Remote Disk Caching

Joshua Job and Ziliang Zong

Department of Mathematics and Computer Science
South Dakota School of Mines, Rapid City, SD 57701
Joshua.Job@Mines.sdsmt.edu *Ziliang.Zong@sdsmt.edu*

Abstract

A current trend in storage systems is centralizing the location of disks to make the system easier to maintain. Two popular systems for this are ATA storage over Ethernet and ISCSI. Both of these systems take advantage of existing IP over Ethernet and other IP networks to transport information to and from the disks. This creates a storage area network (SAN). The cache on the remote system using a SAN remains in RAM. While fast, the capacity of a RAM cache remains a small fraction of the remote disk. A local disk, however, offers a significantly larger capacity for caching. This paper will describe a method of caching the file system of the remote disk on a local disk. Using this method will improve performance by increasing the potential size of the local cache, by using a disk instead of RAM. While the local disk is not as fast as RAM, it is faster than a remote disk. This is accomplished by maintaining files cached locally in the context of the file system of the remote disk. This paper will then present data from using this method on a simulated system.

1. Introduction

A current trend in storage systems is centralizing the location of disks to make the system easier to maintain. Two popular systems for this are ATA over Ethernet and ISCSI. Both of these systems take advantage of existing IP over Ethernet and other IP networks to transport information to and from the disks. This creates a storage area network (SAN). The cache on the remote system using a SAN remains in RAM. While fast, the capacity of a RAM cache remains a small fraction of the remote disk. A local disk, however, offers a significantly larger capacity for caching.

This paper will describe a method of caching the file system of the remote disk on a local disk. Using this method will improve performance by increasing the potential size of the local cache, by using a disk instead of RAM. While the local disk is not as fast as RAM, it is faster than a remote disk. This is accomplished by maintaining files cached locally in the context of the file system of the remote disk. This paper will then present data from using this method on a simulated system.

This method of caching does not only have the potential to improve disk performance where SANs are used. Systems that centralize the file system can also benefit from this method of caching. Systems that take advantage of these two technologies include web, and FTP servers as well as scientific cluster systems. These servers could then take advantage of a central storage system with less of a performance hit.

2. Related Work

In 2002, D. Colarelli and D. Grunwald presented a similar framework as compared to our caching disks. Their architecture was called "Massive Arrays of Idle Disks" or MAID [1]. MAID has increased storage density, and decreased cost, electrical power, and cooling requirements. However, three important problems remain unsolved in MAID. First, they did not clearly mention about the mapping structure of active drives and passive drives, i.e. which buffer disk should be chosen as the candidate to cache the data whenever there is a data missing operation. Second, there is no prefetching strategy in MAID, which will significantly degrade the performance. Third, they did not consider the load balancing issue, which could easily lead to performance penalties.

Another framework similar to MAID, called Popular Data Concentration (PDC), was proposed by E. Pinheiro and R. Bianchini in 2004 [2]. The basic idea of PDC is to migrate data across disks according to frequency of access, or popularity. The goal is to lay file out in such a way that popular and unpopular files are stored on different disks. This layout leaves the disks that store unpopular files mostly idle. However, PDC is a static offline algorithm. In some cases, it is impossible for the system to exactly know which file is popular and which is not. This is especially true for the ever-changing workload, in which some files are popular at a particular period but becomes unpopular the next period.

3. Implementation

There are three main parts to the caching system: 1) the cache director, 2) the cache maintainer and 3) the external interface. Separation into these three parts allows the user to replace a given section to experiment with different parameters and types of caching. The system is designed as a library that could be attached to an existing application.

The cache director decides which files are to be cached based on the file requested by the external interface. Two directors were tested: 1) an on demand director, and 2) a null director. The on demand director caches the file requested by the user. The null director caches nothing, everything is taken from the disk. Future directors can be made to take advantage of different caching styles, including Data-mining methods.

The cache maintainer tracks what files are in cache and moves files into cache based on advice from the cache director. Files are selected for removal using access time to select the least recently used file. A Table is used to track the location of a given file. When directed to place a file in cache, the table is updated to show that a given is being copied, and a thread is created to transfer the file. When the thread completes the transfer, it updates the table again to show that the file is in cache. If at any time the thread runs out of space in the cache, It selects the file to be removed from the table that is not being transferred. If it cannot remove a file, it calls the scheduler, then tries again. If after three attempts, file still cannot be removed, it removes the file that it started to create, updates the table and exits.

The external interface is made to connect to an application that would use the caching system. For the purposes of these tests, it is connected to a test application that requests files from a list provided by the user. The user is permitted to specify several files be accessed at the same time. The application creates a thread to request and access each file. When the application requests a file, the director is called to start transfer of file to be cached. Then, a file descriptor is opened based on what the table shows as the current location of the file. With several threads accessing and updating the file table at the same time, the table needs to be protected. To simplify the needed protection, 2 rules were put in place: 1) files can 2 not be created or destroyed and 2) file names may not change. These two rules ensure that the table will not grow, shrink or be rearranged, so a lock on the whole table is not needed. Instead, accesses to a given file are synchronized. This reduces the chance that 2 or more threads will interfere with each other.

4. Simulation Results

The test application includes instrumentation to monitor the transfer time for a file and whether the file was a cache miss. A cache miss happens when a given file is not in cache before the monitor is called. Aside from cache misses, a file can also be in the process of being transferred into cache making it a partial hit or miss. If the file is a partial hit, then the application waits for the transfer to complete before timing its own transfer. A partial miss can only result when the cache director orders a file be brought into cache as a result of requesting the file. To ensure that a cache hit is properly logged, the time used to move the file into cache is logged for a partial miss.

The system was tested on an AMD Athlon 64 2x machine with 2 GB of RAM running Debian Linux. The remote disk was simulated by running vblade in a Debian Linux VMWare image. Vblade provides an implementation of ATA over Ethernet. The VMware image was configured with two 4GB virtual disks that reside on 2 separate physical disks. Linux Volume Management (LVM) was then used to stripe data across the disks. The result was then provided

to Vblade for the remote disk. The cache used is a file provided to a Linux Loopback device. The system described here was designed to be a scaled down version of the systems at EROS.

For the results below, a large portion of memory(approx. 1GB) was allocated to a separate program to decrease the amount of memory used as a disk cache; however, the effects of RAM caching cannot be completely removed for small file sizes(approx. 100MB). Future implementations will need to take advantage of these caches. With the effects of these external caches minimized, the system performs well. The caching system out performs the system without caching. The system above gave a 1.6x speedup for 50MB files and a 1.4 speedup for 100MB files.

Director	File Size (MB)	Cache Hits	Average Transfer Time (s)
On Demand	50	23	8.5
Null	50	0	13.5
On Demand	100	10	19.93
Null	100	0	27.25

5. Conclusions and Future Work

This paper examined a method of disk caching. Individual files from a remote file system were cached on a local disk. The implementation was made to be attached to an individual application. An application based implementation was shown to improve file access speed.

This work could be improved by applying dynamic batch pre-fetching strategy. The basic idea of batch pre-fetching strategy is that the controller will not only copy the missed data block to the cashing disk, it will also copy adjacent blocks to the missed block to the cashing disks. This idea is supported by the fact that only a small portion of data will be accessed in most commercial applications. Meanwhile, data access patterns indicate that if one data block is accessed, very likely the data blocks, which are related to this block, will be accessed immediately. In the hardware level, usually the related data blocks are stored in continuous cylinders. Therefore, the batch pre-fetching strategy should be able to reduce the average response time.

References

[1] D. Colarelli and D. Grunwald. Massive Arrays of Idle Disks for Storage Archives. *Proc. of the 15th High Performance Networking and Computing Conf.*, November 2002.

[2] E. Pinheiro and R. Bianchini, “Energy Conservation Techniques for Disk Array-Based Servers”, *Proc. of the 18th International Conference on Supercomputing (ICS’04)*, June 2004.