PeerVault: A Distributed Peer-to-Peer Platform for Reliable Data Backup

Adnan Khan[1], Mehrab Shahriar[1], Sk Kajal Arefin Imon[1],
Mario Di Francesco[2,1], and Sajal K. Das[1]

[1] Center for Research in Wireless Mobility and Networking (CReWMaN)
University of Texas at Arlington
{firstname.lastname}@mavs.uta.edu, das@uta.edu
[2] Department of Computer Science and Engineering
Aalto University School of Science
mario.di.francesco@aalto.fi

Abstract. Large scale peer-to-peer (P2P) systems are envisioned as a way to provide online storage service. For a reliable storage service, the participating peers are required to maintain strict commitments for their online duration. On the other hand, recent results show that users participating in volunteer computing collectively exhibit certain patterns in terms of their long-term availability, a metric that denotes periodic online durations for a considerably long time interval. In this article we introduce PeerVault, a P2P platform that leverages the long-term availability of the computer users to form a distributed reliable storage service, targeted to backup of personal data. We further present a distributed monitoring scheme that assists PeerVault to detect peer churns and ensure the reliability of the proposed backup service. To the best of our knowledge, this is the first effort to describe the architecture of a reliable P2P backup service exploiting the long-term availability and idle resources of computing devices. We conduct experiments based on the availability traces of hundreds of thousands of hosts from the SETI@home computing project. The obtained results show that the proposed approach is effective in terms of availability as well as reliability of the offered backup service.

Keywords: Peer-to-peer backup, distributed storage, reliability, monitoring.

1 Introduction

Traditional approaches for reliable storage of data involve maintaining redundant *backup* copies on a variety of storage devices which evolved over time from tapes and optical media to external hard drives. More recently, the availability of storage space in data centers and ubiquitous access to the Internet have made remote storage solutions increasingly appealing. In this context, cloud-based services such as Amazon Cloud Drive, Microsoft SkyDrive and DropBox have become very popular. Such systems are easy to use and provide seamless service as they exploit the reliability inherent to the cloud infrastructure. Most of these services provide a limited storage space without any charge to attract new users. Subsequently, users pay for additional storage space on a monthly or yearly basis.

Although computer users keep on purchasing additional storage for data backup, a significant portion of their own local storage remains unutilized. According to [1], almost half of the users consistently utilize less than 40% of their available disk space. Therefore, a different approach to design a backup system is to exploit the unutilized disk space and bandwidth in a peer-to-peer (P2P) architecture. Such *P2P backup systems* have also received attention by the research community [2] and even resulted in actual commercial products [3, 4] in the last few years. However, most of the existing systems require the users to be online for a considerably long period of time, e.g., more than 80% time of a day [4]. Moreover, these systems require the users to contribute their own storage space to receive the backup service [3] and thus excludes the users who are unwilling to contribute their own resources.

In this article, we introduce *PeerVault*, a platform which exploits the long-term availability of online computer users, as well as their idle resources, in order to realize a distributed online backup service based on a P2P infrastructure. In this approach, participating users advertise their unused storage and network resources based on which PeerVault decides how to store the data by ensuring their long-term availability. To receive the backup service, users are not explicitly required to contribute any resources. Even though the backup service can be supported by an appropriate revenue model [5], in this work we focus on the architectural aspects of the system. The major contributions of this article are as follows.

– We design a novel distributed storage system based on erasure coding which realizes a seamless online backup service on top of idle peers connected over the Internet.
– We propose the concept of *peer path* to derive an efficient solution for distributing data to the peers. Peer paths encapsulate individual peer availabilities to offer a seamless backup service over a given time interval.
– We devise a distributed monitoring scheme to detect peer churn. The proposed algorithm is shown to monitor all the involved peers with high probability, while incurring a nominal bandwidth.
– Through extensive simulations based on the traces of the SETI@home project [6], we show that the proposed approach is effective in terms of long-term service availability.

The remainder of the article is organized as follows. Section 2 details the proposed PeerVault architecture with focus on the feasibility of the offered service. Section 3 introduces a randomized scheme to monitor peer churn in our system. Section 4 presents the details of the simulation setup and the obtained results. Section 5 summarizes the related work and, finally, Sect. 6 concludes the article with directions for future research.

2 PeerVault Architecture

The proposed PeerVault architecture is based on the three basic components illustrated in Fig. 1. The *source peers* are the end-users of the system and are willing to store data (namely, *files*) in exchange for a high reliability. On the other hand, the *storage peers* provide their bandwidth and storage resources to realize the distributed backup service. Finally, the *tracker* supervises the resources offered by the storage peers as well as the mapping between files and peers. Source peers can request a certain amount of remote storage space for a particular period of time, with a minimum bandwidth desired for uploading or retrieving the data. Similarly, a storage peer can choose the amount of space it is willing to share, the minimum upload and download bandwidth, and its availability periods.

Throughout our discussion, the *availability* of a storage peer will refer to its compliance with the advertised resources. We will refer to *service availability* of PeerVault at a given time instant as the accessibility of the stored files at that particular instant. Moreover, we will refer to *service reliability* as the long-term availability (i.e., in a sufficiently large time period) of the offered service. Since PeerVault is based on a P2P infrastructure, intermittent deviation from the advertised resources and also permanent departure of the storage peers are possible. The service availability of PeerVault relies on the group availability of the storage peers instead of the individual availabilities. Thus, service availability can be ensured, even when storage peers have some deviation from their advertised resources. Moreover, in Sect. 3 we explicitly provide a mechanism to detect and adjust with the deviations to ensure service reliability.

2.1 Distributed Storage Scheme

In PeerVault, a file is distributed by a source peer to a set of storage peers in the form of chunks. We exploit erasure coding to create these chunks from a given file. The basic idea behind this approach is to encode data by adding some redundancy. As a result, the original data can be obtained from the encoded data even when part of them is not available. Erasure coding operates on individual *chunks* of a file, where each chunk is of fixed size λ. In the following, we will assume that the source data (i.e., a file) is split into k chunks, and then encoded into $n = \eta k$ chunks, where η is the *replication factor* (see Fig. 1). Erasure coding guarantees that the original file can be reconstructed from any k distinct encoded chunks among the n encoded ones.

A suitable value of η is obtained through a preliminary negotiation phase between the source peer and the tracker, based on the resources available in the system. After that, the source peer applies erasure coding on the given file to produce n different chunks. The tracker derives a mapping between an encoded chunk and a set of storage peers, known as a *peer path*. The storage peers in the mapping are selected based on their advertised resources. Thus, for the entire file (i.e., the n encoded chunks), the tracker finds n peer paths and provides the related mapping to the source peer. The tracker also ensures that no storage

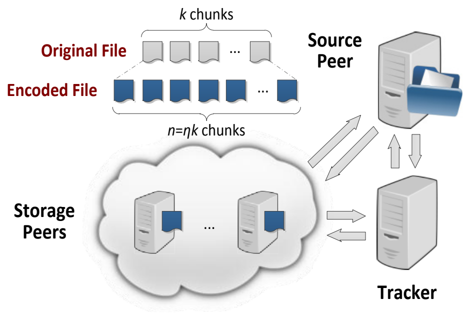


Fig. 1. System Architecture of PeerVault

peer receives more than $k-1$ chunks of a given file. As a consequence, no storage peer can reconstruct or access the given file.

2.2 Characterization of Storage Peers

Different storage peers provide their resources during different time intervals. On the other hand, a source peer may need to store or retrieve a file at any time instant. In the following, we will build our storage scheme based on the availability of the storage peers so that the requirements of the source peers are successfully satisfied.

First, we denote the i-th storage peer by means of its unique identifier, p_i. We assume that the availability of storage peers is periodic over a time frame, defined as *service time frame*. Specifically, the availability of storage peers is characterized in terms of the considered service time frame. For instance, a given peer could be available from Monday to Friday between 12 AM to 8 PM when the service time frame is equal to one week. Within a service time frame, a peer can be available during multiple contiguous time intervals, referred to as the *availability periods*. We represent the j-th availability period of p_i as p_{ij}. In detail, we define as *arrival time* and *departure time* the instants corresponding to the beginning and the end of a single (contiguous) availability period, respectively. For a given availability period p_{ij}, we denote the corresponding arrival time as $a(p_{ij})$ and the departure time as $d(p_{ij})$. Each availability period p_{ij} is associated with its *offered bandwidth* $b(p_{ij})$, which is the minimum between the upload and download bandwidths of the storage peer during the availability period. Moreover, each availability period has an associated cost per unit storage, represented by $c(p_{ij})$. The *duration* of an availability period is denoted by $A(p_{ij}) = (a(p_{ij}), d(p_{ij}))$. The *overlapping time* between two availability periods p_{ij} and p_{kl} is finally defined as $T(p_{ij}, p_{kl}) = \min\{d(p_{ij}), d(p_{kl})\} - \max\{a(p_{ij}), a(p_{kl})\}$ if $d(p_{kl}) > a(p_{ij})$ and $d(p_{ij}) > a(p_{kl})$, otherwise $T(p_{ij}, p_{kl}) = 0$.

Let us consider the example scenario represented in Fig. 2a. For clarity, we assume that each storage peer has a single availability period, denoted by a single subscript corresponding to the peer identifier (i.e., p_i represents the only

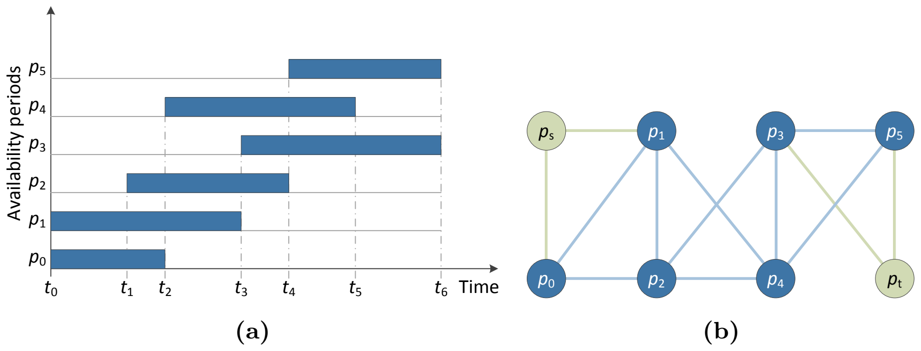


Fig. 2. (a) Availability periods of different storage peers as a function of time. (b) Interval graph corresponding to (a) with the addition of dummy nodes p_s and p_t.

availability period of the i-th peer). The durations of the availability periods p_1 and p_2 are $A(p_1) = (t_0, t_3)$ and $A(p_2) = (t_1, t_4)$, respectively. Note here that the availability period p_0 overlaps with both p_1 and p_2. Specifically, the overlapping time between p_0 and p_1 is $T(p_0, p_1) = t_2 - t_0$, while that between p_0 and p_2 is $T(p_0, p_2) = t_2 - t_1$.

2.3 Managing Storage Requests

The backup service is requested by a source peer (for an individual file) in terms of the following parameters: the *target availability interval* (δ_s, δ_e); the desired minimum download bandwidth μ; and the requested storage space ρ. We assume that the chunk size for the given file is λ and that the target availability interval requested by the source peer is equal to the service time frame.

We use interval graphs [7] to model the considered scenario. An undirected graph $G = (V, E)$ is called an *interval graph* if a one-to-one mapping between the vertices V and a set of intervals I can be established, such that two vertices are connected by an edge in G if and only if there is an intersection between the corresponding intervals. In our case, $V = \{p_{ij}\}$ and $I = \{I_{ij}\} = A(p_{ij}) = \{(a(p_{ij}), d(p_{ij}))\}$ for $0 \leq i < m$ and $0 \leq j < n_i$, where m is the number of storage peers and n_i is the number of availability periods of p_i.

We construct a constrained interval graph, G_c, for the given storage request according to the availability periods of the storage peers. Let us assume, for an availability period p_{ij}, the offered bandwidth and the cost are denoted by $b(p_{ij})$ and $c(p_{ij})$, respectively. Now we restrict the nodes in the graph G_c to those with offered bandwidth higher than $\frac{\mu}{k}$. Furthermore, we restrict the edges between any two nodes p_{ij} and p_{kl} so that their overlapping time is longer than the minimum overlapping time τ, where $\tau = (\min\{b(p_{ij}), b(p_{kl})\})^{-1} \cdot \lambda$. Note that a chunk stored in a peer can be transferred to the next peer along the associated peer path in the minimum overlapping time. Finally, we define the weight of an edge between nodes p_{ij} and p_{kl} as $w(p_{ij}, p_{kl}) = \frac{c(p_{ij})+c(p_{kl})}{2}$.

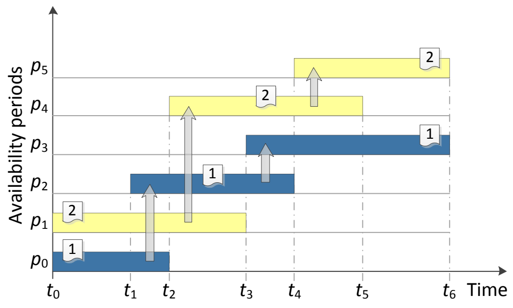


Fig. 3. Dissemination of file chunks

On the basis of the target availability interval (δ_s, δ_e), we add two dummy availability periods p_s and p_t, so that a storage request can be mapped to a path between a single source and a single destination in G_c. The duration of the availability periods associated to the dummy nodes are set to $A(p_s) = (\delta_s, \delta_s + \tau)$ and $A(p_t) = (\delta_e - \tau, \delta_e + \tau)$, respectively. We also set $b(p_s) = b(p_t) = \frac{\mu}{k}$ and $c(p_s) = c(p_t) = 0$. As a consequence, a peer path can be referred by a path between p_s and p_t in G_c. Formally, a *peer path* associated with the interval (δ_s, δ_e) is the set of m availability periods $\mathcal{P}(\delta_s, \delta_e) = \{p_{i_1 j_1}, p_{i_2 j_2}, \ldots, p_{i_m j_m}\}$ such that $T\left(p_{i_x j_y}, p_{i_{x+1} j_{y+1}}\right) > \tau$, $\forall i \in [1, m]$, $a(p_{i_1 j_1}) \leq \delta_s$ and $d(p_{i_m j_m}) \geq \delta_e$. For instance, $\mathcal{P}(t_0, t_6) = \{p_s, p_0, p_2, p_3, p_t\}$ is a peer path in Fig. 2b. Note that the parameters assigned to the dummy nodes ensure the inclusion of the peers with the required amount of overlapping time in a peer path.

For a storage request with n encoded chunks, our system associates a distinct peer path $\mathcal{P}_r(\delta_s, \delta_e)$, with $0 \leq r < n$, to each of the chunks in the source file. For a storage request, we intend to assign at most one chunk to a single availability period so that any interruption during this availability period has minimal impact. Moreover, a storage peer is not allowed to receive more than $k - 1$ chunks of the file, even though it may have multiple availability periods. Otherwise, it would be possible for a storage peer to obtain k or more chunks and reconstruct the file. As a consequence, at most $k - 1$ availability periods of a storage peer are allowed to belong to a single storage request. To this end, for each storage peer p_i, we sort the availability periods based on the weight $\psi(p_{ij}) A(p_{ij})$, for $0 \leq j < n_i$ in decreasing order, where $\psi(p_{ij})$ represents the probability of p_i being online during p_{ij} as explained in Sect. 2.5. Thus, we further restrict G_c by taking the top $k - 1$ availability periods from the sorted list.

Finally, to serve the storage request, PeerVault selects the set of peer paths $\mathcal{X} = \left\{\cup_{j=0}^{n-1} \mathcal{P}_j(\delta_s, \delta_e)\right\}$ so that $\mathcal{P}_i \cap \mathcal{P}_j = \{p_s, p_t\}$, $\forall i \neq j$ and the total cost of the availability periods in the selected peer paths is minimized. Note that this problem can be mapped to the *minimum weight n-node disjoint path problem* in an undirected graph [8] which is well studied in the literature, and can be solved in polynomial time [9].

2.4 Data Dissemination and Retrieval

According to the definition of peer path, at any time instant, a storage peer can be found online. When a source peer intends to backup a file, it creates n encoded chunks and sends a storage request to the tracker. The tracker selects a set of n peer paths and sends it back to the source peer. Now the source peer uploads each chunk to the currently available storage peer from each peer path. After all the chunks are uploaded, the source peer can leave the system. Once a storage peer of a peer path receives a chunk, it transfers the chunk to the next storage peer of the peer path. Thus the chunk is propagated to all the storage peers of the peer path. This dissemination process for a file consisting of two encoded chunks is illustrated in Fig. 3 for the scenario already introduced in Fig. 2a. In this example, the tracker reports two peer paths to the source peer, namely $\mathcal{P}_1 = \{p_0, p_2, p_3\}$ and $\mathcal{P}_2 = \{p_1, p_4, p_5\}$. If the source peer is online at t_0, it can upload chunk 1 to p_0 and chunk 2 to p_1, and may leave the system.

When the source peer decides to retrieve the stored file, it selects k distinct peer paths and proceeds to download the chunks from the currently available storage peer of each peer path. Once k chunks are successfully downloaded, the source peer reconstructs the original file.

2.5 Estimating Available Resources

As the offered backup service is largely dependent on the long-term availability of the storage peers, it is essential to know the relevant parameters of a storage peer – namely, availability periods, bandwidth, and storage space – before it is actually allowed to participate in the system. Unlike some existing approaches [4], we do not rely on the user to define the expected operating parameters. Instead, PeerVault observes the users for a *training period* denoted by σ.

We use the bit vector method similar to [10] to predict the long-term availability of the storage peers. Consistent with that solution, we consider the service time frame of one week, wherein each hour of the week is represented by a bit. For each hour, the corresponding bit is set to 1 if a storage peer is available for more than 55 minutes. The peer is observed for each hour in the entire training period. Let us assume that there are y weeks in σ, and a bit is set for x weeks. Then the *training probability* of the corresponding hour is defined by $\frac{x}{y}$. We consider an hour to include in an availability period if the training probability exceeds a threshold α_b. Finally, the availability periods are obtained by merging the contiguous available hours. The training probability of the availability period is denoted by $\psi(\cdot)$ and computed by taking the average of the probabilities of the constituent hours. A storage peer is considered as eligible if it has at least one availability period with training probability greater than α_b. At the end of the training period, an eligible peer is requested to approve its estimated availability periods and specify the information of the free disk space, bandwidth, and cost it can offer to PeerVault. Subsequently, the associated cost per unit storage, $c(\cdot)$ is derived through a revenue model. These sets of parameters are referred as the advertised resources of the storage peer for the considered availability period. A specific choice of the revenue model is out of the scope of this paper.

3 A Distributed Peer Monitoring Scheme

To ensure the long-term availability of the stored data, peer churns must be detected and the corresponding peers need to be replaced accordingly. In this section, we introduce a distributed algorithm to monitor storage peers and detect churn. In our approach, each storage peer sends a ping message to a set of other peers to monitor whether or not they are maintaining their advertised resources. Our algorithm, called DistMonitor has the following properties: (i) the absence of a storage peer is reported to the tracker with high probability; (ii) the overhead of the monitoring effort is proportional to the number of stored chunk and, hence, is fairly distributed; (iii) newly joined peers can easily be included in the monitoring process, thus making the solution scalable; (iv) most of the monitoring overhead is assigned to the peers themselves, while only limited interactions with the tracker are needed; and (v) overall, the required bandwidth for the monitoring scheme is nominal.

For convenience of discussion, let γ_k denotes a particular availability period. Let $h(\cdot)$ be a one-to-one function that maps an ordered pair of integers $< i, j >$ to a single integer k. Thus, γ_k represents a unique availability period p_{ij}.

Definition 1 (Simultaneous Availability Period List). *Let n chunks of a file be stored among a set of peers with availability periods $P = \{\gamma_1, \gamma_2, \ldots, \gamma_m\}$. The* simultaneous availability period list *(SAPL) for a given availability period $\gamma_i \in P$ is the set $\mathcal{S}_i \subseteq P$ such that $T(\gamma_i, \gamma_j) > t_{max}$, for all $\gamma_j \in S_i \setminus \{\gamma_i\}$ and t_{max} is a predefined timeout period greater than zero.*

Definition 2 (Potential Availability Period List). *The* potential availability period list *(PAPL) $\mathcal{N}_i$ of a given availability period γ_i is a randomly selected proper subset of $\mathcal{S}_i$.*

After finding the peer paths for a given file, the tracker computes the PAPL for each of the availability periods. Basically, when a storage peer participates in storing a file (by holding a chunk during a particular availability period), it is also assigned with the PAPL.

Once a storage peer p_k obtains the PAPL for a particular availability period γ_i, it executes the function DistMonitor illustrated in Algorithm 1. Specifically, the storage peer selects q random availability periods from the PAPL of γ_i and assigns it to a set $\mathcal{B}$ (line 1). For each member of $\mathcal{B}$, there is a counter (*count*) initialized with a value l (line 2). For each of the availability periods γ_j from $\mathcal{B}$, the peer p_k selects a random time in the overlapping time $T(\gamma_i, \gamma_j)$ and sends a ping message at that particular time to the storage peer associated with γ_j, namely, $g(\gamma_j)$ (line 11). Note that $g(\cdot)$ is a function that returns the identifier of a peer corresponding to a given availability period. After sending the ping message, p_k waits for the reply for a predefined timeout period t_{max}. If the reply is received within that period, γ_j is removed from the list $\mathcal{B}$, and it is assumed that the corresponding peer is conforming to its commitment. Otherwise, the value of the corresponding *count* is decremented by 1 (lines 14–15). Note that p_k may try to send at most l messages to a particular peer associated with an

Algorithm 1. DistMonitor($\gamma_i, \mathcal{N}_i, q, l$)

```
output: $\mathcal{R}$                                   // list of reported peers
1  $\mathcal{R} \leftarrow \varnothing$; $\mathcal{B} \leftarrow q$ randomly chosen elements from $\mathcal{N}_i$;
2  foreach $\gamma_j \in \mathcal{B}$ do  $count[\gamma_j] \leftarrow l$ ;
3  while $\mathcal{B} \neq \varnothing$ do
4      foreach $\gamma_j \in \mathcal{B}$ do
5          if $count[\gamma_j] = 0$ then  $\mathcal{R} \leftarrow \mathcal{R} \cup g(\gamma_j)$ ; $\mathcal{B} \leftarrow \mathcal{B} \setminus \gamma_j$;
6          else
7              $t' \leftarrow [t_{now}, \infty]$ ; $t_{ol} \leftarrow T(\gamma_i, \gamma_j) \cap t'$;
8              if $t_{ol} = 0$ then  $\mathcal{B} \leftarrow \mathcal{B} \setminus \gamma_j$;
9              else
10                 $t[\gamma_j] \leftarrow$ randomly selected value from $t_{ol}$;
11                 schedule a message for $g(\gamma_j)$ at $t[\gamma_j]$ ;
12                 schedule a thread waiting for $\gamma_j$ from $t[\gamma_j]$;
13     foreach $\gamma_j \in \mathcal{B}$ do
14         if a reply is received within $T[\gamma_j] + t_{max}$ then  $\mathcal{B} \leftarrow \mathcal{B} \setminus \gamma_j$;
15         else  $count[\gamma_j] \leftarrow count[\gamma_j] - 1$;
16 report $\mathcal{R}$ to the tracker;
```

availability period. If the value of *count* is 0 for a particular availability period γ_j, then p_k adds the corresponding peer $g(\gamma_j)$ to the set $\mathcal{R}$, (line 5), where $\mathcal{R}$ denotes the set of peers that have not responded to the ping messages. Before γ_i ends, p_k sends $\mathcal{R}$ to the tracker as negative feedback.

3.1 Analysis of DistMonitor

The performance of the monitoring algorithm is measured in terms of two metrics, namely, percentage of peers that were monitored and the associated message overhead. Let the chunks of a particular file is stored among a group of peer availabilities. We denote γ_i^j as the i-th availability period in the peer path j (holding the j-th chunk of the file). Let $|\overline{\mathcal{S}}|$ and $|\overline{\mathcal{N}}|$ denote the average size of SAPL and PAPL of the involved availability periods. The following theorem characterizes q (the number of selected availability periods from the PAPL) and $|\overline{\mathcal{N}}|$ to ensure the desired performance of the DistMonitor algorithm.

Theorem 1. *For a stored file, each storage peer is monitored in its availability period by* DistMonitor *with high probability, for a proper choice of q and $|\overline{\mathcal{N}}|$, i.e., $q = |\overline{\mathcal{N}}| = \log |\overline{\mathcal{S}}|$.*

Proof. Let us assume that a file has n encoded chunks and $C = \{\gamma_{i_1}^{x_1}, \gamma_{i_2}^{x_2}, \ldots, \gamma_{i_m}^{x_m}\}$ is the group of availability periods for a particular chunk. Let $\gamma_i^j \in C$ be an availability period such that all other members in C are in the SAPL of γ_i^j (excluding itself). Without loss of generality, let us assume that the size of the SAPLs and PAPLs of all the availability periods in C are $|\overline{\mathcal{S}}|$ and

$|\overline{\mathcal{N}}|$, respectively. We aim at finding the probability of a peer p_k corresponding to γ_i^j being monitored, that is, the probability of receiving at least one message from any of the peers corresponding to the availability periods of C. Essentially, all the corresponding peers of C (except for p_k) contain p_k in their SAPL, so each of these peers has a probability to send a ping message to p_k.

Let us introduce the following notation first. Let D_k be the event that p_k receives at least one message from any of the peers, and G be the event that p_k receives a message from p_l. By recalling that a peer sends out ping messages to q randomly selected peers from its PAPL $\mathcal{N}$ (for a particular availability period), let us also define M as the event that p_k is in the PAPL of p_l, and Y as the event that p_k is sent a message by p_l. Hence, $P(G) = P(M)P(Y|M)$. Now,

$$P(M) = 1 - \left(1 - \frac{1}{|\overline{\mathcal{S}}|}\right)^{|\overline{\mathcal{N}}|}$$

and $P(Y|M) = q \cdot (|\overline{\mathcal{N}}|)^{-1}$. Therefore,

$$P(G) = \frac{q}{|\overline{\mathcal{N}}|}\left(1 - \left(1 - \frac{1}{|\overline{\mathcal{S}}|}\right)^{|\overline{\mathcal{N}}|}\right) = \frac{q}{|\overline{\mathcal{N}}|}\left(1 - e^{-\frac{|\overline{\mathcal{N}}|}{|\overline{\mathcal{S}}|}}\right)$$

So $P\left(\overline{D_k}\right) = (1 - P(G))^{|\overline{\mathcal{S}}|} = f\left(|\overline{\mathcal{N}}|, q\right)$, for a fixed value of $|\overline{\mathcal{S}}|$. Thus $P(D_k) = 1 - f\left(|\overline{\mathcal{N}}|, q\right)$. Thus, the probability of a peer being monitored depends on $f(\cdot)$ which, in turn, depends on q and $|\overline{\mathcal{N}}|$ for a specific file. For an instance, if we choose both q and $|\overline{\mathcal{N}}|$ as 1, a peer is monitored with a constant probability of around 63% by other storage peers. In the specific case where $\log|\overline{\mathcal{S}}|$ is chosen for both $|\overline{\mathcal{N}}|$ and q, $f(\cdot)$ becomes 0 with high probability asymptotically with increasing values of $|\overline{\mathcal{S}}|$. Therefore, a peer is guaranteed to be monitored with high probability in its availability period for storing a single file chunk when $q = |\overline{\mathcal{N}}| = \log|\overline{\mathcal{S}}|$.

Now, we consider the bandwidth requirements for the monitoring scheme.

Lemma 1. *In an availability period, a peer sends/receives a total of* $O(\xi \log|\overline{\mathcal{S}}|)$ *ping messages, where* ξ *represents the number of file chunks it holds.*

Lemma 1 follows from the following arguments. During an availability period, for each file, a storage peer sends out $O(q)$ ping messages. It can be shown that the expected number of ping messages received by a storage peer is also $O(q)$. In addition, a peer has to send (and receive) $O(q)$ reply messages. In our application, a storage peer can send/receive at most $l \log|\overline{\mathcal{S}}|$ ping messages for a single file. Therefore, in total, a peer can send/receive at most $l\xi \log|\overline{\mathcal{S}}|$ ping messages. Similarly, a peer can send/receive at most $l\xi \log|\overline{\mathcal{S}}|$ reply messages.

If the average size of the ping and reply messages is α, and the availability period is $\overline{A}$, a peer incurs an average download/upload bandwidth of $\frac{4 \cdot l\alpha\xi \log|\overline{\mathcal{S}}|}{\overline{A}}$. When $l = 2$, as in our application, a peer with an availability period of 20 hours contributing 100 GB storage space may incur an average upload/download bandwidth of less than 0.75 KBps. This assumes the average size of ping/reply message as 100 bytes, and the average chunk size as 10 MB.

Replacement Strategy. The tracker maintains a negative feedback counter for each availability period. If it receives negative feedback for more than 10 times about an availability period in a particular week, it verifies whether the peer is unavailable by sending periodic ping messages for the next 4 weeks. Based on the response, the tracker computes the probability of the associated peer to be available using the bit vector method (recall from Sect. 2.5). If the probability is less than 0.2, the tracker picks a new availability period with similar or longer availability duration and similar or higher bandwidth offering the minimum cost.

4 Performance Evaluation

We simulated the PeerVault system based on the user availability traces of the SETI@home project [6]. SETI@home is a scientific experiment that uses the idle resources of the Internet-connected computers, in the Search for Extraterrestrial Intelligence (SETI).

4.1 Simulation Setup

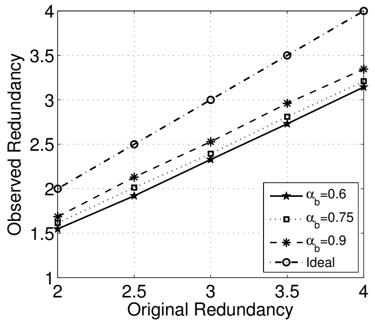
In the following, we will present the details about the traces, the parameters, and the methodology used in the performance evaluation.

SETI@Home Traces. In our experiments, we used the traces corresponding to the CPU availability of the SETI@home project as collected by the Failure Trace Archive [6]. We consider each unique host in the trace as a storage peer. The data reported in the trace spans over a period of a year and nine months that we call the *trace duration*. All the hosts of the trace data are not available for the entire trace duration. Some hosts start contributing after the trace duration starts, while some others leave permanently before the trace duration ends. Thus, each host (or storage peer) has trace data over an interval that we call *host duration*. If the host duration of a storage peer is (t_1, t_2), we define the *prediction interval* as $(t_1 + \sigma, t_2)$ if $t_1 + \sigma < t_2$, and 0 otherwise (in which case, we ignore that particular host). Recall from Sect. 2.5 that σ refers to the training period.

Simulation Details and Relevant Metrics. We carried out the experiments through a custom simulator written in Java to validate the availability and reliability of the backup service as well as the performance of the monitoring scheme, DistMonitor. In order to serve the storage requests, we implemented the minimum weight n-node disjoint path algorithm proposed in [9]. The availability periods were derived by using the method described in Sect. 2.5, with a training period σ of 4 weeks. We considered three different datasets of storage peers for the experiments. For each dataset, we picked a random sample of $10,000$ hosts, from which we extracted storage peers with training probability equal to or greater than $0.6, 0.75$, and 0.9. Throughout this section, they will be referred

Training probability threshold (α_b)	Hosts selected from sampled ones (%)	Average number of availability periods per host	Average length of availability periods (hours)
0.6	71	3.38	31.28
0.75	70	3.4	29.75
0.9	53	3.32	26.84

(a)



(b)

Fig. 4. (a) Availability periods obtained from traces. (b) Effective redundancy against the original redundancy of the files for storage peers with different training probability thresholds.

to as datasets A, B and C, respectively. Table 4a shows the percentage of hosts with the desired training probability and the number of availability periods per host which are considered in the simulation. We performed independent experiments for each of the datasets. In each experiment, 1,000 files are requested and the file sizes were generated from a lognormal distribution with a mean and standard deviation of 100 MB and 20 MB, respectively [11].

We considered the following performance metrics:

– *Observed redundancy*: the ratio of the number of available encoded chunks (n^*) to the minimum number of encoded chunks (k), for a given file.
– *Percentage of available files*: the ratio of the files with greater than or equal to k chunks available to the total number of files initially stored.

4.2 Experimental Results

Figure 4b shows the observed redundancy, averaged over all stored files, against the applied redundancy. In all datasets, the observed redundancy for a single service time frame (i.e., the first week) is summarized in a single plot to assess the availability of the offered service in a short time frame. The figure clearly shows that the observed redundancy increases with the threshold for increasing training probability of the hosts. Therefore, a higher threshold for training probability (e.g., higher than or equal to 90%) can be used to achieve a better performance. The line marked as ideal represents the case wherein all peers are available.

Figure 5 shows the availability of the stored files over a long time period to assess the reliability of the offered service. Specifically, it shows the percentage of accessible files (with $\eta = 2.5$) over a period of 52 weeks for datasets A and[1] C. The figures show that the availability of the files gradually decreases

[1] Results for dataset B are similar to those for dataset A, so we did not report them here due to lack of space.

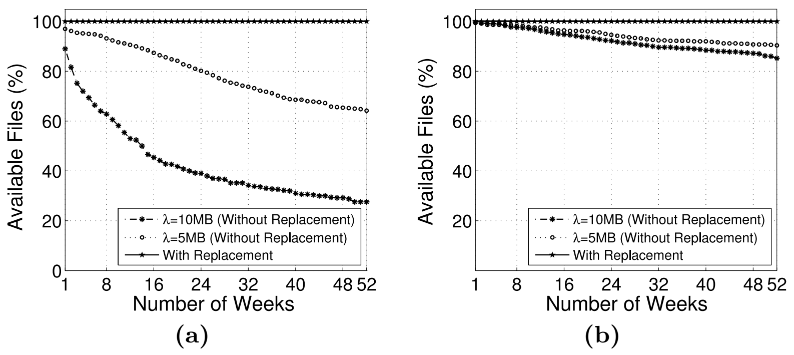


Fig. 5. Available files during 52 weeks for the different datasets: (a) A ($\alpha_b = 0.6$); and (b) C ($\alpha_b = 0.9$).

for all datasets. Even though dataset C shows a much higher availability over time than others (i.e., after 1 year, around 90% files are still accessible), there is no guarantee that all files can be accessed throughout the entire simulated period when no monitoring and replacement are used. This result, in addition to Fig. 4b, suggests that file availability is improved and retained over time when the training probability is high. However, some peers permanently leave the system over time and, thus, the data stored by them become unavailable. The monitoring algorithm and the replacement policy can guarantee that the files are available over the entire simulated period. The results also suggest that the availability of files can be improved by reducing the chunk size. Since the peer paths increase when the chunk size λ decreases, the probability of getting the minimum number of chunks for a file increases as well. On the other hand, very small file chunks result in a higher overhead for both the tracker and the storage peers. After considering all the above-mentioned aspects, 5 MB appears to be a suitable choice for the chunk size.

5 Related Work

In the following, we will summarize relevant literature on P2P systems used for both storage and for monitoring purposes.

The P2P networking paradigm has been exploited in the context of distributed file systems [12–14]. However, none of the proposed approaches exploits the availability pattern and the idle resources of the computer users. In some existing works, P2P networks were used to provide an enhanced online storage service in addition to dedicated servers. FS2You [15] and Amazing Store [16] are examples of such hybrid P2P systems explicitly designed to improve the availability of stored data with efficient bandwidth utilization. Among P2P backup systems, Symform [4] offers up to 200 GB of free storage space. In return, users are required to be online at least 80% of the time and provide at least 1.5 times the storage they receive from the system. Unfortunately, the details on the system design are not publicly available. Wuala [3] is another commercial P2P backup system that relies on a symmetric service between users and exploits a hybrid architecture. A peer-assisted backup service was also proposed in [17], wherein it was shown that a performance comparable to traditional client-server architecture can be achieved by temporarily using storage space from cloud providers. However, all the above solutions still rely on the presence of special servers or data centers. In contrast, our solution is based on a pure P2P architecture. FriendStore [2] is a backup system where users store data by exploiting their social connection with other peers. Specifically, personal data are backed up on "friend" peers. Thus, availability and reliability depend on the number of friends, which can be rather low in realistic scenarios. In [18], a pricing mechanism for the offered resources in a P2P backup system is investigated. However, the work does not define any specific architecture as for the storage mechanism.

P2P networks employ monitoring schemes to ensure peer participation. A generic monitoring system based on the principles of autonomic computing was presented in [19]. Such a mechanism assumes that the P2P network is structured (i.e., has a logical overlay), thus, it is not directly applicable to our system. Existing P2P backup services use monitoring approaches which assign the monitoring responsibility to either a centralized server [4] or the peer that originated the backup request [3, 20]. On the other hand, our approach is distributed, since a peer is randomly monitored by some other peers, and assigns minimal responsibility to the tracker.

6 Conclusion

In this paper, motivated by the availability of unused disk space of the users and their long-term availability pattern, we introduced PeerVault, a data storage system based on a peer-to-peer architecture which can be used to provide a seamless backup service. PeerVault exploits group availability of participating peers to ensure long-term availability of the stored data. Moreover, to address peer churns, we proposed a distributed monitoring scheme that detects peers deviating from the desired availability pattern. Simulation results based on the traces of the SETI@home computing project demonstrated that the proposed approach efficiently utilizes the available resources and obtains a very high service reliability. In future, we propose to investigate revenue and recommendation models that will enhance the peer selection mechanism.

Acknowledgements. This research was partially supported by the NSF grants CNS-1049652, IIS-1064460, CNS-0916221, IIP-1242521 and CNS-1150192.

References

[1] Meyer, D.T., Bolosky, W.J.: A study of practical deduplication. Trans. Storage 7(4), 14:1–14:20 (2012)
[2] Tran, D.N., Chiang, F., Li, J.: Friendstore: cooperative online backup using trusted nodes. In: Proc. of the 1st Workshop on Social Network Systems, pp. 37–42 (2008)

[3] LaCie AG: Wuala – Secure Online Storage, `http://www.wuala.com` (retrieved November 16, 2011)

[4] Symform, Inc.: symform – Revolutionary Cloud Storage Network, `http://www.symform.com/our-solutions/storage-backup/` (retrieved July 22, 2012)

[5] Turner, D.A., Ross, K.W.: A lightweight currency paradigm for the p2p resource market. In: Proc. 7th ICEC (2004)

[6] Kondo, D., Javadi, B., Iosup, A., Epema, D.: The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), pp. 398–407 (May 2010)

[7] Fishburn, P.: Interval orders and interval graphs: a study of partially ordered sets. Wiley-Interscience series in discrete mathematics. Wiley (1985)

[8] Bhandari, R.: Survivable Networks: Algorithms for Diverse Routing. Kluwer Academic Publishers, Norwell (1998)

[9] Bhandari, R.: Optimal physical diversity algorithms and survivable networks. In: Proc. of Second IEEE Symposium on Computers and Communications (1997)

[10] Lázaro, D., Kondo, D., Marquès, J.M.: Long-term availability prediction for groups of volunteer resources. JPDC 72(2) (2012)

[11] Downey, A.B.: The structural cause of file size distributions. In: Proc. of the 2001 ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems. SIGMETRICS 2001, pp. 328–329 (2001)

[12] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: Oceanstore: an architecture for global-scale persistent storage. SIGPLAN 35 (November 2000)

[13] Adya, A., Bolosky, W.J., Castro, M., Cermak, G., Chaiken, R., Douceur, J.R., Howell, J., Lorch, J.R., Theimer, M., Wattenhofer, R.P.: Farsite: federated, available, and reliable storage for an incompletely trusted environment. In: Proc. of the 5th OSDI (2002)

[14] Hasan, R., Anwar, Z., Yurcik, W., Brumbaugh, L., Campbell, R.: A survey of peer-to-peer storage techniques for distributed file systems. In: Proc. of ITCC (2005)

[15] Sun, Y., Liu, F., Li, B., Li, B., Zhang, X.: Fs2you: Peer-assisted semi-persistent online storage at a large scale. In: INFOCOM 2009, pp. 873–881. IEEE (April 2009)

[16] Yang, Z., Zhao, B.Y., Xing, Y., Ding, S., Xiao, F., Dai, Y.: Amazingstore: available, low-cost online storage service using cloudlets. In: Proc. of the 9th International Conference on Peer-to-peer Systems, IPTPS 2010 (2010)

[17] Toka, L., Dell'Amico, M., Michiardi, P.: Online data backup: A peer-assisted approach. In: 2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P), pp. 1–10 (August 2010)

[18] Seuken, S., Charles, D., Chickering, M., Puri, S.: Market design & analysis for a p2p backup system. In: Proc. of the 11th ACM Conference on Electronic Commerce, EC 2010, pp. 97–108. ACM, New York (2010)

[19] Graffi, K., Stingl, D., Rueckert, J., Kovacevic, A., Steinmetz, R.: Monitoring and management of structured peer-to-peer systems. In: IEEE Ninth International Conference on Peer-to-Peer Computing, P2P 2009, pp. 311–320 (September 2009)

[20] Pamies-Juarez, L., Garcia-Lopez, P., Sanchez-Artigas, M.: Rewarding stability in peer-to-peer backup systems. In: 16th IEEE International Conference on Networks, ICON 2008, pp. 1–6 (December 2008)