FEARLESS: Flash Enabled Active Replication of Low End Survivable Storage

Vamsi Kundeti
(vamsik@engr.uconn.edu)
CSE Department
University of Connecticut
Storrs, CT

John A. Chandy
(chandy@engr.uconn.edu)
ECE Department
University of Connecticut
Storrs, CT

Abstract

The reliability of data stored in a storage system is of immense importance for the users of the storage system and every storage system is expected to provide reliability. Along with reliability, storage systems are also expected to provide a degree of high availability. However, availability depends on the context in which the storage systems are used. Traditionally RAID (Redundant Array of Inexpensive Disks) has provided both reliability and availability very effectively. The massive increase in the usage of personal storage devices (like laptops, PDA etc..) has created concern with respect to the reliability and availability of these systems. These issues cannot be addressed by RAID because it is not feasible for this kind of portable personal storage devices. In this paper, we present a system which uses flash as part of the storage hierarchy to provide high reliability for low end personal storage systems.

1 Introduction

Reliability and *Availability* are the two most important factors for any storage system. However, the availability needs of a storage system are highly dependent on the context of the usage of the storage system. Applications such as servers demand high availability, but, in contrast, systems like desktops may not need that level of availability. Irrespective on the context of usage, reliability is always necessary for all type of storage systems. Traditionally, Redundant Array of Inexpensive Disks (RAID) has been used to provide both reliability and availability in storage systems. RAID achieves both of these by using extra disks (check disks) and replicating the data or parity to the check disks. If a RAID system has D data disks in a group of check disks the overhead of a RAID system is $\frac{1}{D}$. So, in a desktop system, where there is typically only space for 2 disks, the overhead of RAID is 100% - i.e. 1 data disk and 1 check disk. However, we assume that these desktop systems have access to a *backup service* which can perform periodic backups. The overhead can be reduced by mirroring only the modified data since the previous backup. The backup systems do provide redundancy; however, we do not count this as overhead since the backup systems exist in any case whether we use RAID or not.

This approach was considered in RAID0.5[3]. RAID0.5 uses a log disk in place of a mirror disk. Using a log disk reduces the seek time involved in mirroring. This reduction in the seek time makes RAID0.5 perform better than RAID1.

RAID0.5 is targeted mostly for desktop or enterprise systems (systems which can handle more than two disks). However, the reduced overhead comes at the cost of availability, since in a RAID0.5 system if the data disk fails the system cannot service the read requests to data which has not been updated from the previous backup window. Thus, the system must be deactivated upon failure even though there is no data loss. Although RAID0.5 seems attractive to personal storage systems the major drawback is the use of an extra disk (check disk) which personal storage systems cannot afford. In [2], the FEARLESS (Flash Enabled Active Replication of Low End Survivable Storage) system was proposed where a flash disk can be used for logging the data rather than a disk. In this paper we present an implementation of the FEARLESS system.

2 Reliability with FEARLESS

In this section we briefly describe how FEARLESS can provide reliability for personal storage devices. FEARLESS assumes that the personal storage systems (laptop, PDA etc.) have the potential of an additional secondary storage device like a flash or a SD card. Hybrid drives with both magnetic and flash storage can enable this architecture in a single device. The key assumption on which FEARLESS is built is that very little data on a disk is modified every day. HP's study of working set sizes showed that, on average, only 2% of available storage is written to over a 24 hour period [8]. The largest observed 24-hour write set size was just over 10%. Thus, in a given day the active data, the data that has changed since the last backup, is only 10% of the size of the disk, assuming the presence of a daily backup service. The FEARLESS system only logs (replicates) the active data.

FEARLESS assumes a single disk failure model - i.e only either the flash or data disk can fail at a given time. If the flash fails the data is safely in the data disk and if the disk fails the active data is present in the flash and the inactive Data can be recovered from the backup service. Thus we can always guarantee the reliability of data in FEARLESS with a single disk failure model. In [2], the $MTTDL$(Mean Time To Data Loss) was determined to be:

$$MTTDL_{FEARLESS} = \frac{MTTF_{Disk} * MTTF_{Flash}}{t_{r\_Disk} + t_{r\_Flash}} \tag{1}$$

where $MTTF_{Disk}$ is the mean time to failure of the disk, $MTTF_{Flash}$ is the mean time to failure of the flash device, t_{r_Disk} is the time to restore the disk after failure, and t_{r_Flash} is the time to restore the flash after failure. Note that t_{r_Disk} is only the time to restore the disk. In a FEARLESS system, since the system is powered off after a failure, there is almost no chance of a second failure while the failed disk is being replaced, so we do not need to include the time to replace the disk. The restore/reconstruction time of a FEARLESS system is determined primarily by the speed of the backup restore - tape can be slow but a D2D (disk-to-disk) backup system can be relatively fast.

The $MTTDL$ of a RAID1 is $\frac{MTTF_{disk}^2}{2 \times MTTR_{disk}}$. In the above if $t_{restore\_flash}$, $t_{restore\_disk}$, and $MTTR_{disk}$ are of the same order and likewise for $MTTF_{disk}$ and $MTTF_{flash}$ then $MTTDL_{fearless} \approx MTTDL_{RAID1}$. From this its clear that the reliability of FEARLESS and Reliability of RAID1 are nearly equivalent. The problem we have is with the availability. In RAID1 if a disk fails, the system is still available because we have the data mirrored on check disks, but in the case of FEARLESS if the disk fails the system has to be stopped and thus creating problems for the availability. However for personal storage systems reliability is of utmost important and availability is not as critical. Thus, the FEARLESS system trades availability for reliability and less storage overhead in personal storage systems.

Table 1 shows MTTDL, availability, and available storage for a single disk, a two disk mirrored system, and a single disk with flash replication. $MTTF_{Disk}$ is 300,000 hours which is typical for low-cost consumer disks. $MTTF_{Flash}$ for USB flash drives is roughly 50,000 hours. In spite of the normally high reliability of electronic components, USB flash drives have relatively high failure rates because of the failure propensity of the USB connector [6]. If the flash is not removable, as in a hybrid drive, the mean time to failure is around 2,000,000 hours. We assume a mean time to repair of 24 hours for the mirrored systems, and 36 hours for FEARLESS systems. $MTTR_{Flash}$ is assumed to be 19 hours. The availability is given in terms of "nines" - i.e. $-log_{10}(1 - A)$ where A is the fraction of time that the system is available.

The tables show that FEARLESS can offer equivalent or better MTTDL than RAID1 without the need for a second drive. With the high reliability of built-in flash, the MTTDL of FEARLESS is significantly better than mirroring. The disadvantage, however, is the availability of the FEARLESS system. While the availability of FEARLESS is good (99.9%), this is due entirely to the reliability of the disk and not due to the flash replication. Enterprise systems typically demand six nines of availability which can be delivered with mirrored systems, but is not possible with FEARLESS. Thus, the use of FEARLESS becomes a choice between high availability with low storage overhead and high availability with high storage overhead. As mentioned before, the ideal environment for such a system is for desktop or single-user systems that contain high value data but do not require 24/7 availability.

3 FEARLESS Implementation

From the previous discussion it is clear that the FEARLESS storage system needs to capture the active data continuously as it is being used. This data can be captured at either the block level or the file level. The advantage of doing the capture at the block level is that there is potentially less data that needs to replicated. The RAID0.5 system used a block level implementation [4]. In this implementation of the FEARLESS system, we instead use a file based system. The primary reason is that a file based system ties in well with a backup system. Incremental backup is also made easier since the changes since the backup are present on the flash drive.

We have used the FUSE (File System in Userspace) framework to implement a prototype version of FEARLESS. FUSE allows pseudo file systems to be developed in user space instead of the kernel. The FUSE library provides an API whereby the pseudo file system can provide callbacks from the kernel which override most of the VFS calls such as `open`, `read`, `write`, etc. In FEARLESS, the VFS calls to `open`, `write`, `mkdir`, `rename`, and other calls that change a file or the file system are intercepted and these changes is replicated into flash. The modified `open` VFS callback is illus-

Table 1. MTTDL and Overhead. ($MTTF_{Disk}$ = 300,000 hours, $MTTF_{USB_{Flash}}$ = 50,000 hours, $MTTF_{Builtin_{Flash}}$ = 2,000,000 hours, $MTTR_{Disk}$ = 24 hours, $MTTR_{Flash}$ = 19 hours)

Configuration	MTTDL (years)	Availability
Single Disk	5.7	3.32
Mirror	213895	6.94
FEARLESS with USB Flash	39794	3.47
FEARLESS with Built in Flash	1591773	3.98

trated in Algorithm1. This open VFS call back tries to check if the file is opened in *write* or *append*, *creat* mode. If a file is opened in any of these modes, then it is a potential candidate for creating active data. In this case, the FEARLESS system creates an entry for this file on the flash in the form of a *LookUpTable* where the key is the file descriptor of the disk file which is being opened. Since the write call is called with the disk file descriptor (fd_{disk}), FEARLESS can find the corresponding file descriptor for the file in flash ($fd_{fearless}$) and updates that file (in flash) with the active data. This is done along with the write to the actual disk file. Flash replicas are kept in an active directory on the flash mount. The Algorithm2 illustrates the outline of the modified VFS write callback.

Algorithm 1: open VFS call back in FEARLESS

```
INPUT: File Path and File mode
if open file mode is write or append then
    if file is not in flash then
        copy_file_into_flash(path) ;
    $fd_{fearless} = open\_fearless\_file(path)$ ;
    $fd_{disk} = open\_disk\_file(path$ ;
    $LookUpTable[fd_{disk}] = fd_{fearless}$ ;
else
    do a normal file system open
```

Algorithm 2: write VFS call back in FEARLESS

```
INPUT: file descriptor of the disk file and data
       buffer
$fd_{fearless} = LookUpTable[fd_{disk}]$ ;
$len = write\_to\_fearless(fd_{fearless}, buffer, size)$ ;
$len = write\_to\_disk(fd_{disk}, buffer, size)$ ;
return $len$ ;
```

4 Backup process in FEARLESS

In this section we briefly describe how the backup process works in FEARLESS. Since the FEARLESS system logs all the active data, the backup process is highly improved since the incremental changes since the last backup are already determined. Thus, there is no reason to perform a file system scan to find all new files. A simple backup strategy can simply copy the data in flash into the backup store. In order to easily maintain the file tree structure, one could use the "tar" utility to archive the flash disk and then regenerate it on the backup system. In order to minimize the amount of data transfer, a utility such as rsync can used to identify small changes within a file.

Before initiating a backup, the user or backup utility sets a special extended attribute, start_backup, on the flash file system. This will initiate a pre-backup procedure. The first step is to simply rename the active directory on the flash mount to a backup directory. This essentially takes a snapshot of the active data. Open files need to be preserved in the active directory so that running applications will still function. To enable this, the pre-backup procedure will move any open files to the active directory and then the file is copied back to the backup directory. FEARLESS keeps track of open files through the *LookUpTable*. Instead of the move and copy, the same process could be achieved by doing a hard link. However, many flash file systems do not support hard links, so the move and copy is more portable.

Once the pre-backup procedure is complete, the user level backup procedure can start. As mentioned before, this can be tar, rsync, or any other file level backup process. A utility that is customized to FEARLESS can be optimized to take advantage of the incremental changes on flash. Since the backup process is operating on the backup directory, normal FEARLESS operation can continue on the primary disk and the flash active directory. After the backup process is complete, the user can then initiate a post-backup procedure by setting the finish_backup extended attribute. This process simply removes the backup directory.

We have used the extended attribute feature to initiate the pre- and post-backup procedures. Though many flash file systems may not support extended attributes, these set attribute calls are intercepted before

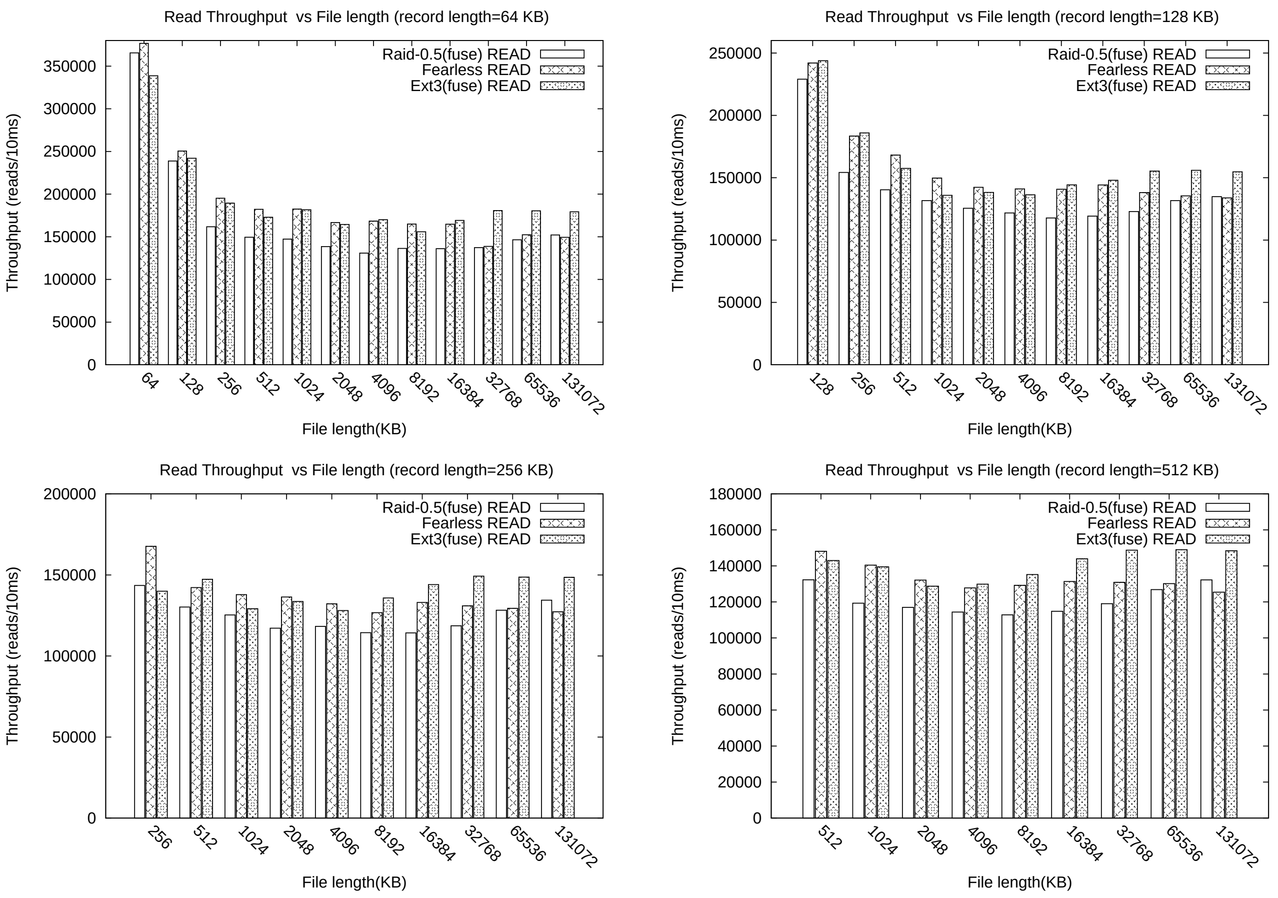

Figure 1. IOZone results for reads

Table 2. Compilation Times (sec)

	FEARLESS	FUSE_XMP	FUSE_RAID0.5
`bash`	25.3	23.3	24.19
`gcc`	129.9	119.2	129.26

reaching the underlying file system, so that is not an issue. However, one could conceivably implement these procedures as ioctls on the flash file system. At this point, FUSE does not support ioctls, so we could not use this option. However, a kernel VFS implementation could support ioctls to initiate the pre- and post-backup procedures.

5 Experimental results

Since FEARLESS performs an extra write to the flash for every write to disk, there is the potential of performance degradation. In this section, we present some results that show that the overhead of using FEARLESS is minimal. We have verified FEARLESS using a 40 GB hard disk as the primary drive and a 2 GB iPod as the flash drive. The hard drive is formatted with the ext3 file system and the iPod uses a HFS file system.

Our first set of experiments use the IOZone benchmark [1]. IOzone measures raw filesystem I/O performance on a set of read/write access patterns. Since our implementation is based on FUSE, there is significant overhead due to using FUSE but not due to our FEARLESS algorithm. FUSE_XMP is a bare-bones FUSE file system where every VFS call is passed through to the underlying ext3 file system. As an indication of the overhead of FUSE, FUSE_XMP has less than 30% of the performance of the underlying ext3 file system. Therefore, we have used FUSE_XMP as a base comparison to remove the effects of the overhead of the FUSE framework. If FEARLESS was implemented within the kernel at the VFS layer, the overheads would be similar. Since FEARLESS is a flash based implementation of a 2 disk RAID0.5 system, we have also included a comparison of flash-based FEARLESS system with a FUSE RAID0.5 disk-based system.

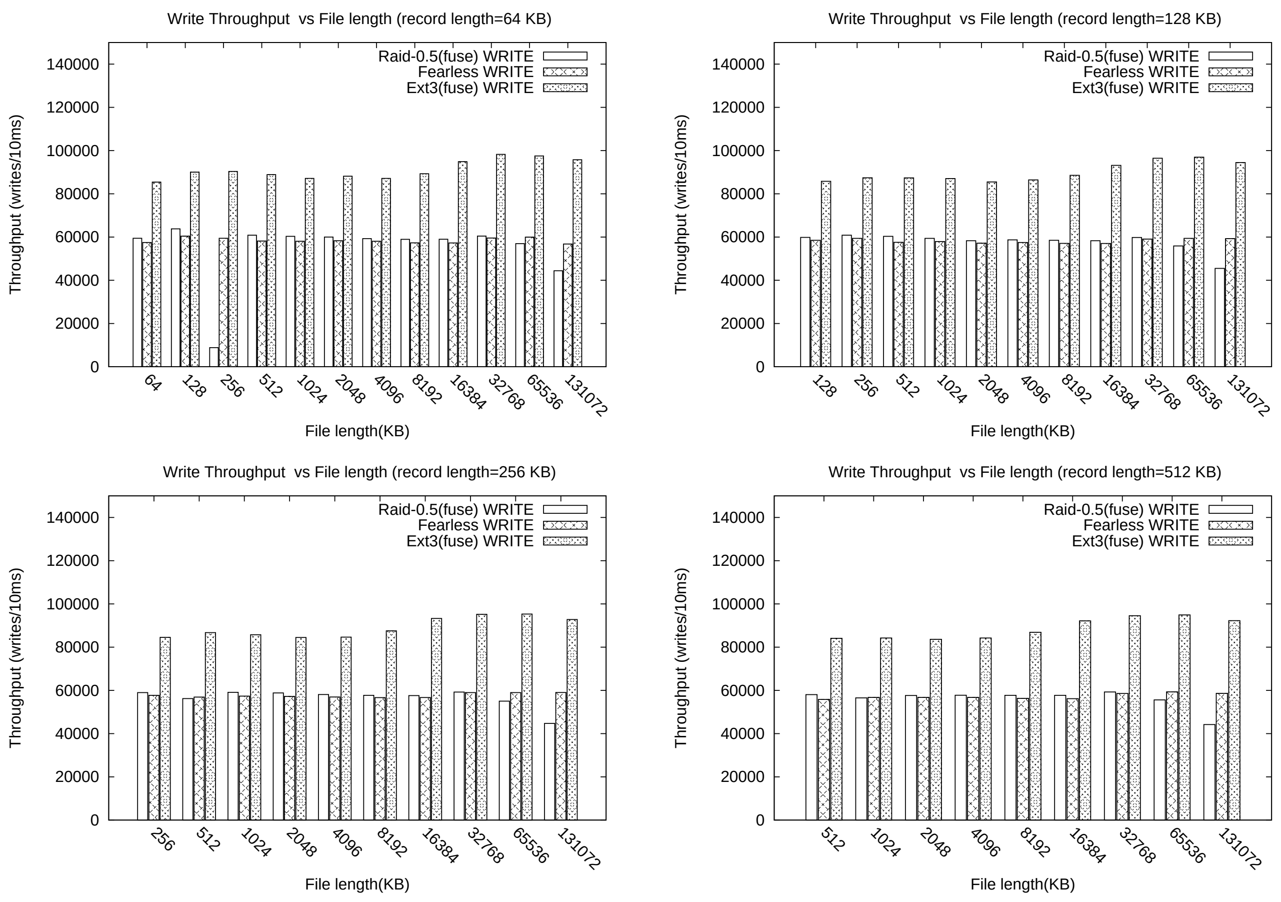

Figure 2. IOZone results for writes

Figures 1 and 2 show experimental results from the IOZone [1] benchmark. The figures show read and write throughput for different file sizes (X-axis) and different record lengths. The record length is the transfer size of each access. Looking at the figures, we find that for reads, FEARLESS performs comparable for reads with FUSE_XMP (1% difference). However, for writes, FEARLESS is on average 29% worse compared with FUSE_XMP because of the extra write into the flash. The read overheads are manageable though the write performance is slightly worse. In the personal storage environment that we are targeting FEARLESS, this is a not significant issue since disk performance is not a critical factor and reads are much more common than writes. In comparison with RAID 0.5. we see that the flash based FEARLESS system performs better than the disk-based RAID 0.5 system because of the faster flash read performance. As expected, write performance is slightly better on the HDD-based RAID-0.5 system, but the difference is not significant.

IOZone is a measurement of raw throughput, which may not be an ideal metric for personal storage systems. Therefore, we have looked at the effects of FEARLESS on application performance - namely compilation. Table 2 show the times required to compile `bash` and `gcc` on FEARLESS and FUSE_XMP. The average overhead is 8% compared to FUSE_XMP and 2.5% compared to RAID-0.5. Once again, the FEARLESS impact on performance is minimal.

6 Related work

The FEARLESS system that we have presented is most similar to disk mirroring techniques from an implementation point of view. The distinction is that the flash mirror does not fully mirror the primary disk. FEARLESS differs particularly in the fact that the flash is not only used as a redundancy store but also as cache to off-line backup media.

The use of removable storage as a cache has been recognized in distributed storage and pervasive storage research. For example, BlueFS uses removable storage as part of a cache hierarchy in a distributed file system [7]. Similarly, Tolia extended the Coda file system to support portable storage devices through lookaside caching built on file recipe hashing [10]. The PersonalRAID system is a portable storage solution where the storage device is the central personal storage device and provides synchronization with local storage [9]. The distinction between these systems and our system is that FEARLESS caching is designed for redundancy - in other words the cache is a redundancy cache of a backup system rather than a performance cache of a distributed file system.

FEARLESS is intimately tied into judicious use of backup and systems that provide seamless online backup can ease this process. Tape backed systems are inherently difficult to use and are not often used in personal storage systems. However, there have been recent efforts to automate the backup process including research efforts such as Pastiche [5] which uses peer-to-peer systems to provide backup storage and commercial efforts in disk-to-disk backup. Online backup storage services such as Moby and Carbonite can also be used in conjunction with FEARLESS.

7 Conclusions

In this work we implemented a storage system that uses flash in conjunction with backup systems to provide high reliability at the expense of availability. The convenience of flash make it a more appropriate solution for redundancy compared to redundant hard disks. Our results indicate that we can achieve reliability for personal storage systems equivalent to RAID1 with little impact on read throughput and only 29% overhead on write throughput. For a real application such as compilation, the overhead is only 8%. The work demonstrates a practical application of integrating nonvolatile memory such as flash into the storage hierarchy.

References

[1] IOZone File System Benchmark. [`http://www.iozone.org`].

[2] J. Chandy and S. Narayanan. Reliability tradeoffs in personal storage systems. In *ACM SIGOPS Operating Systems Review*, pages 37–41, 2007.

[3] J. A. Chandy. RAID0.5: Active data replication for low cost disk array data protection. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2006.

[4] J. A. Chandy. RAID0.5: Design and implementatino of a low cost disk array data protection architecture. *Journal of Supercomputing*, 46(2):108–123, Nov. 2008.

[5] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[6] R. Kozlov and H. Heshes. Extending the warranty period of the DiskOnKey 500 series: Reliability report. White Paper 04-WP-0604-00, Rev 1.0, M-Systems, Newark, CA, Aug. 2004.

[7] E. B. Nightingale and J. Flinn. Energy-efficiency and storage flexibility in the Blue file system. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2004.

[8] C. Ruemmler and J. Wilkes. A trace-driven analysis of working set sizes. Technical Report HPL-OSR-93-23, Hewlett-Packard, Palo Alto, CA, Apr. 1993.

[9] S. Sobti, N. Garg, C. Zhang, X. Yu, A. Krishnamurthy, and R. Y. Wang. PersonalRAID: Mobile storage for distributed and disconnected computers. In *Proceedings of USENIX Conference on File and Storage Technologies*, Jan. 2002.

[10] N. Tolia, J. Harkes, M. Kozuch, and M. Satyanaranan. Integrated portable and distributed storage. In *Proceedings of USENIX Conference on File and Storage Technologies*, Mar. 2004.