Decentralized Replication Algorithms for Improving File Availability in P2P Networks

W.K.Lin, C.Ye, D.M.Chiu
Department of Information Engineering, The Chinese University of Hong Kong
wingkai.lin@gmail.com, {cye5, dmchiu}@ie.cuhk.edu.hk

***Abstract*— Being autonomous and scalable, Peer-to-Peer systems provide a paradigm for sharing files in the Internet. However, different from conventional structured replication systems like content distribution networks (CDN), peers in an unstructured P2P system may have different, sometimes low, online availability, and usually get only partial information about the resources of the system. Therefore, how to achieve good system level file availability by autonomous peers is an important goal in P2P replication systems. In this paper, we investigate decentralized and cooperative resource allocation algorithms in a class of P2P systems that provide replication service. We formulate this replication problem as an optimization problem, and propose several heuristic algorithms respectively. They include (a) a random algorithm, (b) a group partition algorithm that relies on peers' forming groups, and (c) a greedy search algorithm based on an estimated system-level file availability target. We compare and evaluate these algorithms by simulations, and observe that each of them has advantages depending on the system parameters.**

I. Introduction

Peer-to-peer(P2P) applications have become tremendously popular as a way of sharing data in the Internet. Existing P2P systems can be categorized into two groups: (a) a distributed system with central planning and deployment; (b) a distributed system based on ad hoc participation. Examples of (a) are content distribution networks (CDN) [1], serverless video streaming systems [2] and distributed file storage systems [3], [4]. Examples of (b) include file swapping networks [5], [6], application layer multicast services [7] and application layer multicast-based streaming [8]. Naturally, some systems can be considered as in-betweens, such as Napster [9] and Skype [10], where there are central directories or authentication services.

In this paper, our interest is in viewing a class of P2P systems as a mostly ad hoc replication system. In this system, each peer has a number of files to share and also *cooperates* to offer storage space to replicate a collection of files. However, peers are not online all the time. The ad hoc nature, determining how files are replicated, may potentially lead to poor and uneven file availability without proper coordination. Therefore, it is believed that file availability is an important issue when deploying replication service in P2P systems. We interpret this as a *resource allocation problem* which includes the following two issues:

- *Storage allocation*: to decide how many replicas can be produced for each file upon the limitation of storage space.
- *Replica placement*: to decide the set of peers who are going to store those replicas of each file so as to achieve a reasonable level of file availability.

If there is only a small number of files and peers in the replication system, one can centrally search for an optimal solution by enumerating all possible schemes. However, the solution space increases dramatically when the number of peers and files increases, and therefore centralized algorithms will become computationally infeasible. We are concerned about deploying decentralized algorithms for peers to collaboratively solve this resource allocation problem. It is assumed all through this paper that some incentive mechanisms exist in the system to avoid free-riding and to encourage peers' cooperation, as in [11]. Three decentralized replication algorithms are studied:

1) A random algorithm that requires the least information from neighbor peers.
2) A group partition algorithm that attempts to achieve an even file availability distribution using more partitioned knowledge.
3) A greedy search algorithm based on the estimated system-level file availability target.

These algorithms are evaluated by simulations. It is observed that each can be useful depending on system parameters.

There has been a considerable amount of research work on object replication to different locations or computers. In [12], the author studied the file allocation problem of replicating a single file over a set of computers. The file is demanded by various users who have different access costs to different computers. The author proved that the optimal assignment of the replicas to the computers that minimizes the total cost is NP-complete. Authors in [13] studied the problem of replicating objects optimally in content distribution networks, in which each autonomous system (AS) replicates some objects and demands objects stored in other ASes at the same time. The optimality of this replication is defined as the minimization of the average inter-AS hop distance a request must transverse. They proved that this optimization is NP-complete, and proposed some centralized heuristic algorithms to solve the replication problem.

Web caching can also be regarded as a distributed replication network. In [14], the authors studied the problem of distributing a fixed set of web proxies in the Internet. There was a setup cost involved when assigning a proxy to a potential site, as well as a link cost for connecting any two potential sites. They modeled the network topologies as trees, and proposed a centralized algorithm to distribute the proxies with minimum total cost. Authors in [15] also investigated the placement of a fixed set of web server replicas to potential web sites. Instead of using tree-based models, they modeled the placement problem as a

K-median problemand developed several centralized placement algorithms to solve it.

Another approach can be found in [16], which focused on the problem of assigning file replicas according to the files' demand characteristics. The authors described a decentralized and adaptive protocol to replicate files in a way that balances server load. A message-based protocol used to replicate resources in a replication network was developed by Ko *et al.* [17]. In their model, each piece of resources is tagged with a color, and a distributed algorithm was developed for each node to operate, with an attempt to maximize its own distance to a node that holds an object with the same color. The algorithm eventually converged to an optimal allocation. Authors in [18] considered assigning replicas in an unstructured P2P system, focusing on minimizing the expected search size of the search queries. Their results showed that replicating files proportionally to the square root of a file's demand popularity is an optimal replication scheme. While ignoring the issue of search cost, authors in [19] proved that having the number of file replicas proportional to their request rates can minimize the average downloading bandwidth. They showed that such replica distribution can be achieved in a decentralized way by using LRU, with a system performance close to optimal.

Our work in this paper differs from previous works in the following aspects. The problem addressed in [12], [14], [15] was concerned with how to replicate a file so as to minimize transmission cost and link delay. [16] investigated the problem of balancing server load by replication. And the replication objectives in [18] and [19] were to minimize a search metric and downloading bandwidth respectively. As an orthogonal evaluation to the existing replication related studies, we put our focus on how to replicate a set of files so as to optimize file availability. There has already been some research dealing with this issue, such as [20], [21]. However, in our model, peers are allowed to have different availability, which is more general in comparison with previous models [12], [13], [15], [20]. Furthermore, we take erasure code replication into account, which can degenerate to traditional replication and achieve better replication performance for certain system parameters [22]. Finally, our work differs from [21] in that [21] tried to achieve a pre-determined availability target. Our algorithms, on the other hand, try to optimize file availability based on given resources.

II. P2P REPLICATION SYSTEM

In this section, we interpret a P2P network as a replication system, in which peers cooperate to replicate files for each other with the use of erasure coding. Table I summarizes the parameters used in our model.

$\mathcal{F}_i$	The set of files to be replicated in peer i
$\mathcal{F}$	The set of files in the system, $\mathcal{F} = \bigcup_i \mathcal{F}_i$
$\mathcal{P}_i$	Writable peer set of peer i
$\mathcal{P}$	The set of peers in the system, $\mathcal{P} = \bigcup_i \mathcal{P}_i$
N	Number of peers in the system: $N = \|\mathcal{P}\|$
M	Number of files in the system: $M = \|\mathcal{F}\|$
Γ	Size of each erasure coded block
$\mathbf{p}$	Peer availability vector
$\mathbf{s} = [s_i]$	Peer storage capacity
$\mathbf{f} = [f_j]$	File size of each file j
$\mathbf{b} = [b_j]$	Number of blocks *before* erasure coding each file j
$\mathbf{\Omega} = [\Omega_j]$	Storage overhead of each file j
$\mathbf{k} = [k_j]$	Number of blocks *after* erasure coding each file j
$\mathbf{R} = [r_{i,j}]$	A feasible replica placement
$\mathbf{p}[\mathbf{r}_j]$	Availability vector of peers replicating file j
$\mathbf{A} = [A_j]$	File availability distribution

TABLE I
SYSTEM PARAMETERS

A. Peers

In our model, peers are assumed to cooperate aiming at the overall replication goal. When a peer joins the system, it is willing to offer a certain amount of storage resources for other peers to place their file replicas. In return, it can also distribute its file replicas to other peers, thereby increasing availability of its own files.

Each peer in this replication system is characterized by three parameters. First, we denote the online availability $p_i \in [0, 1]$ as the proportion of the time peer i stays online. When a peer is online, all the replicas it stores are available and accessible by other peers in the replication system. Therefore, the probability of retrieving the replicas stored in peer i is equal to its availability p_i. Second, peer i has a set of files $\mathcal{F}_i$ that needs to be replicated. We do not consider the bandwidth consumption between peers in this paper, and file replicas are assumed to be assigned to other peers in a negligible amount of time. The third parameter is the amount of storage space that peer i offers for replication purposes, denoted by s_i. This shared storage space is made available to other peers in the system.

B. Erasure code replication

Peers in this replication system adopt erasure code to replicate files. In erasure code replication, a file is divided into b blocks. A variable amount of erasure code redundancy is then added to these blocks so that $k > b$ blocks are obtained in total, with each block having the same size as before. The erasure-coded blocks are dependent on each other. Retrieving *any* b out of k blocks is enough to reassemble the original file. Authors in [22] provided a comparison between erasure code replication and traditional replication (i.e. replicating a complete file). They also pointed out that when $b = 1$, erasure code replication is equivalent to traditional replication. Therefore, there is no loss in generality by assuming the use of erasure code.

Here we assume that the blocks created are assigned to k different independent peers. The file availability A is defined as the probability of recovering the original (and complete) file based on these stored blocks. As one peer stores one single block of a file, this availability is equal to the probability of having at least b out of k peers online:

$$A(\{p_i\}, b, k) = \sum_{h=b}^{k} P\{h \text{ peers are online}\} \qquad (1)$$

where $\{p_i\}$ represents the online availability of the k peers.

C. Estimation of file availability

The availability of a file after replication is comprised of two parts: the erasure-coded blocks stored in the network and the entire file in the original peer, provided it is kept there. The probability of having exactly h peers/erasure-coded blocks available equals to the sum of the probabilities of any permutation that have h peers online out of k:

$$\begin{aligned} P\{h \text{ peers are online}\} = \\ & p_1 p_2 \ldots p_h (1-p_{h+1})(1-p_{h+2})\ldots(1-p_k) + \\ & p_2 p_3 \ldots p_{h+1}(1-p_{h+2})\ldots(1-p_k)(1-p_1) + \ldots + \\ & p_{k-h+1} p_{k-h+2} \ldots p_k (1-p_1)(1-p_2)\ldots(1-p_{k-h}) \end{aligned} \quad (2)$$

Since each peer in the system may have a different online probability, it will be too expensive to calculate the exact file availability. Therefore, we use the average peer availability as an estimation instead. Hence, the file availability gained from erasure coding can be calculated as:

$$A(\{p_i\}, b, k) = \sum_{h=b}^{k} C_k^h \cdot \bar{p_i}^h \cdot (1-\bar{p_i})^{k-h} \quad (3)$$

where $\bar{p_i}$ refers to the average peer availability of set $\{p_i\}$. By including the original copy, the total file availability becomes

$$A(\{p_i\}, b, k) = 1 - (1-p_i) \cdot \sum_{h=k-b+1}^{k} C_k^h \cdot (1-\bar{p_i})^h \cdot \bar{p_i}^{k-h} \quad (4)$$

where p_i is the availability of the peer who shares this file.

D. Problem formulation

A real world P2P replication system is complicated to model. Part of the problem comes from the complexity of network topology itself, while others, for example, from network dynamics and protocol messages. In this paper, we propose a replication model to abstract the above aspects. Although the model is simple, it is capable of illustrating the difficulties in resource allocation in a real world P2P replication system, specifically, heterogeneity of peer availability and the complexity of replica assignment.

Let us consider a replication system with a fixed population of $\mathcal{P}$ peers whose availability distribution is $\mathbf{p}$. Each peer i has a set of files $\mathcal{F}_i$ to replicate. The set of all files to replicate in the system is $\mathcal{F} = \cup_i \mathcal{F}_i$. We denote the number of peers as $N = |\mathcal{P}|$ and the number of files as $M = |\mathcal{F}|$.

Before replicating, a file $j \in \mathcal{F}$ is first divided into b_j blocks, with each block in the size of Γ. Hence, $f_j = b_j \Gamma$. In addition, all files are segmented by the same block size Γ, and erasure coding produces redundancy without changing the block size. This means $f_j / f_{j'} = b_j / b_{j'}$ for any two files j and j'. For a given file $j \in \mathcal{F}_i$, peer i needs to decide how much erasure coding redundancy should be added, which is denoted by the *stretch factor* Ω. For a particular stretch factor Ω_j, peer i creates $k_j = \Omega_j b_j$ erasure-coded blocks, to be assigned to k_j different peers. For simplicity, we assume the storage space offered by peer i is always in units of the block size Γ. This assumption is justified when the block size Γ is comparatively small with shared storage space s_i, and this is generally true for current commodity storage products.

From the angle of individual file, Ω_j should be as large as possible in order to maximize file availability. The largest possible value is given by $\Omega_j b_j = N$. However, the total storage space that is offered by peers is limited, so it is not always feasible for each file to be replicated by all peers. This implies the need of determining a suitable Ω_j for each file j from the angle of overall replication system.

More generally, we can formulate the problem as to seek an erasure-coded block assignment policy. We define the replication matrix $\mathbf{R} = [r_{i,j}]_{N \times M}$, where $r_{i,j}$ indicate whether an erasure-coded block of file j is assigned to peer i:

$$r_{i,j} = \begin{cases} 1 & : \text{ if peer } i \text{ stores a block of file } j \\ 0 & : \text{ otherwise} \end{cases}$$

where

$$\begin{aligned} i &= 1, 2, \ldots N \\ j &= 1, 2, \ldots M \end{aligned}$$

Obviously, peer i cannot store more than its storage capacity s_i:

$$\sum_{j=1}^{M} r_{i,j} \le s_i \quad \forall i \quad (5)$$

The number of replica blocks of file j stored in the system is equal to k_j:

$$\sum_{i=1}^{N} r_{i,j} = k_j = b_j \Omega_j \quad \forall j \quad (6)$$

A replica placement $\mathbf{R}$ is *feasible* only if it satisfies both conditions 5 and 6.

Let $\mathbf{r}_j$ denote the j^{th} column vector of the replica placement matrix $\mathbf{R}$, which then gives the subset of peers that replicate file j. We select the availability of peers who replicate file j (i.e., $r_{i,j} = 1$), denote it as $\mathbf{p}[\mathbf{r}_j]$. Then the availability of file j can be readily computed as in equation 1:

$$A_j = A(\mathbf{p}[\mathbf{r}_j], b_j, k_j)$$

Based on this, we are able to rigorously define the replication resource allocation problem as to find an optimal $\mathbf{R}$, where the optimality condition is defined by certain system performance metrics.

E. Performance metrics

In order to evaluate the replication algorithms systematically, we employ two performance metrics: the overall expected file availability $E[\mathbf{A}]$ and the variance of file availability $var[\mathbf{A}]$. The expectation measures how well the peers replicate, while the variance serves as a fairness measurement of the achieved file availability distribution.

Due to storage limitation, it happens that some files cannot be replicated at all. Although users may be able to get access to an unreplicated file from the peer who shares it, provided the entire original copy is retained, the file in discussion is still considered to have 0 file availability. That is to say, the contribution of the original copy is excluded. The reason behind this is to extract the file availability achieved by replication from the dependence on availability of the original copy. This promises a more explicit performance evaluation of the replication algorithms. In fact, it is true that peers may not always keep the files they share. In addition, when computing the expectation and variance of the file availability distribution, all files are assumed to have equal weight.

III. Decentralized decisions

In fact, similar resource allocation problems were considered as combinatorial optimization problems in previous studies. They were invariably proved as NP-complete, and coupled with some heuristic solutions, as in [14], [15]. Normally, such heuristic solutions were run by a central agent that had all the necessary system parameters.

However, in a typical P2P system, there might be a huge number of peers whose participations are not synchronized, making timely collection of the system parameters from all peers intractable. Even if it is possible to collect all the parameters needed, it would be very time consuming for a central agent to solve this problem and distribute solutions to all other peers. Therefore, we focus on decentralized solutions that offer each peer autonomous operations.

A. Writable peer set

P2P replication systems are constituted by connected peers. Unlike traditional centralized replication system like RAID, peers in a P2P replication system may not be aware of the presence of *all* other peers in the system. For example, random peers' connections in Gnutella, together with limited flooding search, essentially limit the number of inter-peer connections [23].

We characterize the limited information available to each peer by introducing *degree of connectivity* for a peer. This is not physical connectivity, but actually, the logical reachability of a peer in terms of asking other peers to help it replicate a file. In this sense, a replication system with an indexing server, which allows each peer to know of all other peers' existence, can be considered as a replication system with 100% connectivity, despite the fact that peers are not directly connected to each other.

Given a degree of connectivity, we define a *reachable peer set* of peer i as the set of peers that peer i can potentially reach for replication. However, a peer can either use its entire reachable peer set or randomly choose a subset from it when actually performing replications. We further name the peer set that peer i uses for replication the *writable peer set* $\mathcal{P}_i$. And we assume that no peer is left isolated in the system, therefore:

$$\mathcal{P} = \cup_i \mathcal{P}_i$$

As described before, each peer requires several types of information from other peers in the writable peer set to facilitate making replication decisions. Such information can be encapsulated in the control protocol messages of a P2P system (such as the ping-pong messages in Gnutella), or can be transmitted in a separate protocol message. For each peer i' in the writable peer set of peer i, we define three types of information to be conveyed from i' to i.

1) The storage space offered by peer i' for replication, i.e. $s_{i'}$.
2) The total size of files that peer i' requests to replicate, i.e. $\sum_{j\in\mathcal{F}_{i'}} f_j$.
3) Online availability of peer i', i.e. $p_{i'}$.

Note that peer i has to make a "blind" decision if none of these information is available. Therefore, we assume that at least the first two types of information can be estimated by any peer and delivered readily to other peers. The third type of information, a peer's online availability, cannot be assumed to be always available because it is difficult to be measured accurately, even by the peer itself [24].

B. Stretch factor estimation

As discussed in section II-D, a feasible replica placement solution must satisfy the storage constraint, namely, all created blocks must fit into the storage space offered by the peers. To ensure this feasibility in a decentralized manner, each peer simply collects the pertinent information from its writable peer set and estimates suitable stretch factors for its files. The stretch factor Ω, which controls the amount of redundancy applied to a file through erasure coding, and hence the amount of storage overhead required for replicating the file, is defined as the ratio of storage required with versus without erasure coding.

The storage space available in a peer i's writable peer set is:

$$S_i = \sum_{i'\in\mathcal{P}_i} s_{i'} \tag{7}$$

The total size of files that need to be replicated is:

$$F_i = \sum_{i'\in\mathcal{P}_i} \sum_{j\in\mathcal{F}_{i'}} f_j \tag{8}$$

The ratio of these two parameters gives an estimate of the stretch factor, i.e.

$$\Omega_i = \frac{S_i}{F_i} \tag{9}$$

In order to avoid incorrect estimation of the stretch factor, we employ the *locking phase* strategy. To be specific, once peer i begins the replication process (including estimation of Ω_i), all peers in its writable peer set $\mathcal{P}_i$ are set to be locked. While in the locking phase, a peer is "invisible" to other peers in the system except for peer i, implying that a peer in locking phase cannot participate in other peers' replication process except for peer i's. Any other peer who wants to include peers currently locked in its own writable set should wait until peer i finishes, that is, when those peers are released. The locking phase strategy will definitely prevent the situation that two peers having common elements in their writable sets replicate simultaneously, which would otherwise result in inaccurate estimation of the stretch factors. Therefore, each peer in our system will have a phase indicator to indicate whether it is in the locking phase. When a peer decides its writable set, the locked peers will be filtered unless it is willing to wait until they are released.

IV. Heuristic replication algorithms

Three decentralized heuristic algorithms are described in this section. The first one is *random algorithm* in which peers assign the erasure-coded blocks randomly to their writable peer set. The second one is *group partition algorithm* where peers replicate files in a way to minimize the variance of the resultant file availability distribution. The third one is *highest available first (HAF) algorithm*, which is basically a greedy algorithm that tries to satisfy an estimated availability target. In these algorithms, each peer operates independently, based on the storage resources and information provided by peers in its writable peer set.

A. Random algorithm

Generally, the availability of a file depends on (a) how much redundancy is applied by erasure coding, which is measured in terms of the stretch factor of that file, and (b) the availability of those peers who replicate the erasure-coded blocks of that file. The random algorithm tries to give all files the same stretch factor while assigning randomly selected peers to replicate the erasure-coded blocks. This is a simple yet reasonably fair algorithm because it does not require any knowledge of peer availability, and gives each file the same stretch factor and equal opportunity in selecting peers. The random algorithm can be easily implemented in a distributed manner. Each peer executes the following two steps.

The random algorithm

```
Writable peer set estimation:
    1. Peer $i$ chooses the writable peer set $\mathcal{P}_i$.
    2. All peers in $\mathcal{P}_i$ are "locked".
    3. Estimate $S_i = \sum_{i' \in \mathcal{P}_i} s_{i'}$ and $F_i = \sum_{i' \in \mathcal{P}_i} \sum_{j \in \mathcal{F}_{i'}} f_j$.
    4. Estimate $\Omega_i = \frac{S_i}{F_i}$.
To replicate file $j$:
    5. Divide file $j$ into $b_j$ blocks so that $f_j = b_j\ \Gamma$.
    6. Apply erasure code to create $k_j = \Omega_i b_j$ blocks.
    7. IF peer $i$ cannot find $k_j$ peers with available storage space, skip
       replicating this file.
    8. ELSE peer $i$ randomly picks $k_j$ peers from $\mathcal{P}_i$ to store the
       erasure-coded blocks of file $j$.
       Update available storage space of these $k_j$ peers.
    9. Peers in $\mathcal{P}_i$ are released.
```

Storage allocation: First, each peer i calculates the total storage space offered (S_i) and the total size of all files to be replicated by peers (F_i) in its writable peer set. Peer i then estimates the stretch factor Ω_i for all files $j \in \mathcal{F}_i$ by equation 9 and applies this stretch factor to create $k_j = \Omega_i b_j$ erasure-coded blocks for each file $j \in \mathcal{F}_i$. This stretch factor Ω_i estimates how much storage space each file (in its writable peer set) can use *on average*. If all peers cooperate and follow this estimation, it is very likely that the storage space will not be overused.

Replica placement: After creating the erasure-coded blocks, peer i randomly picks k_j peers, whose storage space is not exhausted, in its writable peer set $\mathcal{P}_i$. Erasure-coded blocks of file j are then assigned to these peers. Peer i stops the replication process when all its files are replicated, or when storage space in the writable peer set runs out.

B. Group partition algorithm

The random algorithm allows peer to independently estimate a "fair" stretch factor for replication, yet the random replica placement step introduces a high variance to the resultant file availability. Some files may be "lucky" when their erasure-coded blocks are assigned to highly available peers, while there will be "unlucky" files whose blocks are replicated by peers with low availability. The group partition algorithm tries to minimize the variance of the file availability distribution based on peer availability information. It also has two steps:

Storage allocation: Peer i allocates storage space in the same way as in the random algorithm. Peer i therefore generates $k_j = \Omega_i b_j$ erasure-coded blocks for each file j, where Ω_i is given by equation 9.

Replica placement: Assume peer i generates k_j erasure-coded blocks for file j. It first collects peer availability information from all peers in its writable peer set, and then sorts the peers (who has available storage space) in descending order according to their availability. The ordered peer set is then logically partitioned into k_j groups $\{g_1, g_2, \ldots, g_{k_j}\}$ so that group g_1 contains the highest available peers and g_{k_j} contains the lowest available peers. Peer i then randomly selects a peer from each group and assigns an erasure-coded block to that selected peer. The replication process terminates when all files of peer i are replicated, or when storage space in the writable peer set runs out.

The group partition algorithm

```
Writable peer set estimation:
    1. Peer $i$ chooses the writable peer set $\mathcal{P}_i$.
    2. All peers in $\mathcal{P}_i$ are "locked".
    3. Estimate $S_i = \sum_{i' \in \mathcal{P}_i} s_{i'}$ and $F_i = \sum_{i' \in \mathcal{P}_i} \sum_{j \in \mathcal{F}_{i'}} f_j$.
    4. Estimate $\Omega_i = \frac{S_i}{F_i}$.
To replicate file $j$:
    5. Divide file $j$ into $b_j$ blocks such that $f_j = b_j\ \Gamma$.
    6. Apply erasure code to create $k_j = \Omega_i b_j$ blocks.
    7. IF peer $i$ cannot find $k_j$ peers with available storage space, skip
       replicating this file.
    8. ELSE:
       8.1 Order the peers with available storage in $\mathcal{P}_i$ in descending
           order according to their availability. Then partition the peers
           into $k_j$ groups.
       8.2 Randomly select a peer in each group and assign an
           erasure-coded block to that peer.
           Update available storage space of these peers.
    9. Peers in $\mathcal{P}_i$ are released.
```

C. Highest available first (HAF) algorithm

The highest available first (HAF) algorithm tries to replicate each file to achieve a target availability A^*. There are different ways to achieve such a target, for example, by trying to use more peers with low availability, or to use as few higher available peers as possible. The HAF algorithm takes the latter approach.

Theoretically, a suitable target file availability A^* can only be determined by the global knowledge of all file sizes, the storage space offered and the availability information of all peers in the system. This obviously cannot be assumed for a decentralized algorithm. Instead, we will describe an associated algorithm for dynamically computing the value of A^* based on the limited knowledge available to each peer. The complete algorithm has the following steps:

Initialization: Before replicating the first file, peer i initializes its file availability target A^* as follows. First, it computes the estimated stretch factor Ω_i for its writable peer set by using equation 9. Then it collects the file size and peer availability information from its writable peer set, and computes the average number of blocks per file before applying erasure code $\overline{b}$ and average peer availability $\overline{p}$. Then it sets the initial value of A^* based on Ω_i and $\overline{b}$, assuming all peers have the same availability $\overline{p}$.

Tentative replica placement: To replicate file j, peer i first orders the peers (with available storage space left) in its writable peer set in descending order according to their availability. It then divides file j into b_j blocks where $f_j = b_j\Gamma$. No erasure coding is applied at this stage and hence $k_j \leftarrow b_j$. Next, it selects k_j peers in the ordered peer set, starting from the highest available peer first, and computes the file availability A_j of file j using equation 1, assuming these highest available peers are used to replicate file j.

Comparing availability with target: The computed file availability is compared to the target A^*. If $A_j < A^*$, peer i will increase the storage overhead for file j by adding an extra erasure code redundancy to create $k_j \leftarrow b_j + 1$ blocks. Once again, it selects k_j highly available peers in the ordered peer set, and computes a new file availability A_j. This whole process is repeated until $A_j > A^*$.

Replica placement: If file j gets a file availability of $A_j > A^*$ from the comparing step, peer i then will create k_j erasure-coded blocks for file j and distribute them to the selected k_j peers. However, if all peers in the writable peer set are selected and yet $A_j < A^*$, peer i then will create and distribute each peer (with enough remaining storage) in its writable peer set with one erasure-coded block of file j. The replication process terminates when all files of peer i are replicated, or when storage space in the writable peer set runs out.

The highest available first (HAF) algorithm

```
In initialization stage:
  1. Peer $i$ chooses the writable peer set $\mathcal{P}_i$.
  2. All peers in $\mathcal{P}_i$ are "locked".
  3. Estimate $S_i = \sum_{i' \in \mathcal{P}_i} s_{i'}$ and $F_i = \sum_{i' \in \mathcal{P}_i} \sum_{j \in \mathcal{F}_{i'}} f_j$.
  4. Estimate $\Omega_i = \frac{S_i}{F_i}$, the average peer availability $\bar{p}$ in $\mathcal{P}_i$ and the
     average number of blocks per file $\bar{b}$.
  5. Initialize $A^*$ based on $\Omega_i$, $\bar{b}$ and $\bar{p}$.
To replicate file $j$:
  6. Divide the file into $b_j$ blocks so that $f_j = b_j\Gamma$.
  7. Do not apply erasure code at this stage, $k_j \leftarrow b_j$.
  8. Calculate the file availability $A_j$ based on the $k_j$ most available
     peers with enough storage in $\mathcal{P}_i$.
  9. WHILE $A_j < A^*$:
     9.1 Increase the stretch factor and create one more erasure-coded
         block, i.e. $k_j \leftarrow k_j + 1$.
     9.2 Select the next most available peer to replicate.
     9.3 Calculate the file availability $A_j$.
  10. IF all peers in $\mathcal{P}_i$ have been selected and $A_j < A^*$, replicate this
      file by distributing each peer with one erasure-coded block.
  11. ELSE IF the new file availability $A_j \geq A^*$, assign the
      erasure-coded blocks to the selected peers.
  12. Update the available storage of the peers who participate in
      replication.
  13. Peers in $\mathcal{P}_i$ are released.
```

Simulation	Peer availability	Average storage	Connectivity
S1.1	Uniform	$\Omega^* = 1.5$	$(1-m) \in$ [0,1]
S1.2	Uniform	$\Omega^* = 2.0$	$(1-m) \in [0,1]$
S1.3	Uniform	$\Omega^* = 2.5$	$(1-m) \in [0,1]$
S2.1	Bimodal	$\Omega^* = 1.5$	$(1-m) \in$ [0,1]
S2.2	Bimodal	$\Omega^* = 2.0$	$(1-m) \in [0,1]$
S2.3	Bimodal	$\Omega^* = 2.5$	$(1-m) \in [0,1]$

TABLE II
SIMULATION SETUPS

V. Evaluation of algorithms

The performance of above algorithms is evaluated by simulations. We will discuss our simulation setups first, and then the simulation results.

A. Simulation Setup

We simulate a replication system with 100 peers that are randomly linked. The connectivity of the network is controlled by a parameter $m \in [0, 1]$, termed *connectivity threshold*. Any two peers in the system are linked if a uniformly generated random number in $[0, 1]$ is greater than m. So the expected number of links would be $N(N-1)/2(1-m) = 4950(1-m)$. These links are logical, and the link delays and transmission costs are ignored in our model. We further name the parameter $(1-m)$ the *degree of connectivity* of the system.

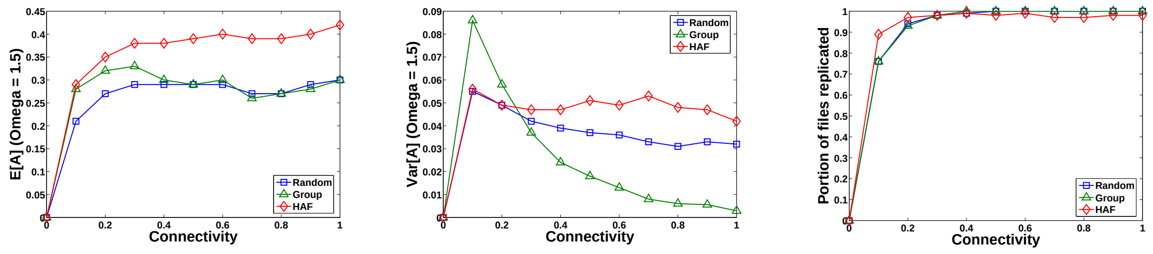
We are interested to find out how the algorithms perform under different system parameters. Two kinds of peer availability distributions are used in our simulations, namely, *uniform* availability distribution and *bimodal* availability distribution. The uniform distribution, in which each peer availability is uniformly distributed in $[0, 1]$, assumes the peers are very diverse in their online patterns. The bimodal distribution, in contrast, is used to model two distinct types of peers, some of them staying online for extended periods of time, while others being usually offline.

In all simulations, the number of files to be replicated in each peer is uniformly distributed in $[0, 100]$, hence the expected number of files in total is 5000 out of 100 peers' participation. The files are assumed to have similar sizes, if not exactly the same. In our case, the common file size is set to be 4 blocks (each block is of size Γ) before applying erasure code.

To study the effects of storage capacity, we run simulations with peers contributing different amount of storage space s_i for replication. We define the system-wide *average storage capacity per file* Ω^* as the ratio of the total storage space offered by all peers to the total size of all the files in the system:

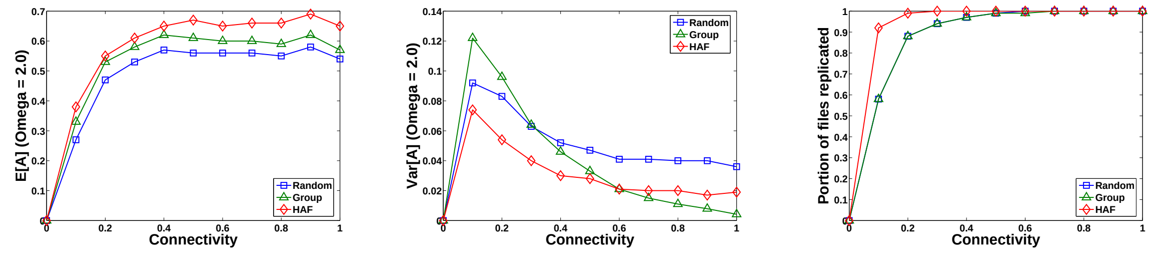
$$\Omega^* = \frac{\sum_{i \in \mathcal{P}} s_i}{\sum_{j \in \mathcal{F}} f_j}.$$

The exact storage space offered by each peer may not be the same. Hence, we set our model in the way that the storage offered by each peer is uniformly distributed in a certain range according to different expected value of Ω^*.



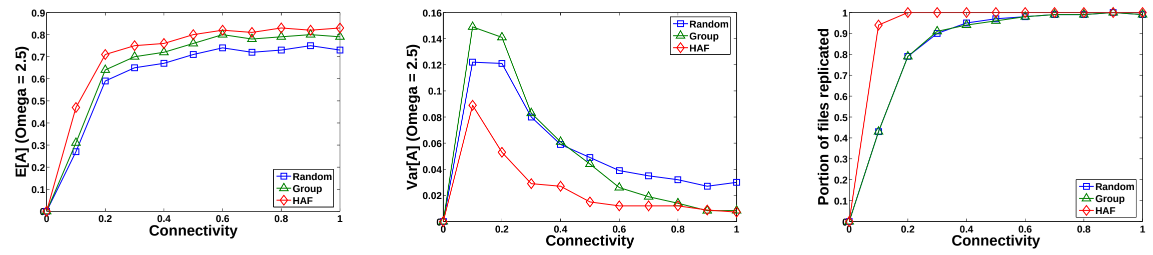
(a) (b) (c)

Fig. 1. Simulation results for $S1.1$



(a) (b) (c)

Fig. 2. Simulation results for $S1.2$



(a) (b) (c)

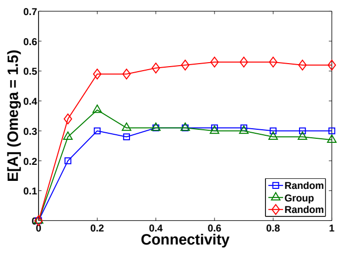
Fig. 3. Simulation results for $S1.3$

To minimize simulation errors due to random perturbations, each simulation is run 200 times, and the average results are reported. Table II summarizes the parameters of the simulation model.

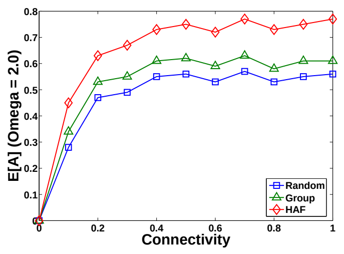
B. Simulation results

We first evaluate the performance of the three replication algorithms using uniform peer availability distribution. Figures 1 - 3 show (a) the expectation ($E[\mathbf{A}]$), (b) the variance ($var[\mathbf{A}]$) of the file availability distribution, and (c) the portion of files replicated, against the degree of connectivity $(1-m) \in [0,1]$ when the replication system is contributing different amount of storage space $\Omega^* = \{1.5, 2.0, 2.5\}$.

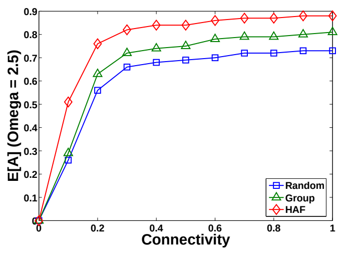
Three general patterns of behavior with these algorithms can be observed from the results, independent of the average storage capacity Ω^*. First, the increase in the degree of connectivity improves the algorithms' performance (in terms of $E[\mathbf{A}]$). This means that the more connected the peers are, hence the more global view of the replication system the peers have, the better the performance is. In particular, the performance increases sharply with degree of connectivity initially, indicating that peers need to have a minimum level of knowledge about other peers in the system (in our simulations, it is when the degree of connectivity $(1-m) > 0.1 \sim 0.2$) so that peers can find enough storage space to replicate files. When $(1-m) = 0.1$, each peer on average has $100 \times 0.1 = 10$ peers in its writable peer set. Since each file occupies at least 4 blocks of storage space, having less than 10 peers in the writable peer set means a peer may fail to find a sufficient number of other peers who have available storage (to replicate the given file). The sharp increases in $var[\mathbf{A}]$ and portion of files replicated (when $(1-m)$ increases from 0 to 0.1) further support this argument. Second, the group partition algorithm achieves lower $var[\mathbf{A}]$ than that of the random algorithm and HAF algorithm, especially when the storage space offered by peers is limited. This result validates our expectation that partitioning the peers in groups can replicate files in a fairer way. Third, in terms of the portion of files replicated, the HAF algorithm converges more quickly to 100% than other algorithms because it makes



(a)



(b)



(c)

Fig. 4. Simulation results for bimodal distribution

more elaborate scheduling when balancing the role of different available peers in the system. That is producing less redundancy using high available peers and more with lower available ones. However, all three algorithms perform similarly well when the number of connected peers in the system is large enough.

A very interesting observation is how the increase in the storage resources affects the algorithms' performance. When the average storage capacity is low ($\Omega^* = 1.5$) , the HAF algorithm outperforms the other two algorithms significantly in terms of the $E[\mathbf{A}]$ (by near 50%), as shown in Figure 1(a). However, when the average storage capacity increases, the performance difference gradually becomes insignificant, as shown in Figure 2(a) and Figure 3(a). When $\Omega^* = 2.5$, the performance difference decreases to only about $5\% \sim 15\%$, even though the HAF algorithm can still achieve the highest expected file availability.

Figure 4 shows the simulation results with bimodal peer availability distribution. When comparing the achieved performance for bimodal distribution versus uniform distribution, the expected file availability $E[\mathbf{A}]$ of the former is usually about $5\% \sim 15\%$ better than the latter. Hence we have a conjecture that if the two sets of peers have the same average peer availability but different variance, then the file replicated by the set of peers with higher variance will achieve higher expected file availability. In addition, for all three algorithms, $var[\mathbf{A}]$ is generally higher than that using uniform availability distribution. This is not surprising because a higher peer availability variance in bimodal availability distribution further increases the divergence of file availability. However, the group partition algorithm is still the most successful in minimizing $var[\mathbf{A}]$ for both distributions.

Furthermore, we also conduct extended simulations which show that the HAF algorithm is even more useful in improving file availability for the system with rare high available peers.

C. Discussion and future work

To summarize, the simulation results reveal the following findings. First, we see that HAF algorithm in general can achieve higher $E[\mathbf{A}]$, especially when peers share a small amount of storage space for replication and when high available peers in the system are rare. However, when peers increase their sharing, we see that the difference between the algorithms become insignificant. This observation has an important implication: when a replication system has high storage capability, the choice of replication algorithms does not make a lot of difference. In that case, it probably makes sense to choose a simpler algorithm, for example, the random algorithm. Second, the group partition algorithm can achieve lower variance in file availability, hence may be a good choice if fairness of file availability is important. Third, when comparing the simulation results between using uniform availability distribution and bimodal availability distribution, we observe that the increase in variance of peer availability tends to improve the expected file availability for all three algorithms.

There are still many interesting topics remained for further work. In this paper, we have assumed a *static* replication system: a fixed set of peers join the system and each of them replicates a fixed set of files. In a real life system, peers continuously join and leave; they may also remove old files and introduce new files. These dynamics bring many interesting possibilities. For example, if some peers leave permanently, how do we redistribute the file replicas they stored in order to maintain the file availability?

VI. Conclusion

In this paper, we address an important issue in P2P replication systems: resource allocation. We demonstrate the difficulty of resource allocation problem and formulate the optimization problem as an integer programming problem.

Since the problem is computation-intensive and needs to be solved in a decentralized manner, we propose three heuristic algorithms for peers to make replication decisions independently. In contrast to previous work, we consider heterogeneity in peer availability in our modeling, which is a more realistic assumption. The three algorithms differ in terms of the information required and computational complexity and their performance for different system parameters are evaluated using simulations. Our results show that the difference among the performance of different algorithms is insignificant when the storage resources are abundant. However, if the storage resources are scarce, we show that the HAF algorithm achieves the highest expected file availability. Meanwhile, the group partition algorithm is able to achieve a lower variance of file availability, which is beneficial if fairness of file availability is an important consideration. We also demonstrate how peer availability distribution affects the resultant file availability distribution, which has not been studied in previous work.

References

[1] “Akamai.” [Online]. http://www.akamai.com/

[2] J. Y. B. Lee and W. T. Leung, “Design and analysis of a fault-tolerant mechanism for a server-less video-on-demand system,” in *Proceedings of International Conference on Parallel and Distributed Systems*, 2002.

[3] E. K. Lee and C. A. Thekkath, “Petal: Distributed virtual disks,” in *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[4] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, “Oceanstore: An architecture for global-scale persistent storage,” in *Proceedings of ACM ASPLOS*, 2000.

[5] “Gnutella.” [Online]. http://www.gnutella.com/

[6] “WinMx.” [Online]. http://www.winmx.com/

[7] “Bittorrent.” [Online]. http://bitconjurer.org/BitTorrent/

[8] “Peercast.” [Online]. http://www.peercast.org/

[9] “Napster.” [Online]. http://www.napster.com/

[10] “Skype.” [Online]. http://www.skype.com/

[11] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge, “Incentives for sharing in peer-to-peer networks,” in *Proceedings of the 2001 ACM Conference on Electronic Commerce*, 2001.

[12] K. P. Eswaran, “Placement of records of a file and file allocation in a computer network,” in *Proceedings of IFIP Conference*, 1974.

[13] J. Kangasharju, J. Roberts, and K. W. Ross, “Object replication strategies in content distribution networks,” in *Proceedings of the 6th International Workshop on Web Content Caching and Distribution*, 2001.

[14] M. J. G. Bo Li, G. F. Italiano, X. Deng, and K. Sohraby, “On the optimal placement of web proxies in the Internet,” in *Proceedings of Infocom*, 1999.

[15] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, “On the placement of web server replicas,” in *Proceedings of Infocom*, 2001.

[16] V. Gopalakrishnan, B. Bhattacharjee, and P. Keleher, “Adaptive replication in peer-to-peer systems,” in *Proceedings of The 24th International Conference on Distributed Computing Systems*, 2004.

[17] B. -J. Ko and D. Rubenstein, “Distributed, self-stabilizing placement of replicated resources in emerging networks,” in *Proceedings of The 11th IEEE Conference on Network Protocols*, 2003.

[18] E. Cohen and S. Shenker, “Replication strategies in unstructured peer-to-peer networks,” in *Proceedings of ACM SIGCOMM*, 2002.

[19] S. Tewari and L. Kleinrock, “Proprotional replication in peer-to-peer networks,” in *Proceedings of Infocom*, 2006.

[20] K. Ranganathan, A. Iamnitchi, and I. Foster, “Improving data availability through dynamic model-Driven replication in large peer-to-peer communities,” in *Proceedings of The 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[21] F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen, “Autonomous replication for high avaibility in unstructured P2P systems,” in *Proceedings of The 22nd International Symposium on Reliable Distributed Systems*, 2003.

[22] W. K. Lin, D. M. Chiu, and Y. B. Lee, “Erasure code replication revisited,” in *Proceedings of The 4th International Conference on Peer-to-Peer Computing*, 2004.

[23] M. Ripeanu, “Peer to peer architecture case study: Gnutella network,” Technical Report, 2001.

[24] J. W. Mickens and B. D. Noble, “Predicting node availability in peer-to-peer networks,” in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2005.