UNIVERSITY of CALIFORNIA
SANTA CRUZ

MRAM – PRELIMINARY ANALYSIS FOR FILE SYSTEM DESIGN

A project report submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Alicja Beata Szczurowska

March 2002

The project of Alicja Beata Szczurowska is approved:

Professor Scott A. Brandt

Professor Darrell D. E. Long

Table of Contents

List of Figures

List of Tables

Abstract

MRAM – Preliminary Analysis for File System Design

by

Alicja Beata Szczurowska

Recent improvements in nonvolatile memory technology show a promise for the replacement of currently used DRAM. Magnetic memory (MRAM) with its low access times and comparable cost per byte to DRAM is a good candidate. It can be effectively used to improve system performance through appropriate file system design. Long *et al.* propose HeRMES [7] an MRAM-based file system. This project report presents the analysis of what might be gained through placement of the file metadata in MRAM.

Dedicated to

my parents and my fiancé Karl,

for their love and patience

Acknowledgements

I wish to thank my advisor, Scott Brandt, for his guidance and ongoing support and understanding, Darrell Long and Ethan Miller for their helpful feedback. Also thanks to my fellow students in the CSRG group, especially Karen Glocer and Feng Wang, and my fiancé Karl Brandt for their valuable advice and moral support.

1 Introduction

In recent years, the relative disk I/O performance has gotten worse when compared to the constantly increasing processor speeds. With disk as the primary source of stable storage there was only so much that could have been done to improve file system performance. This situation could drastically change with the introduction of magnetic RAM (MRAM), a new nonvolatile memory technology. MRAM, with its cost per byte and access times comparable to those of DRAM can be considered as a good candidate for its replacement. A prototype 256 Kb chip has already been introduced by Motorola [10].

Nonvolatile memory can be used to improve file system performance in many ways. One of them is to store all the file system metadata (bookkeeping information about data) in MRAM as proposed by Long *et al.* [7]. HeRMES (High-Performance Reliable MRAM-Enabled Storage), bears a promise of greatly reducing the size of metadata and increasing file system performance. This is a very tempting idea, but since it is a new approach in file system design, it is not yet known just how much can be gained from it.

This project report presents a preliminary analysis of what can be gained from implementing metadata in nonvolatile memory in the HeRMES file system. The analysis shows that HeRMES should be expected to outperform a typical disk-based file system by 70.1%, and the average i-node size per file will shrink

from about 1323 bytes to 379 bytes with 1K block sizes and from 1396 bytes to 290 bytes with 4K blocks.

The current nonvolatile RAM technology will be discussed in Section 2. Section 3 will cover related research, Section 4 will describe necessary information about UNIX file system necessary to understand the different design approach of an MRAM-based file system presented in Section 5 and analyzed in Section 6. The results of the analysis are presented in Section 7, followed by conclusions and proposed future work.

2 Related Work

File system research to date has mainly concentrated on improving performance when the only source of nonvolatile storage was the disk. This was done on two levels. On disk level, Iyer *et al.* proposed anticipatory disk scheduling [6] in an effort to increase disk I/O performance. Berkeley Fast File System (FFS) [9] increases the block size to increase throughput and introduces cylinder groups for better locality. Ganger [4] tries to improve on metadata operations by so-called *soft updates* basing on delayed writes on most file operations.

Many existing or researched in-memory file systems, for example Phoenix [3] are those that can be used in small diskless units such as battery powered PDAs. Wu proposed eNVy [14] a memory-based file system, but it still takes advantage

of disk, treating it as a rescue in case of metadata overflow or to protect against memory failures. Conquest, a disk/persistent-RAM hybrid file system presented by Wang [13] uses nonvolatile RAM as a final storage destination for metadata and files smaller than a set threshold (currently 1 MB), including executables and shared libraries. A fixed threshold, however, may cause this file system to be less adaptive to the changes in memory technology. Conquest also assumes the availability of cheap 1 to 4 GB persistent RAM and seems to be geared more toward the battery-backed RAM available today.

3 Nonvolatile RAM Technology

There are two distinct types of storage devices, rotating and non-rotating devices. The first, which includes the hard drive, is burdened by high access times of a few miliseconds due to seek time rotational latency.

Comparing the read and write times for DRAM and MRAM [11] in Table 3.1 it is clear that MRAM has the potential to be as fast as DRAM. It also has the added advantage of being nonvolatile, which the former cannot guarantee. DRAM is a solid state memory where bits are stored as a charge on a capacitor. Therefore as long as there is charge the data remains in memory. However, as soon as the power supply is switched off the data content of a DRAM chip is lost. Its popularity lies primarily in very low access times of approximately 10 ns.

	DRAM	Flash EEPROM	FeRAM	MRAM
Non-Volatility	No	Yes	Yes	Yes
Write Time	10 ns	≥ 1 μ s	30 – 200 ns	10 – 50 ns
Read Time	10 ns	20 – 120 ns	30 – 200 ns	10 ns – 1 μ s
Rewrite Cycles	10^{15}	10^5	10^{12} – 10^{15}	10^{15}
Data Retention Time	Zero	10 years	10 years	10 years

Table 3.1: Comparison of different RAM technologies [11].

MRAM may be at a slight loss here with its write time up to 50 ns and read time up to 1 μs.

Compared to FeRAM and flash EEPROM [2], two of its nonvolatile competitors MRAM certainly has a promise for faster read and write times. It is clear from Table 3.1 that flash memory write times are unacceptably long, more than 1 μs, to be considered for a replacement for DRAM. Also, its data retention time is far too low. Flash deteriorates after 10^5 rewrite cycles to the same address which is unacceptable for persistent data storage. FeRAM just like MRAM is nonvolatile but has lower, though comparable read and write times. However MRAM has a major advantage over FeRAM. It has a higher promise to be denser and so less expensive [2]. This is an important feature that could make MRAM more attractive and potentially place it in the mainstream of memory technology.

Based on the presented characteristics MRAM certainly has the potential to replace both flash memory and DRAM. It has the potential to guarantee fast writing speed and a high write endurance.

4 Unix File System

A file system is responsible for the structure and control of file storage and is in fact an important component of any operating system. It is necessary for all the data to be stored in a persistent manner and so far the most reliable medium for persistent storage has been the disk. It is cheap, has a large storage space and the stored data is retained even when the power is shut off.

A hard disk is a block device. Its basic storage unit is a disk sector (physical block) and it reads and writes data measured in logical blocks, which are usually some multiple of a disk sector. Currently the size of a block is usually 1K used by Linux *ext2fs* (second extended file system), 4K in Berkeley FFS [9] and the size of a disk sector is currently 512 bytes.

The layout of the file system on disk usually follows the same pattern. The first block of the file system is reserved for the boot block which stores the machine code to bootstrap the OS. The second block is the super-block followed by a list of i-nodes. The file system also keeps information about the usage of disk blocks, a free block list and a free i-node list. This physical and conceptual information about file data is often referred to as metadata.

4.1 The Super-block

The Super-block is the second block in the file system, as stored on disk, after the boot block and it describes the layout of the file system. It holds the information about the size of the file system, a list and number of free blocks, index to first free block in the list, size of the i-node list, number and list of free i-nodes, index to the next free i-node in the list, locks for free block and i-node lists, a flag to indicate if a super-block has been modified and of course the i-node number of the root directory [8].

4.2 I-node Structure

The i-node represents a file on disk by holding its administrative information and addresses of data blocks on disk. The number of i-nodes in the file system is determined at time of its creation; this process is often referred to as static i-node allocation. The default is to allocate one i-node for every 2K of disk space [8].

FreeBSD implementation of an i-node [1] together with byte storage information of its every component can be found in Figure 4.1. It can be seen that even though 512 bytes are allocated for each i-node on disk, less than 200 bytes are actually used to store data and less than 50 bytes to store information about the file.

```
struct      dinode
{
   u_short     di_mode;    /* 0: mode and type of file */
   short       di_nlink;   /* 2: number of links to file */
   uid_t       di_uid;     /* 4: owner's user id */
   gid_t       di_gid;     /* 6: owner's group id */
   union
   {
       /*  u_quad_t v; */
       u_long val[2];
   } di_qsize;                 /* 8: number of bytes in file*/
   time_t      di_atime;       /* 16: time last accessed */
   long        di_atspare;
   time_t      di_mtime;       /* 24: time last modified */
   long        di_mtspare;
   time_t      ctime;          /* 32: last time inode changed */
   long        di_ctspare;
   union
   {
       struct
       {
           daddr_t di_udb[NDADDR]; /* 40: disk block addresses */
           daddr_t di_uib[NIADDR]; /* 88: indirect blocks */
       } di_addr;
       char di_usymlink[MAXRASTLINK + 1];
   } di_un;
   long      di_flags;       /* 100: status, currently unused */
   long      di_blocks;      /* 104: blocks actually held */
   long      di_gen;         /* 108: generation number */

   #define DI_SPARE_SZ 4       /* 112: spare for 4 longs */
   u_long di_spare[DI_SPARE_SZ];   /*reserved (unused) */
};
```

Figure 4.1: FreeBSD Code for an I-node [1]

5 MRAM-based File System

Currently used file systems are optimized for systems where the disk is the only nonvolatile storage medium. However, the new MRAM technology presents a new challenge for file system design. Long *et al.* [7] present a concept for a new file system HeRMES, which would make effective use of the nonvolatile MRAM. Because of its cost characteristics, which are similar to DRAM, MRAM currently cannot be considered as a replacement for disk. However it can certainly replace DRAM and be used to improve the file system performance.

The most important feature of HeRMES is that it would use MRAM to permanently store the file system's metadata. This should greatly improve file system performance as metadata operations would not suffer from slow disk accesses. Since all metadata will be stored in persistent memory, there are a few changes that can be made to the structure of the file system. Those would involve changes to the super-block and i-node design.

5.1 Super-block Design

Separation of metadata from file data (still stored on disk) eliminates the need for static i-node allocation. In HeRMES there is no need to reserve space on disk for i-nodes, which can now be created dynamically when a file is created, as long as there is available memory space in MRAM, or updated upon file modification.

Only i-nodes for existing files will be stored in MRAM and that eliminates the concept of a free i-node. Therefore the super-block will no longer store information related to free i-nodes such as the number and list of free i-nodes, index to the next free i-node in the list or the lock for free i-node list. Also, naturally, the super-block size is not constrained by the physical block size since MRAM can access data in units of bytes or words.

5.2 I-node Design

Since i-nodes are no longer stored on disk, their size is not determined or constrained by the size of the disk physical block. Therefore, it can grow beyond the block-enforced 512 bytes. This eliminates the need for indirect blocks since one i-node can now store all of its file's data block addresses. Figure 5.1 shows the proposed code for a HeRMES i-node.

```
struct      dinode
{
   u_short     di_mode;    /* 0: mode and type of file */
   short       di_nlink;   /* 2: number of links to file */
   uid_t       di_uid;     /* 4: owner's user id */
   gid_t       di_gid;     /* 6: owner's group id */
   union
   {
       /*  u_quad_t v; */
       u_long val[2];
   } di_qsize;                /* 8: number of bytes in file*/
   time_t      di_atime;       /* 16: time last accessed */
   time_t      di_mtime;       /* 20: time last modified */
   time_t      ctime;          /* 24: last time inode changed */
   long        di_flags;       /* 28: status, currently unused */
   long        di_blocks;      /* 32: blocks actually held */
   long        di_gen;         /* 36: generation number */
   u_long      di_naddr;       /* 40: number of address blocks */
   union
   {
       daddr_t di_udb[di_naddr]; /* 44: disk block addresses */
       char di_usymlink[MAXRASTLINK + 1];
   } di_un;

};
```

Figure 5.1: Proposed HeRMES Code for an I-node

6 Method of Analysis

As with every new file system concept, especially if it involves big changes to the way data is stored, there is a need to perform a feasibility analysis. There are a number of questions to be asked:

1. How much metadata is in a file system?
2. How much would it shrink if stored in memory?
3. What would be the performance benefit?

In HeRMES the boot block and super-block will have constant size in memory and would depend on their implementation. Information about free blocks, on the other hand, may vary depending on the chosen implementation method. Also the space taken by the i-nodes would change dynamically depending on the number and size of files in the system.

6.1 I-node Space Measurement

To answer the first question with regards to the new proposed file system it is necessary to determine the file size and disk utilization distributions for an average existing file system. A Unix File System Survey [5] performed in 1993 by Irlam provided the initial data, which proved very useful for further analysis. Perl scripts were also used to scan the hard drives of computers running Linux and Windows

operating systems. The retrieved data consisted of the number of files of specified sizes and the amount of space on disk used to store those files. The code for both scripts (UNIX and Windows platforms) can be found in Appendix A.

Based on the file size distribution it was not difficult to calculate the average size of an i-node per file for a typical UNIX file system and for HeRMES. The calculations for the UNIX file system were based on the FreeBSD i-node implementation, which has twelve 4 byte addresses for direct blocks and three 4 byte addresses for indirect blocks. For a HeRMES i-node all addresses were assumed to be 4 bytes. The first step was to determine the size of an average file from each file size group in terms of the number of blocks needed to store it.

$$number\ of\ blocks = \frac{average\ file\ size}{block\ size} \tag{6.1}$$

where the average size of a file is *cumulative space / number of files*

The next step was to estimate the size needed to store metadata for that file. In a typical UNIX system this would include i-nodes and possibly indirect blocks. To get numerical values it is necessary to use the byte values provided with the code for the FreeBSD i-node in Figure 4.1. In HeRMES all file metadata is stored in i-nodes and their size depends mostly on the number of block addresses for a specific file. Code with byte values necessary to calculate the average i-node sizes is provided in Figure 5.1.

Disk utilization distribution from Figures 7.1 to 7.4 was necessary to evaluate how much space in MRAM would be required to store all the i-nodes for a fully used disk. Based on the percentage of space used by files of a specific size it can be estimated how many i-nodes of what size would be needed for a given disk.

The first step was to determine what percentage of a given disk would be dedicated to store the files from a given file size group. Then the number of files (and therefore also i-nodes in HeRMES) for that part of the disk would be:

$$number\ of\ files\ =\ \frac{fraction\ of\ disk\ used}{average\ size\ of\ file} \tag{6.2}$$

Then the space necessary for all the i-nodes would be

$$i\text{-}node\ space = \sum_{all\ file\ size\ groups} number\ of\ files\ \times\ i\text{-}node\ size \tag{6.3}$$

6.2 File System Performance Measurement

According to Ruemmler and Wilkes [12] analysis the majority of disk accesses are writes, and most of them to metadata. Figure 6.1 presents the graphical representation of their results. Writes to metadata constitute for about 40% of all disk activity. This value may change between systems, but is rather consistent compared to the percentage of reads. However, precise values for reads are not crucial to the correctness of this analysis as most of the reads may be cached and therefore will not require disk accesses.

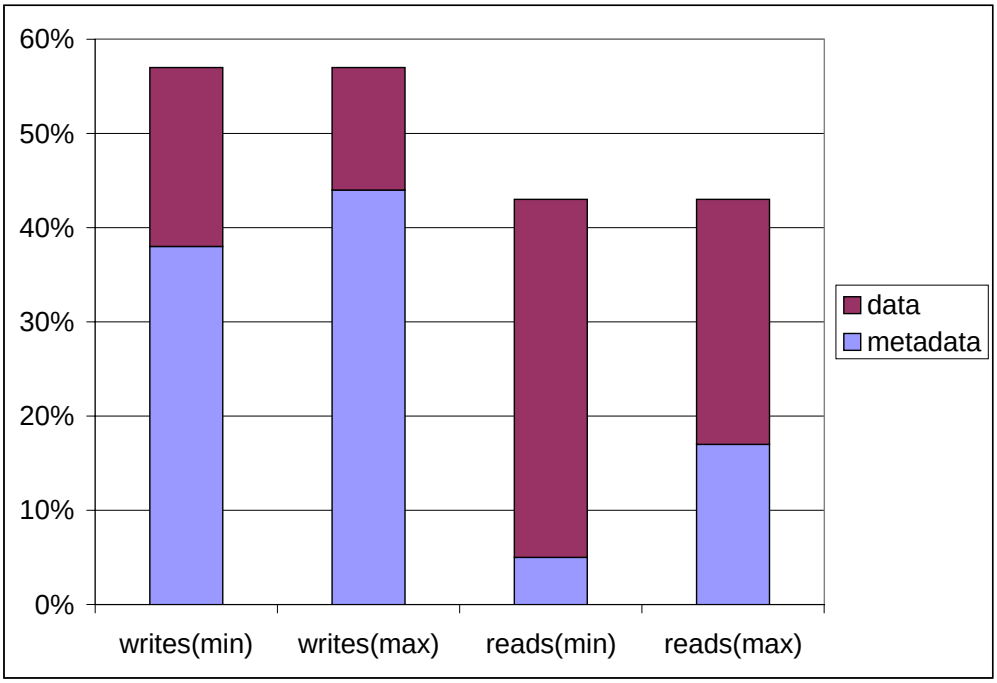

Figure 6.1: File System Activity based on Ruemmler and Wilkes study [12]

The expected speedup for HeRMES was calculated as follows:

$$speedup = \frac{average\ access\ time\ for\ disk\text{-}based\ FS}{average\ access\ time\ for\ HeRMES} \tag{6.4}$$

Since UNIX heavily relies on caching to improve performance I decided to assume that all the data for files to be read will be cached, even though only the recently accessed blocks and the i-nodes of open files can actually be found in the cache. This would in fact be a worst case scenario for MRAM-based file system. Therefore:

$$average\ access\ time\ for\ disk\text{-}based\ FS\ =$$

$$\%writes \times average\ disk\ access\ time\ +$$

$$\%reads \times DRAM\ read\ time$$

Note that for the calculation of the above average access time it is not necessary to differentiate between writes to data and to metadata. This is not the case when dealing with HeRMES because the two types of writes are treated differently, which is apparent in the formula below. Reads are still treated uniformly.

$$average\ access\ time\ for\ HeRMES\ =$$

$$\%writes\ to\ data \times average\ disk\ access\ time$$

$$+\ \%reads \times MRAM\ read\ time$$

$$+\ \%writes\ to\ metadata \times MRAM\ write\ time$$

7 Results

The following subsections present the results of the conducted study. Section 7.1 presents a graphical representation of the collected data, as well as an analysis of the changes in file sizes and disk usage between the years of 1993 and 2002. Those results were used to estimate the required space in MRAM for the file metadata in Section 7.2. Section 7.3 covers a brief performance analysis.

7.1 File System Scan Results

Figures 7.1 to 7.4 present the results of the conducted file system scans. By comparing the first three figures it is clear that file sizes are becoming larger. In 1993 most files were in the range 2 – 8K; in 1995, although most files are still in the same range, there is definitely an increase in the number of larger files, 16 – 128K. Currently, those larger files are the most numerous in a file system, and most of them are user files. Files constituting for an operating system are still small as can be seen from Figure 7.4.

This change in file size distribution has a definite effect on the change in disk space use. In 1993 most files were small and they definitely did not correspond to the space on disk they actually used (Figure 7.1). There was a definite disparity between the number of files of a certain size and the space they occupied on disk. This disparity is not as visible in currently used systems. As Figure 7.3 shows the

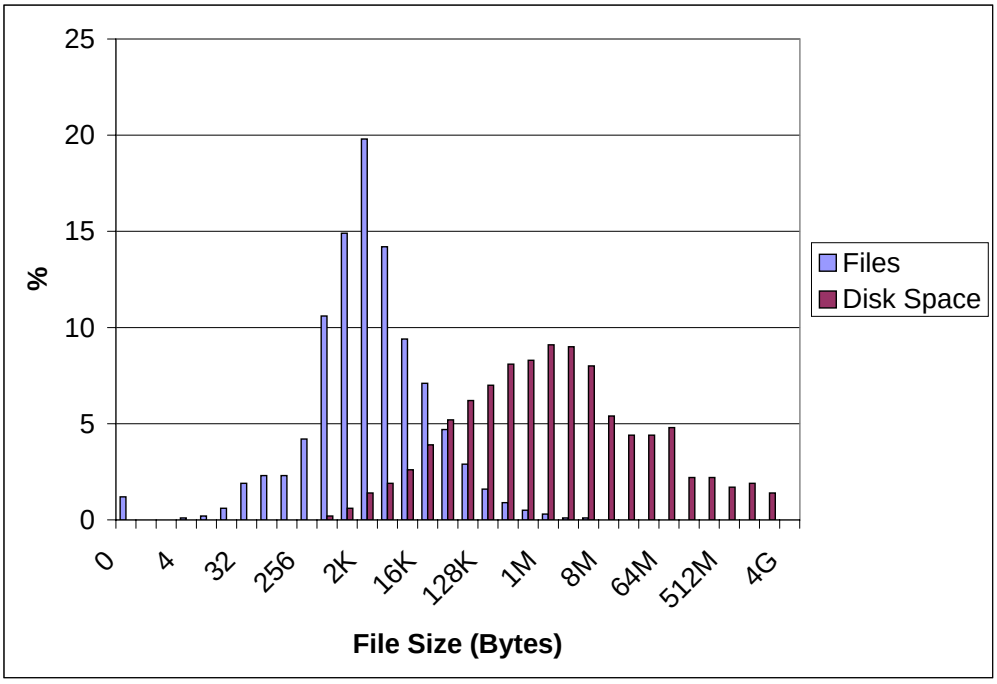

Figure 7.1: File Size vs. Disk Space Distribution from a UNIX study conducted in 1993.

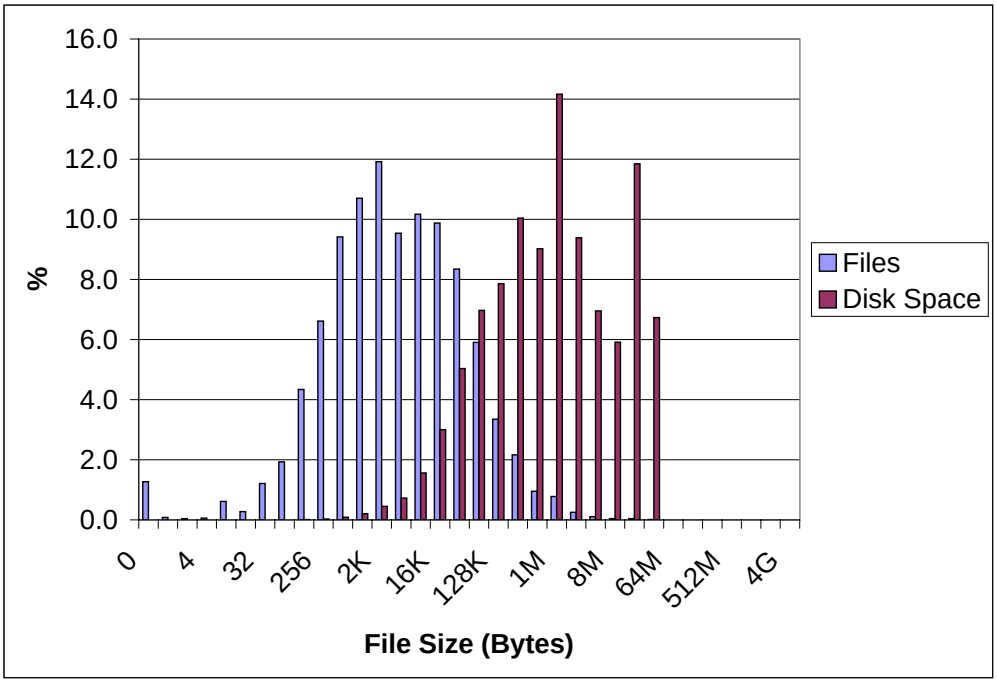

Figure 7.2: File System vs. Disk Space Distribution on a PC running Windows 95. Systems used before 1997.

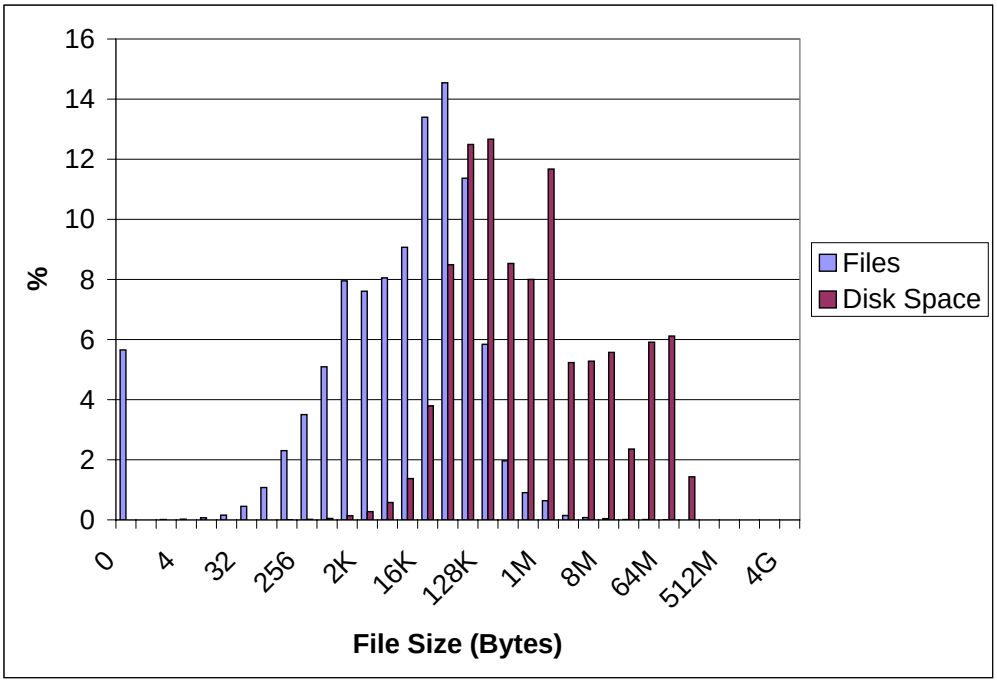

Figure 7.3: File System vs. Disk Space Distribution on a PC running Windows 98. Systems in current use.

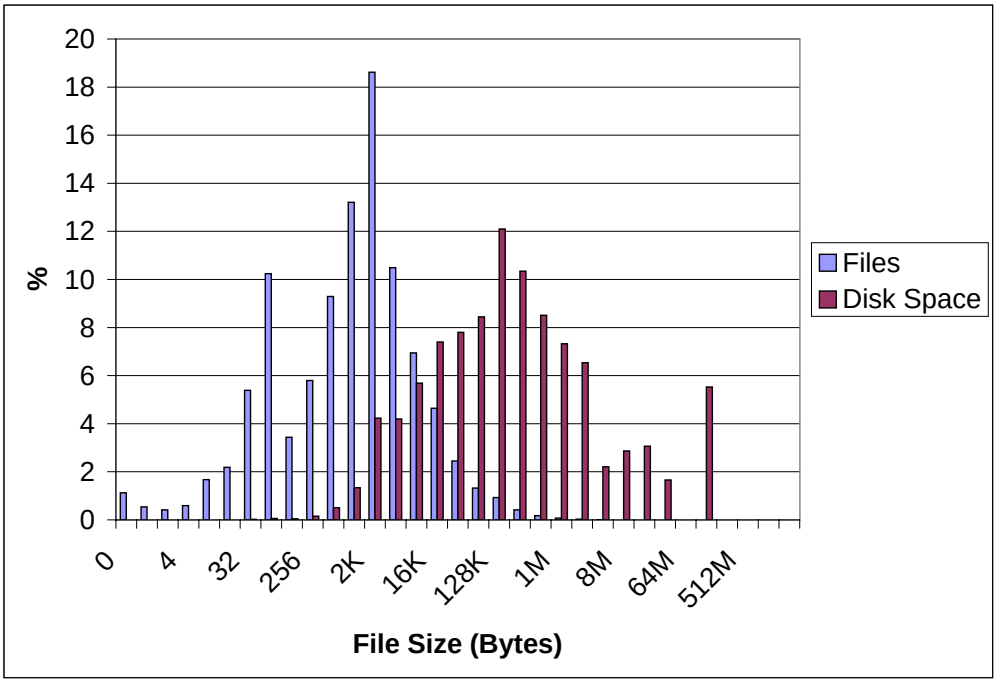

Figure 7.4: File System vs. Disk Space Distribution on a PC running Red Hat 7.2. Systems in current use, however still without user files.

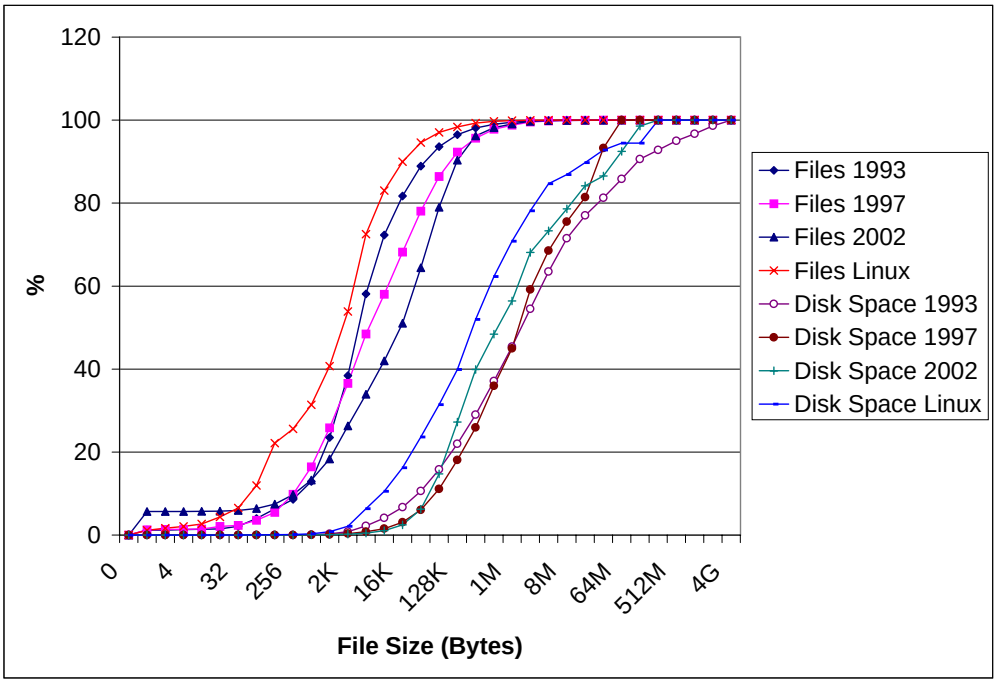

Figure 7.5: File Size vs. Disk Space Cumulative Distribution.

most numerous group are files 16 – 64K and most disk space is taken by files 32 – 128K. These two groups of files overlap, which was not the case ten years ago. It is also visible in Figure 7.5 where the gap between the lines responsible for cumulative file size and disk space distributions for the same year is becoming smaller.

7.2 I-node Space

For computation of the average size of an i-node per file in presently used disk-based file systems and in HeRMES, 1K and 4K block sizes were used. The results obtained for currently used systems are as follows:

- For 1K block size the average size of an i-node per file for FreeBSD is 1323 bytes; for the HeRMES it would go down to 379 bytes, which constitutes for 71% size decrease.
- For 4K block size the obtained results were as follows: 1396 bytes for FreeBSD and 290 bytes for HeRMES, and a 79% decrease.

For both analyzed block sizes it is clear that by moving metadata to MRAM and redesigning the inode itself, the space required by inodes will decrease considerably. There are space savings on two levels, the individual inode size decreases by up to 79% and total space occupied by inodes goes down to 375 Mbytes,

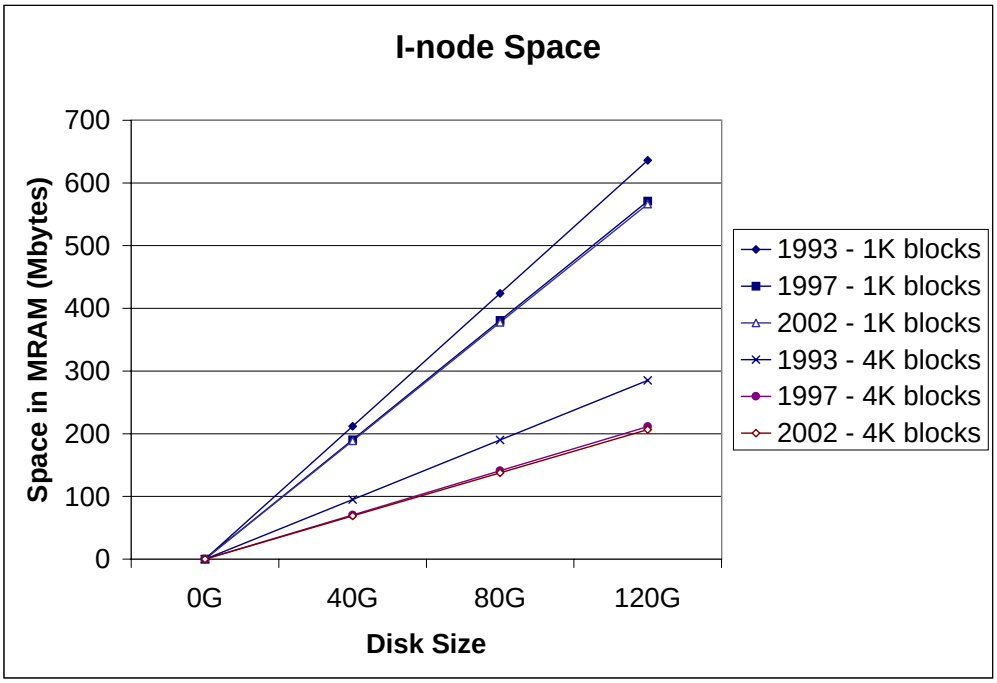

Figure 7.6: Space in memory required to store the i-nodes for a given disk size.

compared to about 4 Gbytes for an 80 Gbyte disk. For 4K blocks the savings are even more impressive. The space required to store all the inodes would go down to 146 Mbytes for the same disk. From Figure 7.6 it can also be seen that the current necessary memory space for inodes is smaller than it would have been in 1993. The reason for this difference is the change in the file size distribution and the corresponding disk usage.

7.3 Performance Benefit

The file system performance analysis was calculated taking into account a most optimistic scenario, that all data to be read is cached in memory. This would account for a worst case speedup for a file system using MRAM as a metadata store. It is important to note that this analysis is still fairly preliminary. It still uses the assumptions regarding metadata structures.

The read/write times used for this analysis were as follows:

- Average disk access time: 5 ms
- DRAM read/write times: 10 ns
- MRAM read time: 1 μs, write time: 50 ns (worst case)

Using these values the average access times for a disk-based file system and for HeRMES were as follows:

$$average\ access\ time\ for\ disk\text{-}based\ FS\ =$$

$$0.57 \times 5 \times 10^{-3}\ + 0.43 \times 10 \times 10^{-9}\ =\ 2.85 \times 10^{-3}\ s$$

$$average\ access\ time\ for\ HeRMES\ =$$

$$0.17 \times 5 \times 10^{-3}\ + 0.43 \times 10^{-6}\ + 0.4 \times 50 \times 10^{-9}\ = 0.85 \times 10^{-3}\ s$$

and the expected speedup came up to be a factor of 3.35 or in other words HerMES would outperform a traditional disk-based file system by 70.1%.

As mentioned above, these results are preliminary. Since storing metadata in MRAM would remove the restricions of having to maintain it in disk blocks, simpler data structures, such as binary trees or hashing could be used. Those data structures would be able to take advantage of newer and faster searching techniques. In general terms using binary trees or other more advanced data structures instead of indexing could guarantee faster search capabilities, which would further increase the performance benefit of using MRAM to store metadata.

While this paper has focused on using MRAM as a storage mechanism for metadata, it could also be used as an additional level of cache for data blocks. This should decrease the number of data disk accessed and further increase performance.

8 Conclusions

Presented work is the analysis of the benefits of HeRMES, an MRAM-based file system. It takes advantage of the fact that MRAM is nonvolatile and uses it to store all of the file system's metadata. This results in at least 70.1% improvement in performance and a reduced metadata size. The above analysis for the HeRMES project deals with the inode space, which is the part that changes dynamically reacting to the changes in the file system itself and therefore is the most interesting. Obtained results show that the space taken up by i-nodes can be greatly reduced if it is not constrained by the block size of the disk. However the relative savings for larger block sizes are not as big, they go up from 71% for 1K blocks to just 79% for 4K blocks. The total space occupied by i-nodes changes more definitely as the block size is increased from 1 to 4K; it shrinks from 375 to 146 MB. However it may turn out that using larger block sizes is less beneficial taking into account the small average i-node size reduction and greater wasted disk space.

9 Future Work

The results of this analysis look very promising. However, taking into account the cost of memory per byte it may be worthwhile to investigate the possibilities of further shrinking the metadata, by taking into account clustering on disk or

maybe applying data compression techniques. Also implementation of this file system and testing it on available file traces is necessary for full assessment of the benefits of HeRMES.

Appendix A

Perl Scripts

The following two scripts were used to scan file systems on UNIX-based and Windows file systems. The purpose of that scan was to count files falling in a specified size range and accumulate their sizes.

Perl Script used to scan UNIX-based file systems:

```
#! /usr/local/bin/perl

# size of the first bin and the bin for oversize files
$j = 1;

#initialize all bins:
for($i = 0; $i < 34; $i ++)
{
    @bin[$i] = 0;
    @binmax[$i] = $j;
    @dsize[$i] = 0;
    $j = $j*2;
}


#scan file system and save it as handle FOO
open(FOO, "ls -lR|");
```

```
#process the file data extracting sizes and classify them
while(<FOO>)
{
    split;
    @foo = @_;

    if($#foo >= 5)
    {
        $size = @foo[4];

        for($i = 0; $size > @binmax[$i]; $i++)
        {
            if($i >= 33)
            {
                $oversize++;
                $doversize = $doversize + $size;
                break;
            }
        }

        if($i < 33)
        {
            @bin[$i]++;
            @dsize[$i] = @dsize[$i] + $size;
        }
    }
}

close(FOO);

#print out the bins
for($i=0; $i<33; $i++)
{
    print "@binmax[$i]\ t @bin[$i]\ t @dsize[$i]\ n";
}
```

Perl Script used to scan machines running Windows:

```
#script to calculate the file size distribution

#open a file for writing
$out="c:/WINDOWS/Desktop/out.txt";

open OUT, ">$out" or die "Cannot open $out for write :$!";

$check = 0;

#get the bins ready
$j = 1;

#set up bins for file sizes:
#count holds number of files less
#than size stored in bin[i]
#size holds the cumulative space taken
#by files from a specific bin
#bin holds file size for comparisons

for($i=0; $i<34; $i++)
{
  @count[$i] = 0;
  @size[$i] = 0;
  @bin[$i] = ($j);
  $j = $j * 2;
}

#store directory listing in list array

@list = `dir /s C:\ \`;

#number of elements in the list array
$last = $#list + 1;

#now go through this list line by line to extract needed
#information
for($i=0; $i<$last; $i++)
{
  #exclude blank lines
  if($list[$i] ne "\ n")
```

```
  {
    #split each line into separate fields
    @line = split/\  s+/, $list[$i];

    #consider only lines that may represent files
    if($#line == 5)
    {
        #store the field that may be the size of the file
        $size = $line[2];

          #look only at fields that start with a digit
          if($size =~ /^\ d/)
          {
            #exclude anything that looks like a date
            if($size !~ /^[0-9][0-9]/)
            {
                @parts = split(/\ /, $size);
                $size = join("", @parts);

                #update bins
                $num =0;

                while($size > $bin[$num])
                {
                    $num++;
                }
                $count[$num]++;
                $size[$num] = $size[$num] + $size;
            }
          }

    }

  }
}

#print out the contents of the bins

for($i=0; $i<33; $i++)
{
  print OUT "$bin[$i]\ t $count[$i]\ t $size[$i]\ n";
}
```

Bibliography

[1] *FreeBSD Documentation*, November 2001.
URL: http://www.freebsd.org/docs.html.

[2] B. Burt. Storage Technologies and Issues for Military Suppliers. *Proceedings of the 19th Digital Avionics Systems Conference, IEEE 2000*, 1:4.A.2 – 1, October 2000.

[3] J. Gait. Phoenix: a safe-in-memory file system. *Communications of the ACM*, 33(1):81–87, January 1990.

[4] G. R. Ganger, M. K. McKusick, C. A. N. Soules, and Y. N. Patt. Soft Updates: A Soulution to the Metadata Update Problem in File Systems. *ACM Transactions in Computer Systems*, 18(2):127–153, May 2000.

[5] G. Irlam. Unix File Size Survey.
URL: http://www.base.com/gordoni/ufs93.html, September 1994.

[6] S. Iyer and P. Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. *Proceedings of the 18th ACM Symposium on Operating Systems Principles SOSP'01*, October 2001.

[7] D. Long, S. A. Brandt, and E. L. Miller. HeRMES: High-performance Reliable MRAM-Enabled Storage. *Proceedings of the Eigth Workshop on Hot Topics in Operating Systems*, May 2001.

[8] M. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System.* Addison Wesley Publishing Company, 1996.

[9] M. McKusick, W. Joy, S. Leffler, and R. Fabry. A Fast File System for Unix. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[10] Peter K. Naji, Mark Durlam, Saied Tehrani, John Calder, and Mark F. DeHerrera. A 256kb 3.0v 1T1MTJ Nonvolatile Magnetoresistive RAM. *IEEE International Solid-State Circuits Conference*, pages 122–123, 2001.

[11] M. Oishi and T. Matsumoto. US Makers Ready MRAM for Post-DRAM Era.

URL: http//www.nikkeibp.asiabiztech.com/nea/200105/cmpo_129399.html.

[12] C. Ruemmler and J. Wilkes. UNIX disk access patterns. *Proceedings of the Winter 1993 USENIX Conference (San Diego, CA)*, pages 405–420, January 1993.

[13] A. A. Wang, G. H. Kuenning, P. Reiher, and G. J. Popek. Conquest: Better Performance Through a Disk/Persistent-RAM Hybrid File System. *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST), Monterey*, January 2002.

[14] M. Wu and W. Zwaenepoel. eNVy: a non-volatile, main memory storage system. *SIGPLAN Notices*, 29(11):86–97, November 1994.