PodBase: Transparent storage management for personal devices

Ansley Post[†‡] Petr Kuznetsov[†] Peter Druschel[†]

[†]Max Planck Institute for Software Systems [‡]Rice University

1. INTRODUCTION

Personal electronic devices that include a large amount of storage are increasingly common. Already, many households use multiple mobile phones, digital cameras, MP3 players and gaming devices, in addition to desktop and notebook computers. Today, users must individually manage these devices to ensure the *durability* and *availability* of the data they store.

Ensuring that data is durable, or regularly backed up, is an onerous task even for a single home computer. As the number of devices increases, it is difficult for the user to ensure that no data is lost in the event of a loss or failure of any one device. Even with the help of device-specific maintenance software, the user must keep track of all devices that need to be backed up and perform the appropriate actions on a regular basis. Anecdotal evidence suggests that many users fail to ensure the durability of their data [5, 6]. Thus, users face the risk of data loss as they are increasingly dependent on digital information.

Making sure that data is available on the devices where it is needed is equally difficult. A user must regularly connect and synchronize devices to ensure, for instance, that changes to her address book are propagated to all communications devices and additions to her music library eventually is present on all devices capable of playing music. Currently, keeping devices synchronized is an inconvenient and error-prone task.

In this paper, we sketch the design of *PodBase*, a system that automatically manages the data and storage across a household's personal devices and frees users from the responsibility of manual data management. Data is automatically replicated to ensure both durability of data and availability of the latest data on relevant devices. The system operates transparently and takes advantage of available storage space and incidental connectivity that occurs among the devices.

Once a household's devices are introduced to PodBase, metadata is gossiped whenever devices are connected via a network, Bluetooth or USB. Moreover, during periods of connectivity, each device makes autonomous decisions about data replication. Through pairwise exchanges of data and metadata, the system makes progress towards ensuring availability and durability of data.

PodBase is self-managing and requires no oversight or input from the user during normal operation. It is completely decentralized and does not rely on the presence of any single device. Given sufficient storage space, the data stored on any device can be recovered automatically in the case of device loss or storage failure. For example, if a user loses his laptop, a recent snapshot of the data that was stored on it can be recovered. Replicated data stored on a device can also be accessed by the user, which provides additional availability.

We present evidence for the feasibility of our design using a storage trace gathered in 11 households, including over 40 storage devices, over a period of several months. The trace captures the interaction of storage devices as well as information such as storage device sizes, available capacity, and rate of data generation. Overall, the trace indicates that, relative to the rate at which new data tends to be generated, the design is feasible. That is, there is sufficient available storage space and devices are connected often enough to allow the transparent and timely replication of new data.

The rest of the paper is structured as follows: Related work is discussed in Section 2. Section 3 describes the target environment and states PodBase's goals. In Section 4, we sketch the design of PodBase. Section 5 studies the feasibility of PodBase using our trace data, and Section 6 concludes.

2. RELATED WORK

Automatically managing personal devices is a largely unsolved problem. Previous approaches can generally be divided into those providing naming and connectivity for personal devices, and systems providing additional storage functionality.

The Unmanaged Internet Architecture [3] provides a naming and routing service for a user's personal electronic devices. It requires zero con-

figuration by the user, and allows rich sharing semantics between users. In PodBase, we address the orthogonal problem of data management on users' devices. UIA can be used by PodBase to provide naming and connectivity among a household's devices regardless of their present location.

One way to ensure availability of data among a user's devices is a distributed file system that supports disconnected operation, like Bayou [13], Ficus [8] and Coda [4]. Attempts have been made to tailor distributed file systems to work with personal electronic devices. Ensemblue [7], the Few File system [9], Segank [11] and Perspective [10] determine which data should be available on a device based on the device's characteristics. Unlike Podbase, distributed filesystems support general write sharing, but they are not concerned with data durability. Unlike distributed filesystems, PodBase is layered on top of device-specific filesystems and can be easily deployed with existing devices.

The Roma personal metadata service [12] provides a shared metadata service for user data. Roma can be used to build higher-level services, including synchronization, consistency and increased availability. Roma relies on devices accessing a centralized metadata server, an assumption that we do not make in PodBase.

There is prior work on providing convenient backup service for user data. Pastiche [2] makes backup easier for users by automatically copying data to other machines in a peer-to-peer network. PodBase provides a similar service, but works with passive devices that are infrequently connected.

Time Machine [14] is an automatic backup utility included in Apple's OS X 10.5 operating system. It allows all Apple machines in a household to automatically backup to a single drive over the network. However, the backup device forms a single point of failure. PodBase is vendor independent, works with heterogeneous devices and does not depend on any single device.

3. ASSUMPTIONS AND GOALS

We now describe PodBase's intended operating environment and the properties it seeks to maintain.

3.1 Target Environment

PodBase is intended for a home environment. We consider a typical home with a set of shared personal electronic devices with mass storage. Now, we briefly describe the assumptions we are making about this environment.

- Most files are read-only. Most mutable files are modified by a single user. Concurrent updates of a file are rare.
- Every device is periodically connected to another device, such that information can eventually propagate among any pair of devices in the household via a series of pairwise connections. The connections can occur via a wireless network, Bluetooth or USB.
- Devices may be turned off when not in use, and it cannot be assumed that any one device will be online all of the time.
- A device may fail or be lost at any time. However, we assume that the failure or loss of multiple devices during a short period of time is unlikely.

An example of a household as described is a couple living in the same residence. Each partner has her own notebook computer, MP3 player, camera, and mobile phone. They share a large collection of read-only music, videos and pictures, and maintain mostly separate sets of documents that they occasionally work on.

3.2 System Goals

Our system aims to relieve users of the burden of explicitly ensuring *durability* and *availability* of their data. First, we want to make sure that all data is made durable, so that a single device failure or loss will not result in the loss of user data. Additionally, we want to establish convenient access of the data by making each device store the latest collection of data *relevant* to that device. For example, given enough storage space, each user should be able to access the entire shared collection of music files on their MP3 players and each communication device should store the latest version of the address book.

As an example, consider the household described above. Alice and Bob are graduate students, and both return home with their notebooks. At some point both Alice and Bob have their notebooks turned on. All the files Alice had edited at the library are replicated, in cryptographically sealed form, on Bob's notebook via their home wireless network. At the same time a replica of a CD that Bob has ripped earlier that evening is replicated to Alice's laptop so she can listen to it in the future. Alice connects her MP3 player to charge; at this time both users' latest additions to the music collection are copied to her player. In this simple example, all of the files modified during the day are made durable and available without requiring any explicit user action.

At a later date, Alice's laptop is stolen. She wishes to restore the data on the lost device's hard drive to her replacement notebook. When she connects over the wireless network with Bob's notebook, the files from her stolen notebook are restored on the new device.

An additional goal for PodBase is transparency: the use of PodBase should not affect the users' experience in unexpected ways. A file modified on one device should not cause the same modification on another device without the user explicitly overwriting the older copy of the file. For example, if Bob and Alice both make changes to their shared address book, these changes are not merged automatically; instead, PodBase creates two versions of the address book. If Alice or Bob wish to integrate the changes and merge the two versions, they must do so manually. Automatic reconciliation techniques can be added to PodBase, but are beyond the scope of this paper.

3.2.1 Desired Properties

Next, we specify the properties that PodBase attempts to maintain. The primary goal is to ensure durability. We want to guarantee that each file is replicated on sufficiently many devices. As a secondary goal, we want to maximize availability by placing copies of each file on devices where they are potentially useful, *subject to available space*.

Let Π be the set of participating devices and F be the set of files that are managed by the system. For each device i, let M_i denote the amount of space available at i for replication. Let k be the replication factor. For a set of files $S \subseteq F$, $size(S)$ denotes the amount of storage required to store S. We assume an *availability* map $\varphi : \Pi \rightarrow 2^F$ that assigns each device i a set of files that i should preferably store.

PodBase places at each device $i \in \Pi$, a set $\psi(i)$ of file copies so that the following properties are satisfied:

Durability. For each file $f \in F$, there exists a set of *at least* k devices that store copies of f: $\forall f \in F$: $|\{i \in \Pi : f \in \psi(i)\}| \geq k$.

Availability. Each device $i \in \Pi$ uses its storage for its preferred files first: $\forall i \in \Pi$: $M_i - size(\psi(i)) < size(\varphi(i) - \psi(i))$.

The choice of the replication factor k and the size of memory used for replication $\{M_i\}_{i \in \Pi}$ depends on the amount of available storage, the efficiency of the replication algorithm (sketched in Section 4), and the users' preferences.

PodBase defines default availability mappings between well-known file types and devices that are capable of interacting with these file types. For example, devices capable of playing MP3 files attempt to store all such files, if they have sufficient space. However, no replicas are normally stored on a camera, since its storage should be reserved for storing new pictures. Advanced users can modify φ to more finely control replica placement.

PodBase attempts to move towards a *goal state* where the above properties hold, even as the sets of files and devices are changing. Whenever a change is made, the system attempts to return to the goal state as quickly as possible, subject to the available storage space and the connections among devices that occur. With every pairwise connection, PodBase makes progress towards the goal state based on the available information about the system state. We want to ensure that with each connection, the number of copies of each file monotonically increases until the durability property is achieved. Likewise, we want to guarantee that progress is made towards the availability property.

4. DESIGN

Next, we sketch the design of PodBase. We describe how devices are introduced and removed from the system and how the storage on an individual device is used and managed by PodBase. Finally, we describe how changes to the system state are propagated and how new replicas are created during pairwise connections of devices.

4.1 Overview

PodBase distinguishes between *computing devices* and *storage devices*. Computing devices are capable of running user-installed software. Storage devices are devices such as hard drives, media players and mobile phones. Generally, a computing device contains at least one storage device, and additional storage devices can be connected via internal or external connections like Bluetooth or USB. Computing devices may connect to each other via network connections.

Computing devices run the PodBase *agent* software. In a PodBase deployment, there must be at least one computing device. The agent interacts with the user, provides connections between attached storage devices, and runs the algorithm for pairwise propagation of data using the metadata present on each storage device. Whenever two computing devices are connected or two storage devices are attached to the same computing device, we say that these two devices are connected.

PodBase uses connections between devices opportunistically to transfer metadata and to replicate data. For example, an MP3 player is connected to a desktop computer. During this period of connectivity, replicas are exchanged with the computer's disk as needed. Finally, the MP3 player is disconnected. At a later time, an external hard drive is connected to the desktop and replicas are propagated to this drive. Thus, through a series of pairwise interactions, data propagates through the system.

4.2 Membership

The set of storage devices used in a household can change over time, as users purchase new devices, and existing devices break or become obsolete. Devices must be explicitly added and removed from the system. When a new device is connected for the first time, PodBase asks the user if the device should be added to the set of storage devices it manages. Whenever two participating storage devices are connected, they exchange information about the other storage devices they know of and thus propagate membership information. A storage device may permanently disappear due to loss, permanent failure, or replacement of the device. In this case, the user can notify PodBase that the device is no longer available[1].

4.3 Storage Management

With PodBase, a storage device stores three types of data. It stores regular files that were explicitly stored by the user or applications, PodBase *metadata*, and PodBase *replicas*.

The metadata describes a device's most recent view of the system. Included in the metadata is the device membership, a logical clock for each storage device, and a list of all files that PodBase manages, along with their state. We assume that each storage device that is part of the system has enough free storage space to store the PodBase metadata.

Any space that remains after storing user data and metadata is used to store replicas of files to increase durability and availability. Each file is stored under its content hash, so that multiple copies of the same data are stored only once. Both the PodBase metadata and replicas are cryptographically sealed on disk using a key derived from a user password.

4.4 Device Interaction

Whenever two devices are connected, the PodBase agent attempts to move the system towards the goal state defined in Section 3.2.1. The pseudocode run by the agent is shown in Figure 1. A pair of devices first exchange a vector a logical clocks for each known device in PodBase. From this, they determine what subset of the metadata they should exchange to have the most up-to-date view of the system. Once this exchange is complete, file replication begins.

Based on the metadata, the agent determines the set of files that are insufficiently replicated and can be exchanged between the presently attached devices. Replicas made for durability have priority over those stored for availability; the latter may be deleted in favor of durability replicas. We use a simple, greedy replication algorithm, which increases the number of replicas monotonically until it is guaranteed that there are at least k copies of each file in the system.

[1]If a device has not been connected for a long time, the agent can optionally query the user for the device's status.

```
exchangeMetadataVersionVectors()
for each j ∈ Π do
  if myMDVersion(j) < remoteMDVersion(j) then
    updateMetaData(j)
  end if
end for
for each f ∈ F, f not stored locally do
  Let S_f be the set of devices storing f
  if |S_f| < k and M_i − M_i^d ≥ size(f) then
    storeDurable(f) {Delete availability copies, if necessary}
  end if
  if f ∈ φ(i) and M_i − M_i^d − M_i^a ≥ size(f) then
    storeAvailable(f) {Add an availability copy of f}
  end if
end for
```

Figure 1: Pseudocode at device i showing what happens when i connects to another device, here M_i^d and M_i^a denote the spaces on i occupied by durability and availability copies, respectively

This greedy algorithm may create more than k replicas of a file. Suppose a device i receives a copy of a file f and i is aware of $\ell < k$ other copies of f stored in the system. By the algorithm, i chooses to store f and (if $k - \ell - 1 > 0$) forwards f to $k - \ell - 1$ distinct devices, $j_1, \ldots, j_{k-\ell-1}$, in that order. In the worst case, each j_s is only aware of $\ell + 1 + s$ copies of f. A recursive application of this scenario results in propagating f along a *binomial* tree [1] of depth k rooted at the originator of the file. Thus, in the worst but very rare case, the algorithm may create up to 2^{k-1} copies of the file instead of the desired k. This bound must be used to choose appropriate values of k and $\{M_i\}_{i\in\Pi}$ to guarantee durability. Note that the algorithm is optimal for $k = 2$, and for practical values of $k > 2$, the bound is still reasonable. More efficient gossip-based replication algorithms are the subject of current work.

4.5 Data Recovery and Access

When a storage device fails, PodBase can recover the files that were stored on that device. The user informs a PodBase agent that she wishes to recover the data from a particular lost device onto a replacement device. From the metadata, PodBase obtains the list of files to recover, and then greedily restores these whenever a device is attached. The recovery happens automatically as devices interact. Users can speed the recovery process by connecting appropriate devices under the guidance of PodBase.

5. FEASIBILITY

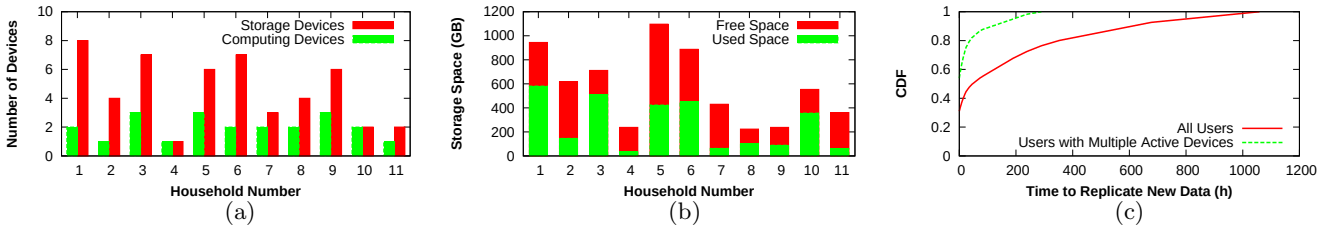

Figure 2: (a) Number hosts and storage devices per household. Each pair of bars represents one household. (b) Used and free space on storage devices. Each bar represents one household. (c) Time before generated data can be replicated in trace.

In this section, we explore whether PodBase's goals are feasible, given the set of devices, available storage, rate of data generation and frequency of device connections that occur in a given household.

5.1 Trace Collection

In order to judge the feasibility of our design, we need data about the intended deployment environment for PodBase. Unfortunately, there is little available data on the disk usage of personal devices such as desktops, notebooks and PDAs. Moreover, in addition to disk usage, our analysis also requires information about the frequency, duration, and bandwidth of connections between devices.

Since such data is not readily available, we gathered a trace by deploying a program on the devices of a small group of participating users for approximately two months. During this time period, each machine reported all storage devices that were attached either directly or via USB. For each device, we recorded the used storage, free space and total capacity at one minute intervals. We also periodically crawled the file system of each storage device to discover which files were stored and how often files were added, deleted, and modified in the file system. Additionally, we collected information about what type of network each computing device was connected to. The measurements were taken between August 18, 2007 and October 18, 2007 and included 11 households with 23 computing devices and 48 storage devices. There were 1568773 samples taken, and 971 disk crawls performed during this period.

5.1.1 Limitations

Ideally, we would like to have collected a trace from a large and diverse user community over a long period of time. However, logistical and privacy issues make this a difficult undertaking. We instead relied on the generous support of a small community of volunteers, namely the members of our research group and their friends and family.

In our group, at least one member of each household was a computer science researcher. Thus, there is a likely bias towards users who have an interest in technology and tend to surround themselves with electronic devices. They may also be more likely to accumulate a large amount of data. In the absence of better data, we believe that our trace data nevertheless gives a useful indication of the feasibility of our approach.

5.2 Results: Feasibility

In order to make effective use of PodBase, a user must have at least two devices capable of storing data. Figure 2(a) shows how many computing devices (desktop or notebook computers) and how many storage devices (hard drives, MP3 players, memory sticks) were present in each household during our trace collection. The results show that there are many households with more than one computer. Also, most households have additional storage devices beyond a single internal hard drive per computer. This shows that many households could benefit from automatic storage management.

Figure 2(b) shows the amount of free storage space. Some households have significant amounts of free space, others have very little. In particular, household 3 has over 70% of its storage space used. For eight of the eleven households there is enough free space available to provide data durability (with $k = 2$); the remaining households would have to add storage in order to get the full benefit from PodBase. These households could simply purchase an inexpensive 320GB USB disk and plug it in; the system would automatically use this new space to provide durability and availability.

Our result on the availability of free space is conservative, because it ignores data redundancy that exists on a device and across multiple devices. For example, there are nine MP3 players in our trace. An MP3 player typically contains a subset of the files that are already stored on one of the users' computers. Also, it has been shown that two machines running the same operating system have a significant overlap in data [2].

Recall that PodBase requires each device to store a copy of the metadata. In our study, the maxi-

mal metadata size is less than 200MB, which is insignificant relative to the capacity of the devices.

Next, we wish to gauge if there is enough connectivity to ensure that new data is made durable (with $k = 2$) in a timely manner. Adding more storage is relatively easy, since a user can simply purchase an inexpensive disk. However, if PodBase required users to connect devices often, this would place a considerable burden on users.

To address this concern, we use the free space, data growth rates, and connectivity measured in the trace to perform a simulation. We begin with the system in stable state, and then have each device generate new data at the average rate from the trace. When two devices can connect to each other (either directly or via the network), they transfer data at the nominal rate of their connections. When a device generates data, it attempts to replicate the new data by transferring it to another device that has free space. We wish to measure the amount of time it takes for generated data to be replicated once created.

To do this, we replay[2] the connectivity among the devices of a single household from our trace. When data is generated it is put into a queue; the head of the queue is transferred first when a connection is present. We measure the time in hours that data waits in this queue before being replicated. If, at the end of the experiment, data has not been transferred, it is included as having waited since its creation time. We repeat this experiment for all households and aggregate the data.

The lower curve in Figure 2(c) shows the results. Approximately 30% of data can be replicated within one hour. This case occurs when the creating device is connected to another device with free space at the time when data is generated. Around 50% of data can be replicated in under 48 hours. Some data could not be replicated for a much longer period of time; most of these cases are due to three of our households who either only had one device, or had multiple devices but only activated them to install our monitoring application and then did not use them again.

The second, higher, curve shows the results with these three users removed from the trace. For the remaining nine households in our trace, it is possible to get quick replication for newly created data.

In summary, for most of our user community automatic storage management is feasible: there are enough devices, storage space, and sufficiently many connections to achieve acceptable replication times. The remaining households could take advantage of PodBase if they were to purchase and connect an additional storage device.

[2]We begin the simulation at the time when all of the users' devices that generate data appear in the trace.

6. CONCLUSION

We have sketched the design of PodBase, a system that frees users from having to manage data on their personal devices. PodBase transparently ensures durability of data despite the loss or failure of a device, and increases availability of data on the appropriate devices. PodBase uses pairwise gossiping to propagate data and information about the system state. The system is decentralized and does not depend on the availability of any one device. PodBase takes advantage of existing free space and connectivity among devices. A trace-collection experiment within a small community indicates that the system is feasible. Going forward, we intend to deploy and evaluate PodBase in actual use.

7. REFERENCES

[1] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1999.

[2] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. *SIGOPS Operating Systems Review*, 36(SI):285–298, 2002.

[3] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. Persistent personal names for globally connected mobile devices. In *Proc. OSDI '06*, Nov 2006.

[4] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Trans. Comput. Syst.*, 10(1):3–25, 1992.

[5] Many PC Users don't backup valuable data. `http://money.cnn.com/2006/06/07/technology/data_loss/index.htm`.

[6] PC Pitstop Research. `http://pcpitstop.com/research/storagesurvey.asp`.

[7] D. Peek and J. Flinn. EnsemBlue: Integrating distributed storage and consumer electronics. In *Proc. OSDI '06*, November 2006.

[8] G. J. Popek, R. G. Guy, T. W. Page, Jr., and J. S. Heidemann. Replication in Ficus distributed file systems. In *Proc. WMRD*, pages 20–25, November 1990.

[9] N. Preguia, C. Baquero, J. L. Martins, M. Shapiro, Paulo, S. Almeida, H. Domingos, V. Fonte, and S. Duarte. Few: File management for portable devices. In *Proc. IWSSPS 2005*, March 2005.

[10] B. Salmon, S. W. Schlosser, L. B. Mummert, and G. R. Ganger. Putting home storage management into perspective. Technical Report CMU-PDL-06-110, Parallel Data Laboratory, Carnegie Mellon University, 2006.

[11] S. Sobti, N. Garg, F. Zheng, J. Lai, Y. Shao, C. Zhang, E. Ziskind, A. Krishnamurthy, and R. Y. Wang. Segank: a distributed mobile storage system. In *Proc. FAST '04*, March 2004.

[12] E. Swierk, E. Kcman, N. C. Williams, T. Fukushima, H. Yoshida, V. Laviano, and M. Baker. The Roma Personal Metadata Service. In *Proc. WMCSA 2000*.

[13] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proc. SOSP'95*, December 1995.

[14] Time Machine. `http://www.apple.com/macosx/features/timemachine.html`.