligned} \tag{12}$$

where the last inequality follows from the fact that no two erasures from the same file are stored on the same node. Let $\mathcal{T}_j = \{A : A \subseteq \{1,\ldots,R_j\} \text{ and } |A| \geq M_j\}$. Combining (10-12) gives

$$\begin{aligned} P_j(hit) &= \sum_{A\in\mathcal{T}_j}\prod_{r\in A}[1-(1-p)^{n_{jr}}]\prod_{r\in A^c}[(1-p)^{n_{jr}}] \\ &:= h(n_{j1}, n_{j2}, \ldots, n_{jR_j}) \end{aligned} \tag{13}$$

Now consider a reliability system that consists of R_j subsystems, with the rth such subsystem consisting of n_{jr} parallel components, with every component being operational with probability p. Suppose that the system is operational if any M_j out of R_j subsystems are operational. It is easy to see that the probability that the system is operational is given by $h(n_{j1}, n_{j2}, \ldots, n_{jR_j})$. Combining this observation with Theorem 2.2 of Boland et al. [2] implies that $h(n_{j1}, n_{j2}, \ldots, n_{jR_j})$ is Shur concave. This in turn implies that this $h(n_{j1}, n_{j2}, \ldots, n_{jR_j}) \leq h(x_j, x_j, \ldots, x_j)$, where $x_j = (1/R_j)/\sum_{r=1}^{R_j} n_{jr}$. Thus

$$\begin{aligned} \sum_{A\in\mathcal{T}_j}\prod_{r\in A}[1-(1-p)^{n_{jr}}]\prod_{r\in A^c}[(1-p)^{n_{jr}}] &\leq \\ \sum_{A\in\mathcal{T}_j}[1-(1-p)^{x_j}]^{|A|}[(1-p)^{x_j}]^{|A^c|} \end{aligned} \tag{14}$$

Combining (13) and (14) gives

$$P_j(hit) \leq \sum_{m=M_j}^{R_j} \left(\begin{array}{c} R_j \\ m \end{array} \right) [1-(1-p)^{x_j}]^m [(1-p)^{x_j}]^{R_j-m} \tag{15}$$

Now define $z_j = x_j/c_j$. From (15) and the definition of $f_j(z)$ we have

$$P(hit) \leq \sum_{j=1}^{J} f_j(z_j) \tag{16}$$

and from (9)

$$\sum_{j=1}^{J} z_j \leq S. \tag{17}$$

The result follows directly from (16-17). ■

C. Logarithmic Replication Rule

Theorem 1 of the previous section can be used to benchmark adaptive algorithms that use erasures. Moreover, by specializing the theorem to adaptive algorithms without erasures , we can obtain a closed-form expression for the optimal hit probability, which sheds additional insight in on how files should be optimally replicated.

First we note that for the case of no erasures, we have $R_j = M_j = 1$, so that $f_j(z)$ simply becomes $f_j(z) = q_j(1-p)^{z/b_j}$ for all $j = 1,\ldots,J$. Let $(z_1^*, z_2^*, \ldots, z_J^*)$ be the optimal solution to the optimization problem in Theorem 1. This optimal solution can be obtained explicitly as follows. Differentiate $f_j(z)$ and solve for z_j^* such that $f_j(z_j^*)\alpha$, and use (6) to solve for α, and taking special care that $z_j^* \geq 0$ is not violated, we obtain the following solution. Now reorder the files so that they have decreasing values of q_j/b_j. There is an L such that $z_j^* = 0$ if and only if $j > L$. (We will indicate shortly how L is determined.) Define

$$B_L = \sum_{j=1}^{L} b_j$$

which is the amount of storage required by the first L files. Finally, let $n_j^* = z_j^*/b_j$, which has the interpretation of the optimal number of replicas of file j in the continuous relaxation of the problem. After carrying out this exercise, we obtain:

$$n_j^* = \frac{S}{B_L} + \frac{\sum_{l=1}^{L} b_l \ln(q_l/b_l)}{B_L \ln(1-p)} + \frac{\ln(q_j/b_j)}{\ln(1/(1-p)}} \tag{18}$$

It remains to specify how L is determined. This is done by finding the largest L such that $n_L^* > 0$ using (18). This can be done by a simple (linear or binary) search.

The n_j^* given by (18) is non-integer and represents the approximate number of replicas for file j. When these values are used in the expression for $P(hit)$, we obtain a closed-form approximation for the hit probability:

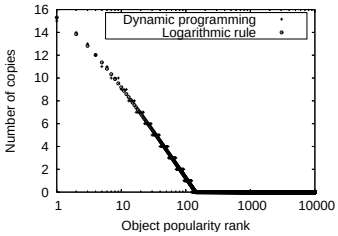
$$P_{approx}(hit) = 1 - a^{S/B_L}\sum_{j=1}^{L} q_j \prod_{l=1}^{L}(\frac{q_l/b_l}{q_j/b_j})^{b_l/B_L} \tag{19}$$

where $a = 1-p$.

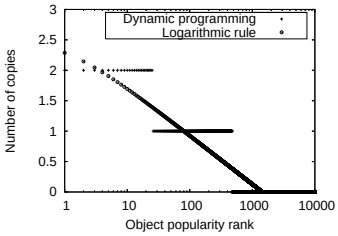
The expressions (18-19) provide significant insight into the nature of the optimal replica profile:

- The ratio q_j/b_j plays a key role in influencing the number of replicas that are assigned to object j. Objects with small values of q_j/b_j (specifically, for objects with $j > L$) are not stored in any of the peers in the optimal solution.
- We call the replication of objects given by (18) the *logarithmic replication rule* since n_j^* is equal to a constant plus a term proportional to $\ln(q_j/b_j)$. It is interesting to note a parallel of the logarithmic replication rule with the *square root assignment rule* that was derived by Kleinrock for the link capacity assignment problem in 1964 [3]!
- The expression (19) is actually upper bound on the true optimal hit rate, since it is the optimal over continuous variables rather than integer variables.

We evaluated the tightness of the upper bound (19) by calculating the difference between between (19) and (3) in several cases. In all cases, we had 100 nodes and each node had storage capacity for 15 objects (all objects assumed to be the same size). We observed similar behavior for other storage capacities not reported here. Our results showed that the upper



(a) Up probability = .2



(b) Up probability = .9

Fig. 2. Values of n_j and n_j^* for Zipf .8 and 5 objects of per-node storage with up probabilities .2 and .9

TABLE I

AVERAGE DIFFERENCE BETWEEN CONTINUOUS SOLUTION AND DYNAMIC PROGRAMMING SOLUTION

Up prob.	Zipf 1.2	Zipf 0.8
0.2	0.02%	0.08%
0.5	0.1%	0.7%
0.9	0.9%	5.8%

bound is typically very tight; the difference was typically less than 1% of the bound given by (19).

We can clearly see the logarithmic replication rule in Figure 2. The replicas given by (18) decrease logarithmically as the file popularity decreases. The replicas given by the dynamic programming are constrained to integer values and decrease in steps, closely following the continuous optimal in most cases. In cases where the most popular objects need only a few copies, such as the case shown in Figure 2(b), the optimal number of integer replicas is coarse and the bound (19) is not as tight (up to 5% difference in our experiments).

We now provide some numerical results. Table I shows the difference between (19) and (3) as a percentage of the upper bound (19). We show the results for three node up probabilities and two values of the Zipf parameter. In all cases, we had 100 nodes and each node had storage capacity for 15 objects (all objects assumed to be the same size). We observed similar behavior for other storage capacities not reported here. The results in Table I show that in most cases the upper bound given by (19) is very tight; typically the difference is less than 1%. However, as the up probability goes up and the Zipf parameter goes down, the difference increases somewhat. This behavior is understood by comparing the optimal number of replicas obtained from the logarithmic replication rule with that obtained from dynamic programming; see Figure 2. The dynamic programming solution is constrained to integer values, whereas the logarithmic replication rule can use real numbers. The difference is more pronounced when the number of copies is small, i.e., in Figure 2(b).

IV. ADAPTIVE CONTENT DISTRIBUTION

We will now discuss a practical, adaptive algorithm, called Top-K Most Frequently Requested algorithm (Top-K MFR) for optimizing file availability in a P2P community.

We consider a DHT-based content distribution community, where each node has a persistent identifier. We assume that each node has access to a DHT substrate which has a function call that takes as input a file identifier j and determines an ordered list of the current up nodes. The substrate then returns, for a given value of K, the first K nodes on the list, $i_1, i_2, \ldots, i_K$. The node i_1 is said to be the current first place winner for file j; the node i_2 is said to be the current second-place winner for file j, and so on. Our work does not rely on any particular DHT, but can use any existing DHT, such as Chord [4], CAN [5], Pastry [6], Tapestry [7], or Kademlia [8]. Most existing DHTs already provide the above winners as specified; others would have to be modified slightly to provide the list of winners.

A. Top-K Most Frequently Requested Algorithm

The Top-K MFR algorithm works as follows. Each node i maintains a table for all files for which it has received a request. For a file j in the table, the node maintains an estimate of $\lambda_j(i)$, the local request rate for the file. In the simplest form, $\lambda_j(i)$ is the number of requests node i has seen for file j divided by the amount of time node i has been up. In practice, we would likely weigh recent requests more heavily in the online calculation of $\lambda_j(i)$. Also, note that ideally the table would contain an entry for *all* objects for which i has received a request; in practice the size of this table could be easily limited to, say, a few thousand most frequently requested objects without any impact on performance.

Each node i stores the files with the highest $\lambda_j(i)$ values, packing in as many files as possible. As we show in Section III, objects with different sizes should be ordered according to $\lambda_j(i)/b_j$, where b_j is the size of object j.

When node i receives a request (from any other node) for file j, it updates $\lambda_j(i)$ and checks if it currently has j in its storage. If i doesn't have j and MFR says it should,[1] then i retrieves j from the outside, puts j in its storage, and possibly evicts one or more files from its storage according to MFR. Node i could retrieve j from the outside, or it could ping the remaining internal winners for the file. (Such pings do not count as requests for j.)

Now that we have defined the retrieval and replacement policy, we need to define the ping dynamics. We want the ping dynamics to influence the rates so that the numbers of replicas across all nodes become nearly optimal. One approach might be for the requesting node X to ping the top-K winners in parallel, and then retrieve the file from any node that has the file. Each of the pings could be considered a request, and the nodes could update their request rates and manage their storage with MFR accordingly.

However, it turns out that the correct approach is for X to *sequentially* request j from the top-K winners, and stop

[1]Node i with storage for n objects, stores the n highest $\lambda_j(i)$.

the requests once j is found. Sequential requests influence the locally-calculated request rates in a manner such that the global replication is nearly optimal. In particular the value of $\lambda_j(i)$ at any node i will be reduced (or "thinned") by hits at "upstream" higher-placed nodes for j. We now summarize the algorithm. Suppose X wants file j. Initialize $k = 1$.

Top-K MFR Algorithm

While $k \leq K$ and X has not obtained j:

1) X uses substrate to determine i, the kth place winner for j.
2) X requests j from i.
   - Node i updates $\lambda_j(i)$.
   - If node i already has j, node i sends j to X; stop.
   - If node i does not have j but it should (according to MFR), i gets j, stores j and evicts files if necessary. Node i sends j to X.
3) $k = k + 1$

If after K iterations, X still does not have j, X gets j from the outside directly (but does not put j in its shared storage). Note that asking the top-K winners sequentially will increase the delay of locating the object (or determining that it is not available). However, since the objects are large, the download delay dominates the total delay experienced by the user. In addition, downloads within the community typically happen at a much faster rate than from outside the community; hence, it pays off to ask the K winners within the community, even sequentially. In practice it is likely that the delay caused by contacting K peers in the community would be negligible and completely unnoticeable by the user.

B. Performance Analysis of MFR

We now present a performance evaluation technique for the MFR algorithm, which is not only accurate and efficient, but also sheds insight into the subtleties of the MFR algorithm.

Given that the MFR algorithms possess many attractive properties, it is desirable to have available a performance evaluation technique for MFR that is more efficient than discrete-event simulation. We now present such a technique,

As before, let I denote the number of nodes. For a given node i, let p_i denote its up probability. Denote by S_i the amount of shared storage in the ith node. Let J denote the number of distinct files, and let b_j denote the size of the jth file. For this performance analysis, we assume that the request probabilities for the J files are known *a priori*. Specifically, we suppose that the request probability for each file j is a known value, q_j, with $q_1 + q_2 + \cdots + q_J = 1$.

We now describe the analytical procedure for calculating the steady-state replica profile and hit probability for Top-K MFR for the case $K = I$. Although we only analyze the case $K = I$, the resulting replica profile and hit probabilities serve as excellent approximations for when K is small.

The procedure sequentially places copies of files into the nodes. Let T_i denote the remaining unallocated storage in node i; let x_{ij} be equal to 1 if a copy of file j has been placed in node i and equal to 0 otherwise. After placing a copy of file j in node i, T_i is reduced and x_{ij} is set to 1.

The procedure first initializes $\gamma_j = q_j/b_j$ for all $j = 1, \ldots, J$ and $T_i = S_i$ for all $i = 1, \ldots, I$. It also initializes $x_{ij} = 0$ for all $i = 1, \ldots, I$, $j = 1, \ldots, J$. At each iteration, the procedure chooses the file with the highest γ_j value, places a copy of that file in a node, and then reduces γ_j appropriately. Specifically,

1) Find the file j that has the largest value of γ_j.
2) Sequentially examine the winning nodes for j until a node is found such that $T_i \geq b_j$ and $x_{ij} = 0$. Then,
   - Set $x_{ij} = 1$
   - Set $\gamma_j = \gamma_j(1 - p_i)$
   - Set $T_i = T_i - b_j$

   If there is no node such that $T_i \geq b_i$ and $x_{ij} = 0$, then remove file j from further consideration.
3) If all files have not been removed from consideration, return to Step 1. Otherwise, stop.

The replication profile provided by this procedure is typically very close to the steady-state profile obtained by the Top-I MFR algorithm. The two profiles may differ slightly due to how files of different sizes are replaced and packed in the nodes, and due to ties in Step 1. However, under the conditions of the following theorem it can be shown that the two profiles are the same. (The proof of the theorem can be found in an extended version of the paper.) Let $\hat{x}_{ij}$, $i = 1, \ldots, I$, $j = 1, \ldots, J$ be the final x_{ij} values from the above procedure. Let $x_{ij}(t)$ be equal to one if at time t there is a copy of file j in node i when the Top-I MFR algorithm is used.

Theorem 2: Suppose all files are the same size. Further suppose that there are never any ties in Step 1 of the procedure. Then $x_{ij}(t)$ almost surely converges to $\hat{x}_{ij}$ for all $i = 1, \ldots, I$ and $j = 1, \ldots, J$.

An interpretation of Theorem 2 is as follows. Note that γ_j is proportional to the request rate to the outside for file j. Whenever a copy of file j is put in node i, the external request rate for file j is 0 when node i is up and γ_j when i is down; thus the expected external request rate is reduced to $\gamma_j(1 - p_i)$. Since Top-K MFR converges to replica profile of the procedure (Theorem 2), the adaptive algorithm has the effect of giving priority to files that have the highest thinned external request rates.

We used this procedure and Theorem 2 to evaluate the performance of the Top-I MFR algorithm. Specifically, we ran the procedure for 30 test cases, with the different cases obtained by different combinations of the node up probability, Zipf parameter, and node storage capacity. Each case had 100 nodes. For the same cases, we obtained the theoretical optimal solution (see Section IV-B). For all of the cases, the conditions of Theorem 2 were satisfied; thus, the procedure provides the replication profile for Top-I MFR algorithm in steady state. We found that in 28 of the 30 cases the MFR algorithm (with $K = I$) converges the optimal replication profile! In the two cases without convergence, the number of replicas were the same for all but two files, with one more replica for one of the files and one less for the other. This experiment further confirms that the MFR algorithms are near optimal.

We now provide a simple example for which the MFR-I algorithm does not produce the optimal replication profile.

TABLE II
REQUEST PROBABILITIES AND WINNERS

File	Req. prob.	1st place winner
1	5/13	1
2	3/13	2
3	3/13	2
4	2/13	1

This example gives insight into why MFR-I is not always optimal. This example has $I = 2$ nodes, each capable of storing two files, each having up probability 0.5. It has four files, with request probabilities and first-place winners shown in Table II. It is easily seen that the Top-2 MFR algorithm puts one copy of file 1 and one copy of file 4 in node 1, and one copy of file 2 and one copy of file 3 in node two. However, the optimal solution puts one copy of file 1 and one copy of file 3 in the first node, and one copy of file 1 and one copy of file 2 in the second node. The problem comes from the fact that MFR first assigns files 1, 2 and 3, thereby filling node 2. For the next assignment, file 1 is more desirable than file 4; however, file 1 cannot be assigned to node 1, as there is already a copy of file 1 there.

V. LOAD BALANCING

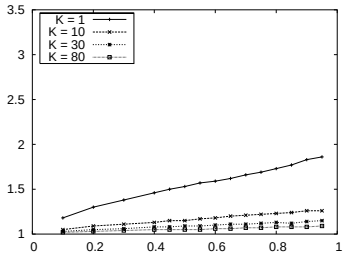
Up until this point our focus has been on managing content to maximize the probability of having a hit in the community. However, consider the case when a very popular object j has a first-place winner i that is almost always up. In this case, the adaptive algorithms will only create one copy of object j, which will be permanently stored on peer i. If the demand for this object is very high, then peer i will become overloaded with file transfers. In this section we present two algorithms for solving the hot-spot problem and evaluate their performance.

A. Fragmentation Approach

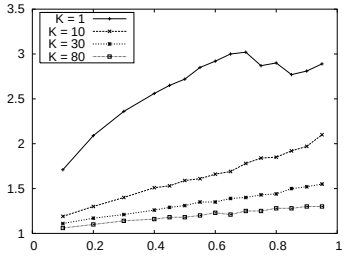
In what we call the *fragmentation approach*, we break each file into several fragments or chunks which are given unique names and stored individually in the DHT. In order to retrieve the file, a peer must retrieve all the fragments. We do not consider erasures in our evaluation, that is, we simply fragment a file into K chunks and each chunk must be retrieved (either from inside or outside the community). Because the fragments of a popular file are likely to stored on different peers and the fragments are much smaller than the original file, the file transfer load on the nodes becomes more balanced.

We evaluated the fragmentation approach using discrete event simulation, as described in [9]. In addition to the parameters defined there, we varied the number of fragments from 1 to 80 per file[2]. Figure 3 shows the 90-percentile of the per-peer load as a function of the up probability for the two Zipf parameters. The y-axis represents the load relative to the "fair share" of each peer. We calculated the fair share by assuming that each peer would handle the same amount of traffic and compared the actual loads to this value. A value

[2]Typical fragment size is 256 KB, hence our range of fragments covers a fair range of file sizes. Furthermore, as the results show, increasing the number of chunks can only improve the balance.



(a) Zipf 0.8



(b) Zipf 1.2

Fig. 3. 90-percentile of per-peer load for fragmentation approach

of 2 indicates that 90% of the peers in the experiment had to serve more than twice the number of files than their fair share.

As a first observation, we can see that for Zipf 1.2 and 1 fragment the load is severely unbalanced. However, as we increase the number of fragments, the load balances itself quite well, and even for Zipf 1.2 remains below 1.3.

Comparing figures 3(a) and 3(b), we see that with Zipf 0.8, all the load curves increase monotonically with the up probability. This is intuitive, since the more peers are up in general, the more of the load they have to handle. The load for Zipf 1.2 peaks earlier, at an up probability of around 0.65–0.7. The results in Figure 3 are calculated after several long experiment runs and we observed the same behavior for all larger Zipf-parameter values.

We believe that the explanation for this is as follows. With higher Zipf-parameter values, the popular files receive a large number of requests. This implies that any peer, who is among the first *few* winners for a popular object, is going to get a large number of requests. Furthermore, up probabilities in the range of 0.5–0.8 imply that peers are typically up, but not enough to make the first-place winner get almost all of the requests. In other words, the load of the most popular objects is spread over several peers, which, in turn, raises the 90-percentile of the load. The load of the most heavily loaded peer typically increases monotonically with the up probability, as can be expected (not shown in the figure).

In summary, the fragmentation approach is very efficient in balancing the load among the peers. The more fragments there are, the better the load is balanced.

B. Overflow Approach

In our second approach, namely the *overflow approach*, individual peers can refuse to serve a request which causes the request to be sent to the next winner, or to the outside, as appropriate. Because peers are independent, an individual peer can use the overflow approach to lower its load by an

arbitrary amount, hence guaranteeing that a peer does not have to serve more requests than it is capable. This guarantee allows us to state that the overflow approach (with or without the fragmentation approach) can deliver us *any* load distribution.

When a peer refuses to serve a request, another peer has to serve it, or the file has to be retrieved from the outside. Therefore, when a peer is using the overflow approach, the load on other peers increases *and/or* the hit-rate goes down. We now evaluate the effects of the overflow approach according to these two metrics. We consider complete files and files with 30 fragments. We performed the same experiments as in Section V-A and measured how much the load increases on each peer. We then calculated the average of the *increases* and show it in the figures.

We randomly determined a fraction of the peers and each of these peers would then refuse a given percentage of the requests it receives. We varied the two percentages and report the values for 10 and 50% of the peers refusing either 10 or 90% of the requests.

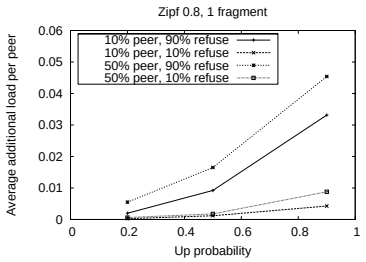
Figure 4(a) shows the increased load per peer as a function of the up probability for the 4 combinations and Zipf-parameter 0.8, for the case of complete files. Figure 4(b) shows the results for 30 fragments per file. The y-axis reports the *additional* load as a percentage of the old load that peers have to handle on average as a direct result of the refusals. For the complete file case, when a small number of peers is refusing a small number of requests, the overall effect on the load is negligible, on the order of 1%. In the worst case, half of the peers refusing 90% of the requests, the additional load on the other peers is about 5–6%. For the 30 fragments per file case in Figure 4(b), the shapes of the curves are similar, but the increase in load is much smaller, less than 0.5% in all the cases. This is because the fragments are small (compared to the case of complete files in Figure 4(a)) and the additional load from the fragments is better distributed over the peers, since all fragments are individually stored on the peers.

In summary, the overflow approach can be used to reduce load, but for complete files this can imply an increase of load by 6% when nodes have high up probabilities. For fragmented files, the increase in load is negligible.

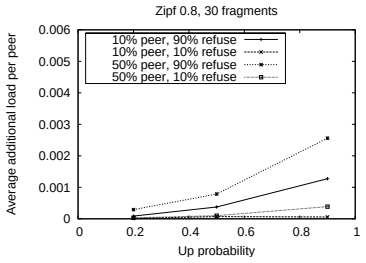
Effect of Overflow on Hit-Rate

As mentioned above, the overflow approach increases load on other peers, but can also decrease the overall hit-rate of the community, if large enough number of peers is refusing a large amount of traffic.

We observed that the refusals to serve traffic have the same effect when we would reduce the amount of storage in the nodes. In other words, if 50% nodes are refusing 90% of the traffic, the community has roughly the same hit-rate as a community which serves all the traffic, but has only 55% of the storage of the community refusing traffic. The explanation behind this is intuitive. If a peer refuses to serve a request, the request gets passed to the next-place winner for the object, who will then see a higher request rate for that object. This increases the priority of that object in the MFR algorithm and might kick a less requested object out of the storage. This effect propagates along all the nodes and causes the least requested objects to be completely evicted from the community, thus effectively reducing the amount of storage provided by the community.



(a) Complete file



(b) 30 fragments

Fig. 4. Increased load due to overflow approach for Zipf = 0.8

VI. Related Work

Gkantsidis et al. [10] consider a large-scale software update system. The bulk of their work considers measurements made from an existing software patching system, but they also evaluate the effectiveness of community-based content distribution schemes in P2P content distribution. Their main concern in terms of P2P content distribution is in evaluating how different peer selection strategies affect the inter-AS traffic. Their results show that selecting nearby peers, i.e., forming an ISP-based community, is effective at reducing traffic to outside of the ISP. Karagiannis et al. [11] consider measurement results from BitTorrent tracker logs and show that considerable bandwidth savings for ISPs would be possible, if P2P content distribution would be based on communities, instead of peers downloading files from external sources. However, the results are likely to apply only to larger communities.

Lv et al. [12] and Cohen and Shenker [13] studied optimal replication in an unstructured P2P network in order to reduce random search times. Our work differs in that we are replicating content in structured networks, and we take intermittent connectivity explicitly into account. Furthermore, we replicate to decrease average file transfer time rather than to decrease the random search time in an unstructured P2P system.

There has also been work comparing replication and erasure coding in distributed storage infrastructures [14]. The authors also discuss the drawbacks of using erasures. The paper focuses on persistent storage, and does not consider adaptive replication and replacement of replicas or erasures. Work in [15] discusses when erasures are preferable to complete file replication. Their main result shows that when the availability of the components (up probability of peers in our case) is below a certain threshold, then complete file replication is to be preferred. The exact threshold depends on the number

of erasures and the behavior of the peers. Rodrigues and Liskov [16] also compare replication and erasures and conclude that erasures have an advantage in terms of bandwidth consumption, but conclude that the advantage is not large enough to merit the added complexity of erasures.

One early proposals for P2P-based content distribution was Squirrel [17], which is a distributed P2P web cache. The work focuses on detailed protocol design and implementation and does not address the fundamental issues of how content should be managed in a P2P content distribution community.

FarSite [18]–[20] is a P2P file system with the strong persistence and availability of a traditional file system. The FarSite filesystem uses the same number of replicas – three – for each file. Glacier [21] is a distributed storage system which uses erasures to store the files. The goal of Glacier is to provide high availability of files in the presence of large-scale failures. They evaluate the storage overhead required for a given level of availability, assuming that each file requires the same number of fragments (5). In contrast with a file system, the goal of a P2P community is not to provide strong file persistence, but instead, maximal content availability. Thus, in a P2P community, the number of replicas of a file depends on the popularity of the file.

Network coding [22], [23] could possibly help improve the hit-rate in a fragment-based community. Unavailable fragments, which could be generated from available fragments, could be made available. However, algorithms based on request rates, like MFR, likely replicate all fragments in a similar way. Hence, it is likely that either most of the fragments are unavailable and network coding cannot help, or that most of the fragments are available, and the possible gain in hit-rate from being able to generate the missing fragments is low.

In our previous work [9], we have considered the content management problem from an experimental point of view. In [9] we present several adaptive algorithms (including the MFR algorithm) and evaluate their performance through discrete event simulation. The experimental results confirm our theoretical results from this paper by demonstrating the extremely good performance of the MFR algorithm (near-optimal in most practical scenarios).

VII. Conclusions

In this paper, we have introduced an optimization methodology for maximizing file availability in peer-to-peer content distribution. Our methodology directly applies to communities whose nodes have homogeneous up probabilities, and can be extended to heterogeneous environments. The methodology applies to designs that use erasures. The main result is the logarithmic replication rule which states that the number of copies for a file should be proportional to the logarithm of the ratio of the file's request probability and file size.

We have shown that a simple Top-K Most Frequently Requested algorithm almost always achieves optimal performance. We have also evaluated different load balancing mechanisms and have found out that even a simple fragmentation of the files will lead to an evenly balanced request load. The load can be adjusted by allowing individual peers to refuse requests, which, according to our results, has only a negligible effect on the other peers.

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[2] P. J. Boland, E. El-Neweihi, and F. Proschan, "Stochastic order for redundancy allocations in series and parallel systems," *Advances in Applied Probability*, vol. 24, no. 1, pp. 161–171, Mar. 1992.

[3] L. Kleinrock, *Queueing Systems. Volume II: Computer Applications*. New York: John Wiley & Sons, 1976.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *ACM SIGCOMM*, San Diego, CA, Aug. 27–31, 2001.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *ACM SIGCOMM*, San Diego, CA, Aug. 27–31, 2001.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001.

[7] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan. 2004.

[8] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proceedings of International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 7–8, 2002.

[9] J. Kangasharju, K. W. Ross, and D. A. Turner, "Adaptive content management in structured P2P communities," in *International Conference on Scalable Information Systems*, Hong Kong, China, May 2006.

[10] G. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovic, "Planet scale software updates," in *ACM SIGCOMM*, Pisa, Italy, Aug. 2006.

[11] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should Internet service providers fear peer-assisted content distribution?" in *Proceedings of Internet Measurement Conference*, Berkeley, CA, Oct. 2005.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proceedings of the 16th annual ACM International Conference on Supercomputing*, New York, NY, June 22–26, 2002.

[13] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[14] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proceedings of International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002.

[15] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in *International Conference on Peer-to-Peer Computing*, Zurich, Switzerland, Aug. 2004.

[16] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *Proceedings of International Workshop on Peer-to-Peer Systems*, Ithaca, NY, Feb. 2005.

[17] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in *Proceedings of ACM Symposium on Principles of Distributed Computing*, Monterey, CA., July 21–24, 2002.

[18] A. Adya et al., "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," in *Proceedings of Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[19] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.

[20] J. R. Douceur and R. P. Wattenhofer, "Optimizing file availability in a secure serverless distributed file system," in *Proceedings of IEEE Symposium on Reliable Distributed Systems*, 2001, pp. 4–13.

[21] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *Proceedings of Symposium on Networked Systems Design and Implementation*, Boston, MA, May 2005.

[22] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[23] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proceedings of IEEE Infocom*, Miami, FL, Mar. 2005.