A Resource Allocation Problem in Replicated P2P Storage Systems

Sriram Ramabhadran and Joseph Pasquale
Department of Computer Science & Engineering
University of California, San Diego
{sriram,pasquale}@cs.ucsd.edu

***Abstract*— This paper focuses on peer-to-peer storage systems that achieve availability through replication. We study the problem of resource allocation when the system must replicate multiple files using a fixed amount of resource. We characterize the optimal allocation that maximizes the average availability of the files in the system, and also study two simple, decentralized allocation schemes: Uniform and Proportional. We show that while Uniform is fair in terms of allocating resources, it may be arbitrarily sub-optimal. On the other hand, Proportional, though unfair in resource allocation, is competitive with the optimal allocation.**

I. Introduction

Replication is fundamental to building highly available distributed systems. To guard against unavoidable failures of system components, applications use redundancy, replicating functionality and state at multiple locations in the system. This is especially significant when the underlying system components are unreliable; good examples of this are peer-to-peer storage systems like PAST [9], CFS [5] and TRFS [2]. These systems use large-scale replication to build available storage on top of nodes that are highly unreliable. While these systems have addressed the question of how much redundancy is required to guarantee a certain level of availability, one important open question concerns *resource allocation*: How do constraints on available resources such as storage and bandwidth affect replication in such systems?

In this paper, we study a resource allocation problem that arises in the context of such P2P storage systems. where individual files are replicated at multiple peers for availability. When different files are competing for a limited amount of resources, how should the available resources be divided among the files? For example, in a co-operative P2P storage system, each peer contributes some amount of storage, which limits the total amount of storage in the system. Another example of a resource constraint is when there is a limit on the bandwidth available for long-term repair [3], which is required to guard against data loss due to the permanent departure of peers from the system. In both cases, the amount of resource consumed by a file is proportional to both its size, and the number of replicas. Given a constraint on the total amount of resource available, the problem is to determine how many replicas of each file to create.

We believe resource allocation is important for the following reasons. First, resources may be scarce; a peer-to-peer environment consists of autonomous peers *voluntarily* donating resources to the system. Second, there may be different demands on the system; for example, some peers may wish to replicate large files where as others may have smaller files. Finally, peers are *competing* for the available resources; in general, a peer is interested in maximizing the availability of its own files, while the system may try to optimize some overall notion of availability. In this paper, we assume that the system attempts to maximize the *average* availability (and not not minimum, as in prior work [1]) of the files. In a cooperative environment, where the files belong to the same user, minimum availability perhaps is a better metric. However, in the context of a system shared by multiple users, it is desirable that a single user does not impose an undue burden on the system by, for example, storing much larger files. Optimizing average availability, as opposed to minimum, allows the system the flexibility to give differentiated levels of service to users who make different demands on it. This naturally leads to an inherent tradeoff between *efficiency* and *fairness*. A fair division of resources among multiple files or users may not be optimal in terms of maximizing the availability of the system, i.e., it is inefficient; conversely, an optimal allocation of resources may be quite unfair. Finally, allocation schemes should be decentralized and assume minimal global knowledge.

II. The Replica Allocation Problem

A. Problem Description

There are k files that need to be replicated for availability; file i has a size of b_i bytes. The system consists of some number of nodes on which files can be replicated. Nodes have an unavailability of p , i.e., on average, a node is likely to be offline a fraction p of the time. The *replica allocation* problem is to determine how many replicas of each file to create subject to a resource constraint. Formally, let x_i denote the number of replicas of file i. The unavailability of file i, $q_i = p^{x_i}$, therefore, the average unavailability of the k files is $q = \frac{1}{k}\sum_{i=1}^{k} q_i = \frac{1}{k}\sum_{i=1}^{k} p^{x_i}$ [1]. We now model the resource constraint by assuming that the amount of resource consumed by file i is proportional to $b_i\, x_i$. This models the case where the system is constrained by storage or by long-term maintainance bandwidth [2]. Let the total amount of resource in the system be c (henceforth referred to as the *capacity* of the system), which is to be allocated among the k files. Therefore we have the constraint $\sum_{i=1}^{k} b_i\, x_i \leq c$. The goal is to minimize the average unavailability subject to the above constraint. We define the *optimal* allocation $(x_i^* \ldots x_k^*)$ as the one that minimizes q Replica allocation boils down to how to divide the available capacity among files of different sizes.

The above formulation makes several assumptions that simplify a more general allocation problem. First, we assume nodes are *homogeneous* in that they all have the same availability p. Relaxing this assumption implies that our notion of capacity must somehow incorporate availability as well; for example, one unit of storage at a highly available node is not the same as one unit of storage at a node that is less available. Second, we impose a constraint on the *aggregate capacity*; in general, replication may be constrained by capacity constraints at each individual node. Finally, we consider only a simplistic *user model*, which essentially assumes that there is a 1-1 correspondence between a user and a file, and that all files are equally important [3]. Realistically, there may be many users, each of which owns multiple files. In this case, average availability of the files may no longer be a desirable metric, and secondly, notions of fairness must be on a per-user basis (as opposed to a per-file basis). In this paper, we study only the simplified problem as a first step towards understanding the more general problem.

When all files have the same size, the optimal solution is simply to create an equal number of replicas for all files, as determined by the capacity. When files have different sizes, it may be possible to optimize by creating more replicas of smaller files at the expense of having fewer replicas of larger files. However, each additional replica of an file yields diminishing returns; therefore it is not always useful to allot capacity to a smaller file, which implies a non-trivial optimal allocation exists. What can we say about an optimal allocation? First, a smaller file will have a larger number of replicas. Second, similar sized files will have roughly the same number of replicas. In particular, if two files have the same size, the corresponding number of replicas cannot differ by more than 1. Finally, when two files differ in size, the difference in the corresponding number of replicas is bounded by a quantity independent of the system capacity. Although we omit proofs for the sake of brevity, it is relatively straightforward to verify that these properties are true.

B. An example

To illustrate the problem, consider the simple example of a system of capacity 12. Suppose there are 3 files to be replicated with sizes 1, 1 and 4 respectively. Figure 1 shows the behavior of q as a function of p for three different allocations (2,2,2), (4,4,1) and (6,6,0) [4]. We note that $q_{(2,2,2)} = p^2$, $q_{(4,4,1)} = \frac{1}{3}\,(2p^4 + p)$ and $q_{(6,6,0)} = \frac{1}{3}\,(2p^6 + 1)$.

[1]To avoid notational clutter, we consider the equivalent problem of minimizing average unavailability

[2]The long-term maintainance bandwidth, i.e., the bandwidth used in creating new replicas to replace replicas lost due to permanent departure of nodes from the system, is proportional to both the size of the fi le and the number of its replicas.

[3]It may be desirable to associate each fi le with a weight that models, for example, the popularity of each fi le. In this case, the metric to optimize would be the weighted availability.

[4]Although other allocations are possible, we omit those that are clearly sub-optimal.

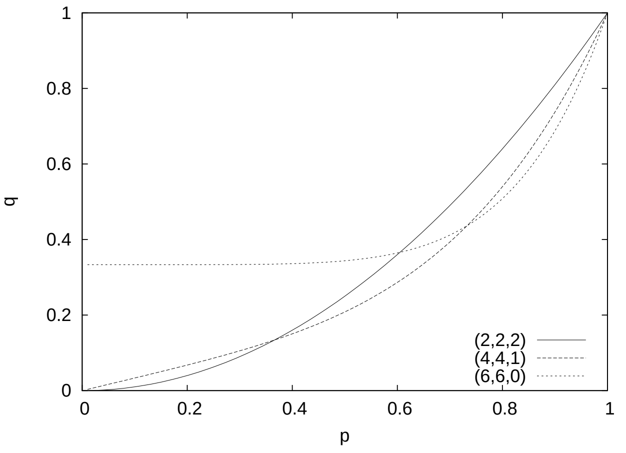


Fig. 1. Unavailability of different allocations with respect to p

For each allocation, q monotonically increases with p. At $p = 0$, i.e., when nodes are completely reliable, $q = 0$ for all allocations except $(6, 6, 0)$; in this case there are no replicas of one file, whose unavailability is therefore 1 irrespective of p. At $p = 1$, i.e., when nodes are completely unreliable, $q = 1$ for all allocations. As p varies from 0 to 1, the optimal allocation changes. For $p \leq 0.3660$, $(2, 2, 2)$ is the optimal allocation. For $0.3660 \leq p \leq 0.7329$, $(4, 4, 1)$ is the optimal allocation. Finally, for $p \geq 0.7329$, $(6, 6, 0)$ is the optimal allocation. We observe that, as p becomes larger, the optimal allocation becomes biased in favor of smaller files, i.e., smaller files have more replicas at the expense of larger ones.

C. Algorithms

Since file sizes are integral, dynamic programming [5] can be used to compute the optimal solution exactly. However, this is computationally intractable for large system sizes, and hence we develop a greedy approximation algorithm instead. Suppose file i currently has x_i replicas. The incremental benefit, i.e., increase in availability, of creating one more replica of file i is $p^{x_i} - p^{x_i+1} = p^{x_i}(1 - p)$ while the cost of doing so, i.e., capacity used, is b_i. The greedy algorithm (Greedy) picks the file with the maximum incremental benefit per unit cost, and creates another replica for that file. The algorithm terminates when the file that was picked cannot be replicated further due to insufficient capacity. Pseudocode for Greedy is shown below.

Algorithm 1: Greedy Algorithm

```
for $i = 1 \ldots k$ do
    $x_{i,G} = 0$
end
$c' = 0$
while true do
    $j = \arg\max_{1 \leq i \leq k} \left\{ \frac{p^{x_i}(1-p)}{b_i} \right\}$
    if $(c' + b_j \leq c)$ then
        $x_{j,G} = x_{j,G} + 1$
        $c' = c' + b_j$
    else
        break
    end
end
```

Let q_G denote the average unavailability of Greedy, and q^* that of the optimal allocation.

Theorem 1 (Greedy): $q_G < \frac{1}{p} q^*$

Proof Greedy terminates when $c' + b_j > c$, i.e., when an additional replica for file j cannot be created without violating the capacity constraint. Consider the allocation

$$x_i = \left\{ \begin{array}{ll} x_{i,G} & \text{if } i \neq j \\ x_{i,G} + \frac{c-c'}{b_j} & \text{if } i = j \end{array} \right\} \qquad (1)$$

This allocation is identical to the greedy allocation with the exception of file j which is given an additional fractional allocation of $\frac{c-c'}{b_j}$. Note that for all i, $x_i < x_{i,G} + 1$. The allocation $(x_1, \ldots, x_k)$ is optimal when the integrality constraints on the solution are relaxed. This is similar to the optimality of the greedy algorithm for the fractional knapsack problem [4] [6]. Now

$$q^* \geq \frac{1}{k}\sum_{i=1}^{k} p^{x_i} > \frac{1}{k}\sum_{i=1}^{k} p^{x_{i,G}+1} = \frac{p}{k}\sum_{i=1}^{k} p^{x_{i,G}} = p\, q_G$$

Therefore $q_G < \frac{1}{p} q^*$.

The above theorem shows that for large p, Greedy is close to optimal. When p is small, Greedy may be sub-optimal by an arbitrary amount [7]. However,

[5]A solution to the replica allocation problem satisfies optimal substructure. Let $Q_i(c)$ be the optimal solution using only the first i files and a capacity c. $Q_i(c)$ consists of assigning some capacity c' to the i^{th} file and $c - c'$ to the remaining $i - 1$ files (given by $Q_{i-1}(c - c')$); we take the minimum over c'.

[6]We note that the replica allocation problem is somewhat similar to the *unbounded knapsack* problem [4], with the important difference that each additional replica of a file gives diminishing returns in terms of availability.

[7]In fact, there are pathological problem instances when $q_G \geq \frac{q^*}{2p}$, which implies that the above bound is tight within a factor of 2.

in practice, for all file distributions we considered, Greedy may be considered as a close approximation of optimal.

III. Allocation Strategies

Let $b = \sum_{i=1}^{k} b_i$ be the sum of all file sizes. In the *Uniform* allocation policy, each file is allocated equal shares of the resource, i.e., each file is given an allocation of $\frac{c}{k}$. In this case, the number of replicas of file i is $x_{i,U} = \left\lfloor \frac{c}{k\, b_i} \right\rfloor$. We note Uniform is fair in terms of equitably allocating the total capacity among the files.

In the *Proportional* allocation policy, each file is allocated a share of the resource proportional to its size, i.e., file i is given an allocation of $\frac{b_i\, c}{b}$. In this case, the number of replicas of file i is $x_{i,P} = x_P = \left\lfloor \frac{c}{b} \right\rfloor$. We note that Proportional results in each file having the same number of replicas. This means that Proportional also minimizes the *maximum* unavailability as opposed to the average unavailability that we are considering.

We consider Uniform and Proportional for a couple of reasons. First, they are simple and intuitive from a resource allocation perspective. Second, they are amenable to a distributed implementation in which each file can compute its number of replicas with minimal global knowledge, and without knowing the distribution of file sizes. Estimates of c, the system capacity, and k, the number of files in the system, are sufficient to implement Uniform. Implementing Proportional requires an estimate of b, the total size of the files. It is relatively straightforward to propagate this information through the system, using gossip [7] for example.

A. Analytical Results

Theorem 2 shows that Uniform may be arbitrarily worse than the optimal allocation for any value of p. Let q_U denote the average unavailability of the Uniform allocation.

Theorem 2 (uniform): $\frac{q_U}{q^*} \rightarrow \infty$ as $c \rightarrow \infty$.
Proof Since $q^* \leq q_P$,

$$\frac{q_U}{q^*} \geq \frac{q_U}{q_P} = \frac{1}{p^{x_P}} \left(\frac{1}{k} \sum_{i=1}^{k} p^{x_{i,U}} \right) = \frac{1}{k} \sum_{i=1}^{k} p^{x_{i,U} - x_P}$$

Clearly,

$$c \left(\frac{1}{k\, b_i} - \frac{1}{b} \right) - 1 < x_{i,U} - x_P < c \left(\frac{1}{k\, b_i} - \frac{1}{b} \right) + 1$$

Therefore

$$\lim_{c \to \infty} x_{i,U} - x_P = \left\{ \begin{array}{ll} +\infty & \text{if } b_i < \frac{b}{k} \\ 0 & \text{if } b_i = \frac{b}{k} \\ -\infty & \text{if } b_i > \frac{b}{k} \end{array} \right\}$$

Since $0 < p < 1$,

$$\lim_{c \to \infty} p^{x_{i,U} - x_P} = \left\{ \begin{array}{ll} 0 & \text{if } b_i < \frac{b}{k} \\ 1 & \text{if } b_i = \frac{b}{k} \\ \infty & \text{if } b_i > \frac{b}{k} \end{array} \right\}$$

As the capacity of the system increases, the availability of an individual file depends on how small or large it is relative to the average file size $\frac{b}{k}$. If a file is small, i.e., less than average, its availabilty can be arbitrarily larger under Uniform than Proportional. Conversely, if an file is large, i.e., more than average, its availability can be arbitrarily larger under Proportional than Uniform. What happens to the average availability? There is at least one file j such that $b_j < \frac{b}{k}$ [8]. Now

$$\frac{q_U}{q^*} \geq \frac{1}{k} \sum_{i=1}^{k} p^{x_{i,U} - x_P} > \frac{1}{k}\, p^{x_{j,U} - x_P}$$

The right-hand side goes to infinity with increasing capacity, therefore so does the left-hand side, i.e., $\lim_{c \to \infty} \frac{q_U}{q_P} = \infty$.

The above theorem implies that, as the amount of resource available in the system increases, Uniform does arbitrarily worse than optimal. This is true for any value of p. Since Uniform is also fair, this implies that there is a conflict between fairness and efficiency: the fair policy can be arbitrarily inefficient. We now consider the Proportional allocation policy. Theorem 3 bounds the deviation of Proportional from the optimal allocation. Let q_P denote the average unavailability of the Proportional allocation.

Theorem 3 (proportional): $q_P < k\, q^*$
Proof There is at least one file j such that $x_j^* \leq x_P$. Otherwise $\sum_{i=1}^{k} b_i\, x_i > \frac{c}{b} \sum_{i=1}^{k} b_i = c$, which would violate the capacity constraint. Now

$$q^* = \frac{1}{k} \sum_{i=1}^{k} p^{x_i^*} > \frac{1}{k}\, p^{x_j^*} \geq \frac{1}{k}\, p^{\frac{c}{b}} = \frac{q_P}{k}$$

Therefore $q_P < k\, q^*$.

[8]This is true unless all files are of equal size, in which case both Uniform and Proportional are vacuously optimal.

The above theorem implies that for a given file distribution, i.e., constant k, Proportional is always within a constant factor of optimal, regardless of p or the distribution of file sizes. Thus as the amount of resource increases, Proportional is always within a factor of k of optimal, whereas Uniform may be arbitrarily worse. In addition, Theorem 3 implies that Proportional is competitive, with a competitive ratio of at most k [9]. Thus, irrespective of p, Proportional is always within a factor of k of the optimal. Moreover, when c is an exact multiple of b, Proportional is in fact the *only* competitive allocation. To see this, observe that, as in the proof of Theorem 3, there is at least one file j such that $x_i < x_P$. Now

$$\frac{q}{q_P} > \frac{1}{k} p^{x_j - x_P} > \frac{1}{kp}$$

As $p \longrightarrow 0$, the competitive ratio of any allocation becomes unbounded. This implies that in a system where p is not known and may be small, Proportional is guaranteed to be competitive. When c is not an exact multiple of b, there are other allocations that may be better. This is because rounding wastes capacity, which we address later in this section.

B. Simulation Results

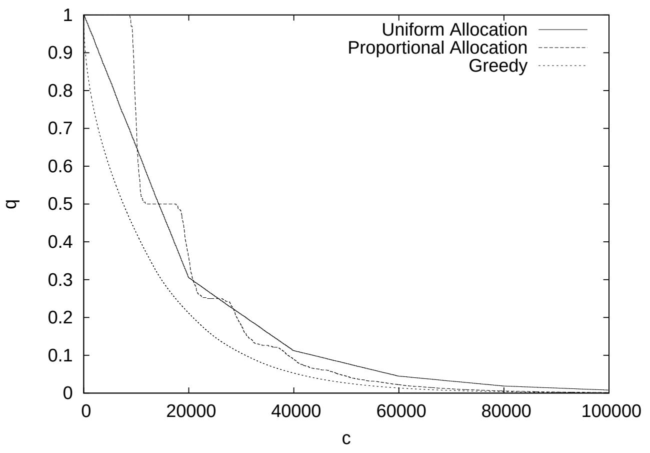
In this section, we simulate the Uniform and Proportional allocation strategies and compare it with Greedy. Greedy is used as an approximation of the optimal allocation as the optimal is computationally intractable to calculate for large system sizes. We use two input file distributions: uniform and zipf. In both cases, there are $k = 100$ files with mean file size 100. In each experiment, we average over 100 iterations to eliminate the effect of a specific set of file sizes.

Figure 2(a) shows q as a function of system capacity c when file sizes are uniformly distributed. Figure 2(b) shows the same when the file sizes are zipf. The value of p is 0.5 . First, as expected, q decreases with increasing c; greater capacity implies more replication, which results in increased availability. Second, Greedy results in better availability than both Uniform and Proportional; the difference is more pronounced in the case of zipf distributed file sizes. Finally, for uniform file distributions, Proportional exhibits a staircase behavior. This is because Proportional uses rounding to compute the number of replicas. For $k = 100$ files of mean size 100, the expected value of the total size b is 10000. For every additional 10000 of capacity, the number of replicas of each file x_P increases by 1; recall that Proportional allots the same number of replicas for all files. Due to rounding, the number of replicas (and hence q) increases discontinuously with increasing system capacity. This effect can also be seen for Uniform, although to a lesser extent. This is because in the case of Uniform, the replication level does not increase uniformly for all files; as capacity increases, smaller files will increase their replication level before larger files. However, the rounding effect is still present; thus the curve for Uniform appears piecewise linear. In the case of zipf distributed files, this effect is much less pronounced; the larger variance of b (recall we average over 100 iterations) mitigates this effect to some extent.

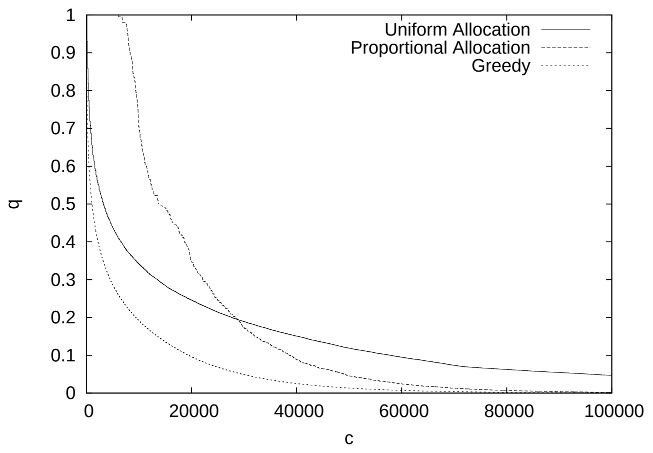
The above discussion highlights some inefficiency in both Uniform and Proportional. Because a fractional number of replicas must be rounded down to satisfy the capacity constraint, both allocations do not fully utilize the available capacity. Thus the sub-optimality of these schemes could be due to both the incomplete utilization of capacity as well as some intrinsic inefficiency in the allocation. We can eliminate the former through a simple randomized scheme which makes the *expected* value of the number of replicas is equal to its allocation. For example, if a file is to be allotted 6.6 replicas, the file is allotted 7 replicas with probability 0.6, and 6 with probability 1 - 0.6 = 0.4.

Figure 2(c) shows the behavior of the randomized versions of Proportional and Uniform for a uniform file distribution. Figure 2(d) shows the same for a zipf file distribution. As can be seen, randomization enables more efficient use of marginal increases in capacity; the curves are therefore smoother. Also for a given system capacity, the randomized variants have lower q. In the case of the uniform file distribution, both Uniform and Proportional are close to Greedy. In the case of the zipf file distributions, they are less so, especially for small capacities. In both cases, Uniform is better than Proportional for small system capacities; for larger capacities, Proportional

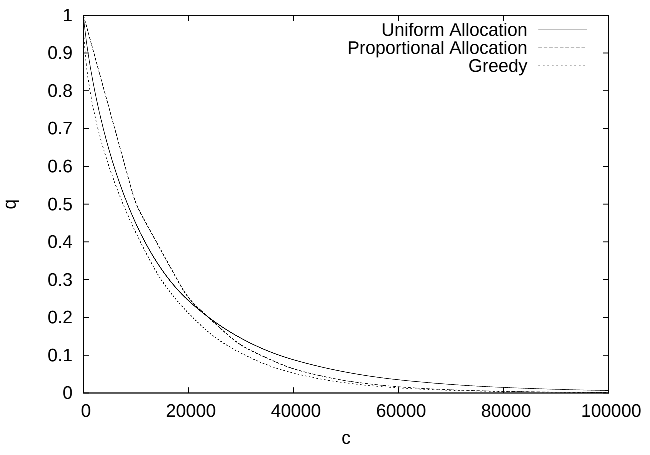
[9]The competitive ratio of an allocation is a bound on how much worse an allocation can do with respect to the optimal allocation, *for any value of p*. Note that the optimal allocation itself will depend on the value of p. In the example of Section II-B, the competitive ratio of $(2, 2, 2)$ can be computed to be 1.25 using simple calculus.



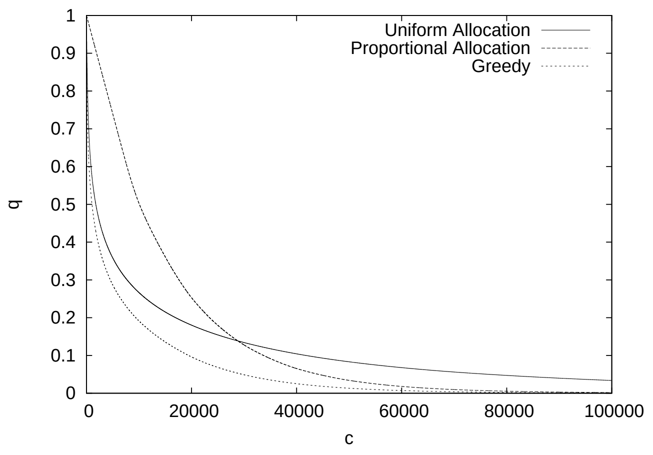
(a) Uniform file distribution



(b) Zipf file distribution



(c) Uniform file distribution



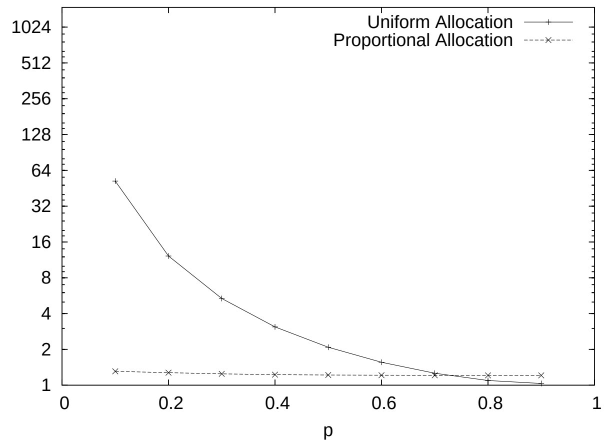
(d) Zipf file distribution

Fig. 2. Unavailability of Uniform and Proportional and randomized variants

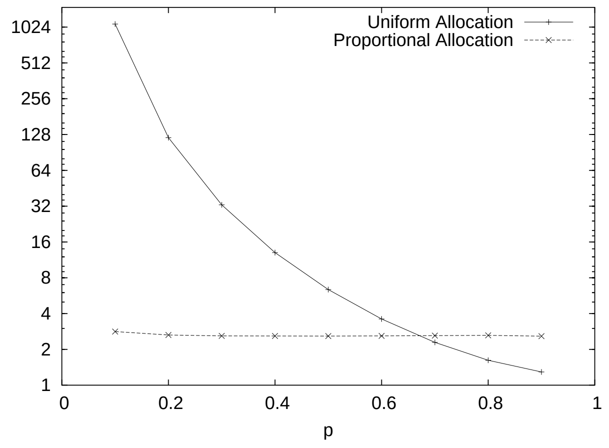
is better. This is because Uniform results in smaller files being replicated independant of larger files, which is important for smaller capacities when there is insufficient capacity for larger files to be replicated. Since Uniform divides the capacity of the system fairly among files, a large file does not affect the availability of a small one when there is insufficient capacity.

Figure 3 shows the behavior of Greedy for uniform and zipf file distributions, as a function of p. The value of c is 50000. As expected, q increases with increasing p. When p is small, the file distribution does not affect the optimal solution to a great extent. This is because the optimal allocation tends to be more uniform when the nodes are relatively reliable. When p is large, i.e., nodes are more unreliable, Greedy does better in the case of the zipf distribution due to the preponderance of small files. One replica less of a large file allows several smaller files to be replicated more. Since a zipf distribution results in a large number of small files and a smaller number of large files, replicating smaller files more is more efficient. When file sizes are more uniform, this is not possible.

Figure 4(a) shows how Uniform and Proportional perform with respect to Greedy as a function of p. In the case of uniform file distributions. Specifically, the y-axis shows the ratio of Uniform (Proportional) to the Greedy allocation. Figure 4(b) shows the same for the zipf file distribution. First, we note that Proportional is a constant factor away from Greedy for all values of p; the constant is about 1.2 in the case of uniform and 2.5 in the case of zipf. Theorem 3 shows that Proportional is always within



(a) Uniform file distribution



(b) Zipf file distribution

Fig. 4. Unavailability of Uniform and Proportional allocations compared to Greedy

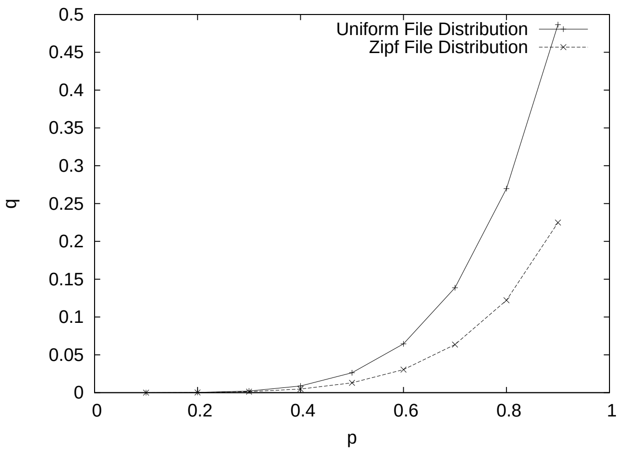


Fig. 3. Unavailability of Greedy allocation

a factor of k of optimal; the simulations indicate that the actual number is much smaller. Second, Uniform is significantly sub-optimal for small values of p, but improves with increasing p. For large values of p, it is actually better than Proportional. This reflects the fact that when p is small, i.e. nodes are relatively reliable, the marginal increase in availability due to an additional replica is small. Therefore, favoring smaller files at the expense of larger files is sub-optimal. As p becomes larger, the converse is true: it is better to sacrifice large files to ensure that smaller ones are available. We note that Theorem 2 implies that for any value of p, Uniform will eventually be worse than Proportional as system capacity increases.

IV. Discussion

We have examined a resource allocation problem that arises in the context of replicated P2P storage systems. We have considered some simple algorithms that allocate resource among files in a distributed storage system, and the efficiency/fairness properties of the resulting allocations. Our model is simplistic; further work is necessary to refine it and study the problem in a more realistic setting. We briefly discuss possible directions.

When nodes are heterogeneous, a more sophisticated resource model is necesary to quantify the capacity contributed by each node. Rather than using raw capacity, a better is use to define some notion of *available capacity*, which incorporates the availability of a node as well. Our initial intuition suggests that the product of the raw capacity and the negative log of a node's unavailability may be a good candidate for this. Secondly, a more sophisticated user model will allow many users, each of which have multiple files to store. In this case, computing average availability on a per-user basis, and then taking an average over all users, may be a reasonable metric to optimize on a system-wide basis.

With more complex resource and usage models, the questions are: 1. How does the system balance efficiency with fairness, and 2. How does the system achieve desired allocations? In the latter case, the seminal work on replica placement in FARSITE [6] suggests that a disributed hill-climbing approach may be viable. Finally, when nodes in the system are controlled by users, as is likely in a P2P scenario, there is the question of how the system deals with strategic behavior on the part of users. Thus, an open question would be is it possible to design mechanisms that incentivize achieving the desired allocation.

V. Conclusion

In this paper, we have formulated and studied the replica allocation problem, which deals with optimizing the availability of files in a distributed storage system subject to a resource constraint. Our work is related to recent work in building highly available storage systems based on the peer-to-peer paradigm. CFS [5] and PAST [9] use conventional replication while Oceanstore [8] and TRFS [2] use erasure coding. To the best of our knowledge, these systems do not consider optimizing the number of file replicas based on a resource constraint.

Our conclusions may be summarized as follows. When peers in the system are relatively reliable, the optimal allocation does not favor smaller files over larger files. When peers are unreliable, the converse is true , i.e., smaller files tend to have more replicas. Uniform, though fair in terms of dividing the available resource, is sub-optimal, and in fact, may be arbitrarily worse than optimal. On the other hand, Proportional, though unfair in terms of resource allocation, is always within a constant factor of the optimal allocation, which in practice tends to be quite small.

References

[1] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *In Proceedings of USENIX OSDI*, 2002.

[2] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. Total recall: System support for automated availability management. *In Proceedings of USENIX NSDI*, 2004.

[3] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. *In Proceedings of HotOS IX*, 2003.

[4] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[5] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. *In Proceedings of ACM SOSP*, 2001.

[6] J. Douceur and R. Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed fi le system. *Proceedings of DISC 2001*.

[7] M. Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinsk, 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.

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