Availability in Global Peer-To-Peer Storage Systems

Thomas J. E. Schwarz, S. J.
Computer Engineering Dept.
Santa Clara University

Qin Xin Ethan L. Miller
Storage Systems Research Center
University of California, Santa Cruz

Keywords: Availability, Peer-to-Peer Systems, Hill-Climbing

Abstract

The recent popularity of peer-to-peer (P2P) systems for file sharing has led to increased demand for higher data availability from such systems. Measurements of popular P2P systems indicate that a large number of hosts appear to be available on a cyclic basis, and their availability shows great heterogeneity. This paper proposes a cooperative storage technique which employs erasure coding schemes on a collection of data objects and provides various levels of data redundancy. Based on this technique, we propose a history-based hill climbing scheme that takes advantage of varied time zones in a global system, and compare it to more conventional approaches to providing data availability. Our simulation results show the improved data availability by this scheme. We also investigate several climbing strategies including choice of coding schemes and laziness of data movement.

1. Introduction

The past few years have seen peer-to-peer (P2P) systems increase in popularity. Unlike traditional client-server systems which essentially rely on a few servers with high reliability and availability, P2P systems decentralize the management of data storage and harness unused resources on computers distributed across the world. Within a P2P system, failure of a few nodes *may* not result in a sharp decrease in data availability because all nodes are peers and the remaining peers may provide or reproduce the required data. However, nodes in P2P systems are not always available. The variation in properties and the complexities in distributions of the millions of individual nodes in a P2P system provide a great potential to improve availability.

This paper proposes a hill-climbing scheme based on a cooperative storage technique to boost data availability in world-wide peer-to-peer storage systems. Recently, a few models have been proposed focusing on availability enhancement in P2P systems [3]. OceanStore [7] uses P2P techniques to store data reliably. By supporting nomadic data, this system can provide a highly available storage utility, in which any computer can join the infrastructure at any time. In exchange for

economic compensation, participants in OceanStore contribute remote storage or provide local user access. Since nodes cannot be individually trusted, each data object is broken into n fragments by using an erasure coding scheme so that m out of n fragments suffice to reconstruct the object. Each data object is versioned and spread over hundreds or thousands of servers randomly in read-only form. To keep track of the sites where each object lives causes much system overhead. We propose a variation of this scheme based on redundancy groups [13]. A *redundancy group* is a collection of data objects. By using erasure coding schemes, a redundancy group is broken into m fragments in a manner that the original n data objects in the group are broken as well. This is done together with the *parity fragments*, which provide fault tolerance. Such a method is called an m/n coding scheme and denoted as "m out of n." We use a deterministic data distribution algorithm such as RUSH [6] to map data objects into redundancy groups and to allocate redundancy groups to the nodes in a P2P system. Given an identifier of each redundancy group, RUSH provides a list of nodes for each redundancy group so that we can locate the nodes and access data very quickly. We further extend the list to include the potential sites where fragments in the redundancy group can be moved to later on. With redundancy groups and the smart data allocation scheme, data is stored in a cooperative way among the nodes in a P2P system. Also, more importantly, we support efficient data movement via the extended node lists, therefore, the desired adjustment for data storage can be done without expensive cost of system overhead.

Data availability can be improved by adjusting data locations because of the great heterogeneity among nodes in a P2P system. Studies of P2P systems [10] show that individual nodes tend to be quite unreliable. In a storage system such as OceanStore, this might not be the case, since participation would be based on an explicit contract. In contrast, participants in a system like Napster view their participation as casual. Douceur [4] postulates that node availability is cyclic. Bhagwan, *et al.* [2] also pointed out that availability in P2P systems can be modeled by a combination of two distributions: daily join/departure and long-term behavior. We distinguish between two type of nodes: server-type nodes that tend to be always accessible, and cyclic type nodes that are not. The latter nodes reflect the established Internet habits of their owners that depend on the time and day of the week. Nodes can go offline for other reasons; for example, an owner might take a vacation, during which the node is probably not available. Thus, we cannot view the unavailability of these nodes as completely cyclic.

We distinguish between data reliability (the probability that data is not irretrievably lost) and data availability (the probability that data is not accessible). Data reliability depends on the long-term behavior of a node, whereas data availability is mostly determined by the current behavior of a node. A node that is up for small periods of time contributes much to data reliability but little to data availability. Because of the cyclic behavior of peers in a global P2P system, a node that is up at one time might be down at other time; thus, data availability depends on the status of the nodes on which data is stored.

In this paper, we focus on availability, rather than reliability. The central idea is to improve availability by selecting the right combination of storage sites for a redundancy group. For example, we assume a 32 out of 64 scheme is used and our data placement algorithm [6] gives us more than 64 sites, say 75 candidates for placement, but we only use 64 of these sites. If we find a number of cyclic nodes among those 75, they are likely to be used; otherwise, there is an incentive to turn off computers, because their owners get the economic benefit of participation in the scheme without the cost of contributing storage. However, if 15 nodes are cyclic nodes located in California, for an example, all of these 15 nodes will be likely to be off during the night. It would be much better to trade the cyclic nodes in California for several cyclic nodes among the 75 or

a few cyclic nodes in other time zones, such as New York, Lausanne and Tokyo, to distribute the downtime. By judiciously selecting nodes, we can greatly improve data availability.

2. P2P Storage

A global P2P system is very dynamic: at any given moment, one node may join or leave the system. Data is viewed as nomadic, and redundancy is necessary to provide fault tolerance and highly availability of data access. In order to locate data quickly and decrease system overhead in book-keeping, we propose to collect a set of data objects as a group, and configure the group by adding replicas or parity information. We define such a collection of data as a *redundancy group*. In this section, we first have a brief look at characteristics of data storage in peer-to-peer systems, and then describe the construction process and varied configurations of a redundancy group. We also discuss how we support adaptive data placement in a peer-to-peer system.

2.1. Availability Characteristics in Global Peer-to-Peer Systems

One prominent attribute of P2P systems is that peers are symmetric: they are able to function as both a server and a client. This distinguishes P2P systems from traditional server-client systems. Measurement of two popular P2P systems, Gnutella and Napster, indicates that a great amount of heterogeneity exists among individual peers in various aspects: number of files shared, bandwidth, latency, and availability [10]. Surprisingly, very few peers in both systems fall in the high-availability category. The study of node availability in Overnet, another P2P file sharing application, by Bhagwan *et al.*, arrived at similar conclusions with respect to availability.. They proposed that availability in P2P systems can be modeled by a combination of two distributions: daily join/departure and long-term behavior. Douceur [4] also suggested that hosts exhibit availability behavior from each of the two classes: always-on and cyclic on/off; so he modeled host availability by a gradual mix of two distributions.

Sites in a P2P system are usually distributed across the world. Their locations in various time zones influence their cyclic behavior. For example, two cyclic nodes that are down during the night, but located in Los Angeles and Bern, show complimentary availability due to the clock difference. We can take advantage of this kind of behavior to improve availability.

2.2. Redundancy Groups

A global P2P storage system contains hundreds of terabytes to petabytes of data objects. For each object, we assign a unique Object Identifier (OID) for it in the system; this identifier can be either assigned randomly or perhaps calculated by the MD5 or SHA1 hash. Each object belongs to a unique redundancy group with its own unique group identifier (GID). We assume that data objects have approximately the same size. If necessary, data objects can be subdivided and / or padded with zeroes. The best performance can be achieved if small data objects are accessed from a single process of data transfer, but large data objects must be transferred from multiple sources.

In light of the dynamic nature of P2P systems, new objects and groups are created, and some objects might change their group membership. Similarly, as peers enter and exit the system, some nodes will transfer data objects dynamically from

other nodes, either to balance the load, or to take over from a failed or leaving node. The details are beyond the scope of this paper, but can be found in other papers on P2P systems [8, 11].

To guarantee high data availability, we use an m/n erasure coding mechanism to configure redundancy groups. Each redundancy group is split into n fragments so that m fragments suffice to recalculate the contents of the group. To save download time, we allow the data itself to be a fragment and thus accessible in a single transfer. We use systematic Erasure Correcting Codes (ECC), such as generalized Reed-Solomon codes, Tornado codes, etc. These codes take the data and generate additional parity fragments. Some codes, such as generalized Reed-Solomon, create parity fragments of the same size as data fragments, while others will create parity fragments that are slightly larger.

Given a group identifier (GID), we use a placement algorithm such as RUSH [6] to provide a number $N > n$ of possible peers for storage. We call the N nodes, identified by the placement algorithm, the *candidate list*. When a redundancy group is created, we store its fragments on the first n servers in the candidate list, and then use measured uptime as the system is used to find n among the N servers that can give the best availability. The candidate list provides the possible locations of data objects for further adjustment of data placement with the aim to achieve high availability.

To access a data object, we first determine the redundancy group. This depends on the state of the system, but a local address cache will typically determine the group correctly. If not, the system will find the correct group by the placement algorithm. Next, the client accesses a node in this group. All nodes in a redundancy group have complete location information for the group. This protects against negative effects from large values of n. The request will then be directed to the correct node, which hopefully fulfills the request. By caching the return address, the owner will have a good chance to retrieve the object directly from the storage node at the next time.

If the access did not succeed, then the client asks all the nodes in the group to send the fragments to it. If the client receives m or more fragments, then the client can reconstruct the data object, that is, the data object is available. Otherwise, the object is unavailable, and only further probing will show whether the unavailability is temporary or not.

The amount of metadata stored in each node depends on the number of groups on the node. Since nodes carrying fragments in the same group need to monitor each other, each node should only carry fragments from a few groups. Of course, the load at each site will be well balanced with many groups statistically. We estimate that 10 GB total storage at each site and 10 fragments of 1 GB each are reasonable choices. Each node then would have $(10N - 10)$ other sites to be concerned about; this does not constitute significant overhead.

3. Improving Availability By Hill-Climbing Algorithms

The storage technique based on redundancy groups provides an efficient way to store data objects in a cooperative environment. With a list of nodes indicating the current and possible locations, the process of locating data object becomes fast, and more attractively, we are able to adjust data locations. Thus we achieve superior data availability in a peer-to-peer system even though hosts are frequently down or offline.

We place data objects within a redundancy group g on peers $s_{g,i}$, for $i = 1 \ldots N$ and select n peers among them that yield the best availability. One among these n peers is randomly elected as a *leader* whose task is to optimize availability. If a leader is unavailable for a long time, it will be replaced. The leader uses statistical data on the availability of its "buddies"—the other

nodes carrying objects from the redundancy group. Based on its historical knowledge, the leader then selects the buddies that can improve the availability of the group.

Ideally, we should evaluate every possible set of n buddies, but $\binom{n}{N}$ is typically far too large to proceed efficiently; this problem is NP-complete. Instead, we have developed heuristics that approach the optimal solution.

We propose a data placement adjustment mechanism that considers the cyclic behavior of nodes in a P2P system. Since it can steadily improve data availability, we put it in the category of *hill climbing* algorithms. FARSITE [5] uses a random hill-climbing method, which differs with our mechanism in that history of node availability is not considered in the random climbing. We call our mechanism **H**istory **B**ased **H**ill **C**limbing algorithm (HBHC). We will discuss how it works in detail in this section and show the evaluation results in the next section, with a comparison against the random hill climbing algorithm.

3.1. History Based Hill Climbing (HBHC)

History Based Hill Climbing (HBHC) considers cyclic behaviors of nodes in P2P storage systems. It gathers statistics on nodes' up-times and uses a simple optimization to select those nodes that historically have shown the best availability. The availability of each node is represented by up-time scores. The score of each node is set up to be zero at system initialization, and it is increased by one if the node is available during a periodical scan, and decreased by one if the node is down. Note that we view a node to be available/up only when it is online. Members in one redundancy group keep a record that lists the up-times scores of all the buddies in the group and disseminate this record among the nodes within the redundancy group in an epidemic way. After a period of collecting statistics, our algorithm begins to run. The node with the lowest uptime score will be swapped with the node selected from the list of candidate locations (which differs from the locations of the current buddies) of redundancy groups if the node has a higher score. This climbing process can be very aggressive: the locations of buddies are adjusted as long as the redundancy group is available, and a number of buddies in a group can be adjusted in parallel. However, aggressive climbing causes too much system overhead. We propose a lazy version of hill climbing by setting up a threshold number T. Only if the number of available peers in a redundancy group reaches or falls below T, would the climbing process be carried out.

The pseudo-code of HBHC is shown in Figure 1(a). Suppose each redundancy group is configured by m out of n erasure coding scheme, in other words, data is accessible when any m nodes are available among n nodes where data with its parities are stored, and access to a data is regarded as failed if less than m nodes are up at a given time. Note that hill climbing can be started only when data is accessible. We label the lazy option by a flag *lazy_flag*, indicating whether the climbing is active whenever data is accessible, or is only active when the number of unavailable nodes reaches a threshold number T. Our data placement algorithm, RUSH [6], gives the current locations where data is stored, with a list of candidates where data can be moved to later. For one redundancy group G, HBHC selects a node currently in the redundancy group whose up-time score is the lowest, say node A. It then chooses a node with a higher score from the candidate list, say node B, and moves the data chunk in the group G from node A to node B. Node B must be available during the moving period, but node A can be down at that time since the data chunk can be regenerated from other members in group G by the erasure coding scheme, but at the cost of reading multiple nodes.

```
int HBHC (int m, int n, int lazy_flag, int T)
{
peers_unavail[ ] ← node IDs of current unavailable peers
num_unavail ← the number of current unavailable peers
if num_unavail > m
  return 0
else
  if (lazy_flag == LAZY)&&(num_unavail >= T)
    for i ← T to num_unavail
      new ← a random available candidate
      new ↔ peers_unavail[i]
  if lazy_flag == NON-LAZY
    for i ← 0 to num_unavail
      new ← a random available candidate
      new ↔ peers_unavail[i]
  return 1
}
```

(a) history based hill climbing scheme

```
int RANDOM (int m, int n, int lazy_flag, int T)
{
peers_unavail[ ] ← node IDs of current unavailable peers
num_unavail ← the number of current unavailable peers
if num_unavail > m
  return 0
else
  if (lazy_flag == LAZY)&&(num_unavail >= T)
    for i ← T to num_unavail
      new ← an available candidate with higher score
      new ↔ peers_unavail[i]
  if lazy_flag == NON-LAZY
    for i ← 0 to num_unavail
      new ← an available candidate with higher score
      new ↔ peers_unavail[i]
  return 1
}
```

(b) random hill climbing scheme

Figure 1. Pseudo-code of two kinds of hill climbing algorithms, by using m out of n configuration, lazy option labeled by $lazy\_flag$ with threshold T.

3.2. Random Hill Climbing

Random hill climbing, on the other hand, does not keep track of the uptime records of the nodes in a redundancy group. Instead, it randomly chooses a down node among the current peers within the group and replaces it with randomly-chosen available node from the candidate list, as shown in Figure 1(b). The process is similar to hill climbing data placement in the FARSITE system [5]. In contrast to HBHC, this scheme avoids collecting statistics, but does not prevent instances of thrashing in which ill-suited nodes are selected.

4. Simulation Evaluation

Our simulation evaluation is based on a simple model of a global P2P storage system. The model considers various worldwide time zones and cyclic behavior among peers in this system. We explore the improvement of data availability by the history based hill climbing algorithm and compare it with the random hill climbing policy. System availability is measured in units of "nines" [5], defined as $-log_{10}(1-P)$, where P is the fraction of the time when data objects in a system are available. For instance, an availability of 4 nines implies that data objects are accessible during 99.99% of the time.

Figure 2. Node locations in a global P2P storage system.

4.1. System Assumptions

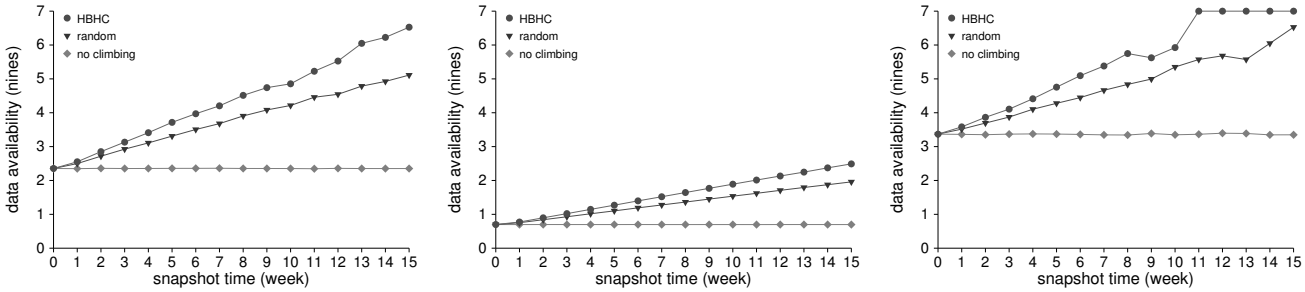
The simple model of a global P2P storage system is shown in Figure 2. We assume 10^6 peer sites in our P2P system are evenly distributed in the nine timezones across the world: Los Angeles (GMT-8) (GMTn is calculated by adding n hours to Greenwich Mean Time), Chicago (GMT-6), New York (GMT-5), London (GMT), Lausanne (GMT+1), Moscow (GMT+3), Hongkong (GMT+8), Tokyo (GMT+9), and Sydney (GMT+10). There is no central controller in such a system, and each peer plays dual roles—server and client. Files are stored in a cooperative fashion as each redundancy group is broken into fragments and placed across a number of peers. However, the group membership is rather loose: any peer can join or leave a redundancy group at any given time.

It is assumed that peers have two classes of availability behavior [2, 4]: always on and cyclically on/off. The ratio of cyclic-type to always-on type peers depends on the particular P2P system. Douceur [4] reported that about 60% of the hosts in Napster and Gnutella system had less than 70% availability (0.5 nines), due mainly to the cyclic behavior. We call the ratio of cyclic to always-on nodes the *cyclic ratio*. In our simulations, we configured the cyclic ratio to be 50% and it can be altered to simulate availability behavior of various types of P2P system. Cyclic behavior is further classified into two categories: daily-off and weekend-off. Daily-off behavior includes daytime off and nighttime off and takes three quarters in the cyclic-type nodes, while the remaining one quarter of the nodes appears to be weekend-off, either daytime and weekend off or night time and weekend off. A finer partition of cyclic behavior will be closer to real systems, so we consider using real system traces in our future work.

4.2. Simulation Results

We explored system availability improvement from history based hill climbing (HBHC) and compared it with random hill climbing (RANDOM) and no-climbing.

We use three variants of erasure coding schemes to configure redundancy groups: 16 out of 24, 16 out of 32, and 32 out of 64. Each redundancy group is stored across 24, 32, and 64 peers in each scheme, respectively. With 1 GB client data in a redundancy group, the size of fragments on each peer is 64 MB for the 16/24 and 16/32 schemes, and 32 MB for 32/64. Total client data capacity is 200 TB. We measured system availability by randomly sampling a certain number of redundancy groups hourly. The maximum tolerable number of unavailable peers for m out of n schemes is $(n-m)$. When the number of



(a) coding scheme = 16/32/32. (b) coding scheme = 16/24/32. (c) coding scheme = 32/64/32.

Figure 3. Availability Improvement of HBHC and RANDOM

unavailable peers in a redundancy group is greater than $n - m$, the group is unavailable and hill climbing can not be carried out. We simulated the system for 15 weeks. As we will see, data availability reaches 5–6 nines at the end of 15 weeks when a proper erasure coding scheme, such as 16 out of 32, is used. There is little room to improve availability beyond this point. If the erasure coding is less aggressive, say, 16 out 24, the steady improvement by hill climbing algorithm will continue. We assume HBHC starts after one week of statistics collection. We define the number of nodes on which data can be moved onto in each redundancy group as *the length of candidate list*, and we extend the configuration parameters of the erasure coding scheme as $m/n/k$, where k is the length of the candidate list.

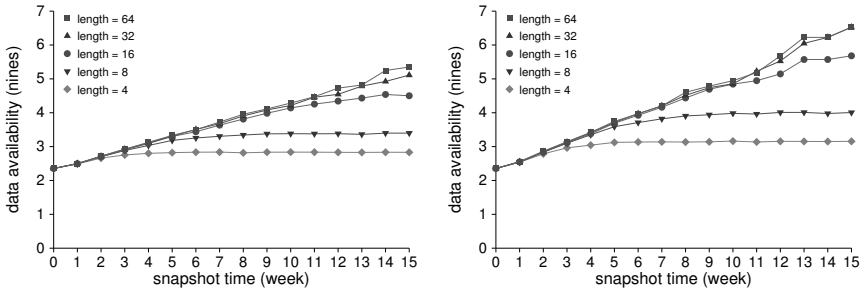
4.2.1. Availability Improvement by HBHC

We first measured the data availability improvement by history-based hill-climbing algorithm (HBHC). We compare this scheme with random hill-climbing method (RANDOM) and no-climbing case, in a P2P system with 10^6 nodes and cyclic ratio of 50%. Redundancy groups are configured by using 16 out of 32 erasure coding, with length of the candidate list as 32, which is referred to as 16/32/32.

Our results, shown in Figure 3(a), indicate that both hill climbing algorithms improve data availability steadily while HBHC performs better in a long run. Availability under no-climbing policy remains unaltered at 2.5 nines within our measurement period. RANDOM climbing achieves 5 nines after 15 weeks since it always brings an available node to each redundancy group, thus the chance that data is accessible is increased. During the first week, HBHC does not differ much from RANDOM, but it outperforms RANDOM by 1–2 nines as more statistical information is gathered as time goes on.

4.2.2. Redundancy Group Configuration

The m out of n erasure coding scheme is one of the determining factors in data availability. Any number can be picked as n, and the larger the n is, the higher the bandwidth can be used, but the more coordinating effort should be paid among each redundancy group. m is dependent on n, and the ratio of between m and n is the *storage efficiency* r ($r = m/n$). Higher storage efficiency implies lower redundancy, and thus weaker protection capability against node unavailability. The choice of the parameters m and n depends on the design goal of a particular peer-to-peer system. We chose three kinds of erasure coding configurations to investigate the tradeoffs among them: 16 out of 24 (r = 66%), 16 out of 32 (r = 50%), and 32 out of 64 (r = 50%). The length of the candidate list is set up to be 32 for each configuration.



(a) RANDOM climbing (b) HBHC

Figure 4. Availability Improvement of Candidate List Length. Redundancy group configuration: 16 out of 32.

From Figure 3, we see that the configuration of erasure coding greatly impacts data availability. It is no surprise that 32/64 provides the best availability amongst the three schemes. 16 out of 24 can only give 3 nines even with HBHC algorithm, but 3.5 nines can be achieved without any hill climbing schemes when 32 out 64 is used. Our simulator accuracy is 7 nines. With HBHC, data availability of the 32/64/32 configuration reaches this level after 11 weeks. We noticed a slight drop in availability from the 8th to the 9th week by HBHC with 32 out 64 scheme, and we attribute this to lack of measurement accuracy. As we see, the 16/24/32 configuration has very limited availability, and the 32/64/32 one works well in the system but its overhead is higher than 16 out of 32. In the rest of the paper, we use 16 out of 32 as the default configuration of redundancy group if not specified.

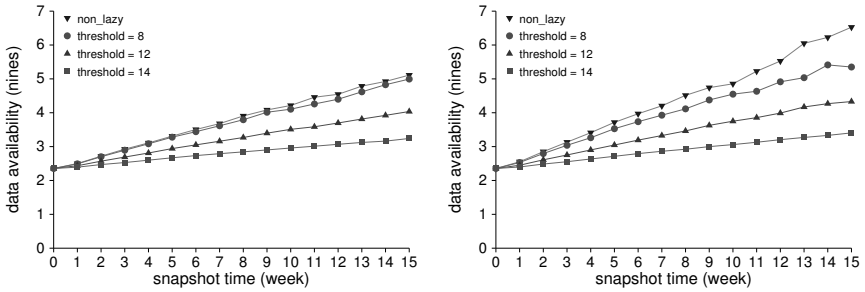
4.2.3. Length of the Candidate List in Hill Climbing Schemes

The key part of hill-climbing algorithms is the ability to choose a node from a candidate list and to swap it in the redundancy group, but how long should a candidate list be? A longer list has a higher potential to include "good" nodes in a redundancy group; on the other hand, a shorter list invokes lower overhead for coordinating nodes among a group. We investigated the impact of the length of the candidate list by varying its length from 4 to 64 in the 16 out of 32 configurations for both RANDOM and HBHC. Shown in Figure 4, longer candidate lists improve the overall availability, but the impact of moving from 32 candidates to 64 is limited. We concluded that a very long candidate list is not necessary because 32 candidates are enough to make efficient placement adjustment.

4.2.4. Lazy Climbing Strategy

As we discussed in Section 3.1, lazy climbing can reduce the number of adjustment process and thus decrease the overhead by aggressive hill climbing. We offer a lazy option in random hill climbing and HBHC scheme. A parameter T is set up to control the degree of laziness in hill climbing. For an example, if T is 14, then only when the number of unavailable nodes in a redundancy group is equal to or greater than 14, the hill climbing process would start; otherwise, data stays on the original set of nodes in the group.

We examine the impact of the threshold of the lazy climbing on data availability in both hill climbing schemes by varying



(a) RANDOM climbing. (b) HBHC

Figure 5. Availability at varied lazy options. Redundancy group configuration: 16/32/32.

the threshold values. As expected, Figure 5 shows that a higher lazy threshold leads to lower data availability. However, more than 5 nines are achieved even when the threshold is 8, which is $\frac{1}{2}(n-m)$ for the m out of n configuration where m is to 16 and n is 32. Although it is about one nine lower than the non-lazy strategy when HBHC is used, the adjustment effort is about half of the non-lazy one. We are in favor of lazy climbing strategy when the threshold is properly set up since we can both achieve high data availability while maintaining low system cost for placement adjustment.

4.3. Summary

Non_lazy HBHC performs best among all the policies examined across three coding schemes. However, it pays a higher cost of more-aggressive adjustment of data placement than lazy HBHC and produces extra overhead of statistics collection over RANDOM. In order to improve availability while still keeping low system overhead, the choices of the erasure coding scheme and the level of laziness in hill-climbing (*i. e.* the threshold) are essential. The coding scheme is also directly related bookkeeping overhead in each redundancy group. The more aggressive the coding scheme (*i. e.* the larger n in an n/m scheme), the more complicated the maintenance of redundancy groups. For example, the membership in each redundancy group under 32/64 scheme is about twice that under 16/32, resulting in more messages for group communication. Availability favors using more available nodes and hill-climbing schemes are biased against cyclic nodes. This introduces an interesting question at the level of the business model, namely how to dissuade people from exploiting this bias to protect their node from being used and how to detect this behavior.

5. Related Work

A measurement study of peer-to-peer file sharing systems [10] captures a significant amount of heterogeneity across the peers in the two popular multimedia file sharing systems, Napster and Gnutella. This heterogeneity appears in bandwidth, latency, and availability. It is also reported that about 60% of the hosts in each systems had less than 70% availability.

Douceur [4] examined availability distributions and proposed an analytical model which postulates that hosts exhibit two classes of availability behaviors— always-on and cyclic-on/off. The availability study of the Overnet P2P file system by Bhagwan *et al.* [2] observed the similar behavior pattern. We built our model based on this pattern.

FARSITE [1] is a serverless distributed file system that provides improved file availability and reliability. A directory

group is used to ensure that the files they store are never lost. FARSITE assumes that recovery is sufficiently fast that data is never lost as the maximum size of clients is at the order of 10^5. Their large-scale simulations indicate that the random replica placement is better than the replacements that consider availability at initialization, for reasons of the distribution of free space and evenness of machine reliability. Different from P2P systems, FARSITE is designed to support typical desktop workloads in academic and corporate environments. Nodes in FARSITE are far less dynamic than those in a global peer-to-peer system.

Pangaea [9] provides fault tolerance through server-based pervasive replication. It assumes trusted servers and allows individual servers to continue serving most of their data even when disconnected. It distinguished between "gold" and "bronze" replicas, where the golden ones play an additional role in maintaining the hierarchical name space.

OceanStore [7] is a wide area storage system which provides high availability, locality, and security by supporting nomadic data. It uses erasure coding to provide redundancy without the overhead of strict replication and is designed to have a very long Mean-Time-To-Data-Loss (MTTDL). An analysis of the reliability of OceanStore [12] showed that erasure codes had higher reliability than simple replication for a given amount of storage overhead.

6. Conclusion and Future Work

This paper introduces a cooperative storage technique which employs erasure coding schemes on a collection of data objects, as called redundancy groups, and provides various levels of data redundancy. Based on this technique, we propose a history-based hill climbing (HBHC) data placement policy to improve availability in P2P systems, which considers cyclic behavior including daily and weekend-off availability pattern. We have studied several essential factors related to HBHC, including coding schemes for redundancy groups, length of candidate list, and laziness of climbing.

Our simulation results show that hill climbing schemes, including HBHC and RANDOM, can achieve higher and higher availability as time goes on, by adjusting the locations of the nodes in each redundancy group. HBHC outperforms RANDOM by keeping track of nodes and taking advantage of their availability history. With lazy options in hill climbing schemes, system overhead is reduced as data availability gets enhanced. We also noticed that a very long candidate list is unnecessary.

Our work shows the potential to use hill climbing algorithms to boost data availability in global P2P systems. We are still refining our model and plan to feed traces of P2P systems into our simulator in order to consider the changing cyclic behavior in real systems. We also plan to measure the efficiency of message dissemination in such a system and use machine learning techniques to obtain an optimal combination of nodes in each redundancy group.

Acknowledgments

We would like to thank the members of the Storage Systems Research Center for their help and guidance. We especially thank Professor David Helmbold for his advice on this research. We would also like to thank the anonymous reviewers for their helpful comments.

References

[1] A. Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, Dec. 2002. USENIX.

[2] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, Berkeley, CA, 2003.

[3] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. TotalRecall: System support for automated availability management. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2004.

[4] J. R. Douceur. Is remote host availability governed by a universal law. *Performance Evaluation Review*, 31(3):25–29, Dec. 2003.

[5] J. R. Douceur and R. P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*, pages 311–319, Cincinnati, OH, Aug. 2001. IEEE.

[6] R. J. Honicky and E. L. Miller. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In *Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, NM, Apr. 2004. IEEE.

[7] 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 the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Cambridge, MA, Nov. 2000. ACM.

[8] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 188–201, Banff, Canada, Oct. 2001. ACM.

[9] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the Pangaea wide-area file system. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, Dec. 2002.

[10] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical Report UW-CSE-01-06-02, University of Washington, July 2001.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '01)*, pages 149–160, San Diego, CA, 2001.

[12] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, Cambridge, Massachusetts, Mar. 2002.

[13] Q. Xin, E. L. Miller, T. J. Schwarz, D. D. E. Long, S. A. Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 146–156, Apr. 2003.