A DATA MINING ALGORITHM FOR MULTI LEVEL PREFETCHING IN STORAGE SYSTEMS

Mais Nijim, *Yousef Nijim, Vamshi Reddy, Raghu Nandiraju
Texas A&M University Kingsville, USA
Department of Electrical Engineering and Computer Science
mais.nijim@tamuk.edu
Cox Communications, USA
yousef.nijim@cox.com

ABSTRACT

Parallel storage systems have been highly scalable and widely used in support of data-intensive applications. In future systems with the nature of massive data processing and storing, hybrid storage systems opt for a solution to fulfill a variety of demands such as large storage capacity, high I/O performance and low cost. Hybrid storage systems (HSS) contain both high-end storage components (e.g. solid-state disks and hard disk drives) to guarantee performance, and low-end storage components (e.g. tapes) to reduce cost. In HSS, transferring data back and forth among solid-state disks (SSDs), hard disk drives (HDDs), and tapes plays a critical role in achieving high I/O performance. Prefetching is a promising solution to reduce the latency of data transferring in HSS. However, prefetching in the context of HSS is technically challenging due to an interesting dilemma: aggressive prefetching is required to efficiently reduce I/O latency, whereas overaggressive prefetching may waste I/O bandwidth by transferring useless data from HDDs to SSDs or from tapes to HDDs. To address this problem, we propose a data-mining multi-layer prefetching algorithm that can judiciously prefetch data from tapes to HDDs and from HDDs to SSDs. To evaluate our algorithm, we develop an analytical model and the experimental results reveal that our prefetching algorithm improves the performance in hybrid storage systems.

Keywords: hybrid storage systems, solid state disks

1 INTRODUCTION

A recent study shows that new data is growing annually at the rate of 30% and five exabytes (5x2^60) of new information were generated in 2002 [6]. Major corporations, supercomputing centers and rich media organizations, including Lawrence Livermore National Laboratories, Oak Ridge National Laboratory, NASA Ames, Google, Boeing, and CNN rely on large-scale storage systems to meet demanding requirements of large data capacity with high performance and reliability [7]. Large-scale storage systems have to be developed to fulfill rapidly increasing demands on both large storage capacity and high I/O performance [8][9]. Traditionally, the storage capacity and I/O performance of a system are scaled up by simply employing more HDDs [10][11]. However, I/O performance and capacity improvements through the increased number of HDDs are an expensive solution due to huge expenses on new storage equipments and the increased maintenance fee. Hybrid storage systems (HSS) – containing SSDs, HDDs, and tapes subsystems – can provide an ideal data storage solution to significantly improve storage capacity and I/O performance at low cost. Authors who are not fluent in English are requested to ask a native English speaking colleague to correct their manuscript before submission.

Large-scale hybrid storage systems will become increasingly popular in the next few years for the following two reasons. First, HSS will keep its high performance by prefetching and caching highly accessed storage objects in high-speed storage components such as SSD or HDD. Second, hybrid storage systems are cost-effective, because inexpensive tapes help in increasing storage capacities at very low cost. Therefore, it is believed that hybrid storage systems, which have high performance, long archive life, and low cost, are ideal data storage platforms for a wide variety of data-intensive applications from human genome analysis [1] to remote-sensing applications [3]; from long running scientific simulations [4] to biological sequence analysis [2]. Hybrid storage systems are practically feasible because SSDs, as one of the newly developed storage components, can be easily connected to any other types of storage devices [5].

Highly efficient data transfer between tapes, HDDs and SSDs is important to hybrid storage systems. For a wide range of data-intensive applications, critical data are required to periodically
or continuously backed up to tapes so that data restoration is possible in case of system crash or data loss. It is imperative to minimize data restore time in order to improve I/O performance, which largely depends on high data transfer rate between disks and tapes. Thus, transferring data back and forth among SSDs, HDDs, and tapes plays a critical role in achieving high I/O performance.

Prefetching is a promising solution to reduce the latency of data transferring among SSDs, HDDs, and tapes. Prefetching is a process that aims at reducing the number of requests issued to HDDs or tapes by caching popular data in SSDs. Prefetching can be used to prevent I/O bandwidth underutilization by fully exploiting idle times of storage components to hide I/O latency. Prefetching in the context of hybrid storage systems is technically challenging due to an interesting dilemma: aggressive prefetching is needed to efficiently reduce I/O latency, whereas overaggressive prefetching may waste I/O bandwidth by transferring useless data from HDDs to SSDs or from tapes to HDDs. To overcome these technical obstacles, we investigate a multilayer prefetching algorithm (PreHySys) to enhance I/O performance of hybrid storage systems.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the architecture of the hybrid storage system. In section 4, we present the multi-layer prefetching module. Section 5 describes the analytical model of the multi-layer prefetching. The conclusion of the paper is discussed in section 6.

2 RELATED WORK

Integrating Hard Disks with Tapes: Tape-based storage systems are essential archival storage components required by high-performance computing communities. In the past two two decades, magnetic tapes have been considered the most cost effective and reliable way to archive and backup data for a long period of time. Although tape storage systems offer high capacity at low cost, tapes have a performance drawback – high I/O latencies and lower bandwidth due to sequential data accesses. Conventionally, the performance of tape-based storage systems can be improved by integrating hard disks with tapes [26]. Hard disks (e.g., RAID – redundant array of independent disks) can be employed in tape storage systems to store frequently used backup data. To reduce data transfer time between disks and tapes, data striping ideas are applied to increase I/O throughput and reduce response time [12]. increase I/O throughput and reduce response time [12].

Solid-State Disks: Solid State Disks are made of semiconductor memory devices. Currently, there are two types of SSDs: flash memory based SSDs and DRAM based SSDs. Both flash memory based and DRAM based SSDs can be integrated with HDDs and tapes to increase I/O performance of large-scale hybrid storage systems. Recently, flash memory based SSDs are becoming very popular for data-intensive computing because of the advantages inherited from flash memory such as high density and low power properties. There has been some work on applying caches to boosting performance of parallel disk systems. Kotz and Ellis investigated cache management techniques used in parallel file systems to close the gap between processor and disk speeds [15]. Karedla et al. examined the use of caching to reduce system response times and to improve data throughput [16]. It is believed that if the caches are separately treated, it is easy to control cache miss sequences. Each partition for a disk will be managed separately using the conventional LRU (Least Recently Used) replacement algorithm. We developed a cache partitioning mechanism for cluster storage system for achieving high security for data-intensive applications running on clusters.

More to the point, a number of cache partitioning methods have been proposed with different optimization objectives include performance [17], fairness [18], and quality of service [19]. On the other hand, SSDs can provide more advantages than the ones provided by the built-in cache because the built-in storage cache can only be used with that specific storage system while SSDs can be connected to any storage devices [20].

Hardware-Based Prefetching: Potential factors degrading I/O performance of storage systems include heavy load, low bandwidth, bandwidth underutilization, long seek time, and disk rotational delay. A promising approach to boosting I/O performance is to increase I/O bandwidth by prefetching data sets at various places, including storage servers, clients, and proxies. Prefetching techniques found in the literature take either software-based or hardware-based approaches [21]. Software-based prefetching schemes depend on software to detect regular data access patterns, whereas hardware-based approaches rely on hardware to reduce data access penalty [22][23]. Hardware-based prefetching approaches can be classified into two groups: Spatial schemes where data access to the current block is the basis for making prefetching decisions, and temporal schemes where blocks to be prefetched is based on values of speculated data rather than data locality.

Software-Based Prefetching: Existing software-based prefetching solutions can be generally categorized into two camps: informed prefetching [24] and predictive prefetching [30]. Informed prefetching algorithms rely on user-disclosed information about future requests to bring data into buffers. Predictive prefetching techniques utilize data access history to predict future data requests. There exist two types of predictive
prefetching approaches: dependency-graph-based predictive prefetching (DGP) and partial matching-based predictive prefetching (PMP). DGP algorithms use dependency graphs to model access patterns, whereas PMP algorithms maintain Markov predictors to calculate probabilities of future requests. These existing prefetching solutions have several defects: First, informed prefetching is inapplicable in scenarios that data-intensive applications are unable to provide a priori information about which data blocks are likely to be accessed. Second, DGP algorithms predict forthcoming requests using previously accessed data with only considering first order dependencies. Third, PMP algorithms do not address the issue of how to choose a constant value for maximum order. Last, none of these existing prefetching algorithms considers bringing popular data sets from hard disks into solid-state disks.

3 THE HYBRID ARCHITECTURE

Figure 1 reveals the hybrid storage system architecture with prefetching. It consists of FTP servers with SSDs, storage area network (SAN) with HDDs, and a tape subsystem. The solid-state disks are designed to keep the most highly accessed files. Files with less priority or less access frequency are stored in HDDs of the SAN subsystem and all other files are stored in the tapes.

The SSDs and HDDs are considered as high-end storage components while the tapes are classified as low-end storage components.

A prefetching scheme is designed to bring data into HDDs of SAN and optimally, into the SSDs before it is requested. Our multi-layer prefetching algorithm consists of two parts, the upper level prefetching and the lower level prefetching. The upper level prefetching transfer the data from the HDDS to the SSDs and the lower level prefetching transport the data from the tapes to the HDDs. A miss command will be issued when requested files cannot be found in SSDs and the missing files will be fetched to the SSDs from the HDDs. If the requested files are not located in the HDDs, the algorithm will issue next-level miss command and place the missing files from the tapes to the HDDs and finally to the SSDs.

4 PREFETCHING ALGORITHMS

4.1 Parallel Data Transfer from Tapes to Disks

In this section, we propose an approach to quickly move data from parallel tape storage system to hard disks to achieve high I/O performance.
Parallel tapes libraries can intuitively increase the aggregate bandwidth between the disk storage and the tape storage and reduce the tape switch time by introducing parallel load/unload operation. This prefetching mechanism needs to schedule read requests in a way that the highest priority data blocks will be fetched first. To support the data transfer parallelism, a striping and data placement techniques for the hybrid storage system are used.

Striping is a well-known technique for improving the effective I/O bandwidth for storage systems. We consider data striping to access tape-resident data sets in parallel. Data sets will be divided into uniform chunks to be prefetched and stored to disks simultaneously. The data striping scheme is completely transparent to the PreHySys prefetching mechanism. To determine an optimal striping width, we consider both data size and I/O workload.

The striping technique can support data transfer parallelisms, thereby shortening I/O response times. However, striping causes a large number of small I/O calls, which in turn increase switch time of tape drives [27]. Considering the big penalty associated with tape switches and the long transfer time associated with the huge object size, we propose a data placement scheme to leverage object access probabilities and the relationship between objects. This scheme can improve tape switch parallelism and synchronize data seek with high probability, and thus increase the parallelism of the object transfer. The main goal of our data placement algorithm is to reduce the tape switches within the tape library, increase tape switch parallelism among tape libraries, and increase the data transfer parallelism among tape drives. To achieve these goals, we propose a data clustering mechanism that clusters objects with high probability to be requested together.

Our data clustering idea is based on an assumption that related data requests are highly to be requested together. Data blocks with a high probability to be requested together will be grouped in one cluster. The tape subsystem will be composed of \( n \) tape libraries. Each tape library has \( d \) identical tape drives, \( s \) switch drives, and \( t \) tapes. The basic idea is to divide each tape library into two groups. The first group contains tapes that are mounted all the time and contains high emergency data. The number of tapes that are mounted all the time is \( d-s \). The second and later clusters contain \( s \) tapes for each tape library, which is mounted during startup time and will swapped out if the requested stripe cannot be found within the mounted tapes. Fig.2 is an illustrated example of object clustering mechanism in the tape storage. Assume there are three tape libraries where each tape library contains seven tapes, five tape drives, and two switch drives. The first batch for each library contains three tapes, and the second and later batches contain two tapes each.

We developed a prefetching algorithm called PRE_TD for tape resident data. When the requested data sets are not located in the SAN subsystem, PRE_TD will fetch the data set from the tapes to the HDDs in SAN. PRE_TD does not prefetch data blocks if a higher priority data block must be evicted from the disks. We assume that the user’s access pattern is known in advance. Each I/O request in the access patterns is assigned a priority level depending on data access history. It is worth noting that the priorities of striping units are equal to the priority of the corresponding I/O requests. Moreover, the priority of a data block in a disk is derived from the priority of the next reference to the data block. If there is no reference to the data block in the access pattern, the block’s priority will be \( i-Max \), where \( i \) is the index of the most recent reference in the data block, and \( Max \) is a large positive integer that is greater than the length of the access pattern. If data blocks residing in the disk subsystem are not appeared in access reference lists, then the blocks will be assigned the lowest priority. Data blocks will be migrated to tapes based on the least recent used policy. When PRE_TD is invoked, the striping units of an I/O request will be mapped to the disk subsystem using the round-robin mapping strategy. PRE_TD will construct a list of prefetched requests in accordance with their priorities. PRE_TD will have all the I/O requests in the list processed by disk drives after the requests are assigned to the disks. The PRE_TD algorithm is described in fig.3a.

- **Figure 2:** The data structure of the PRE_TD Algorithm. Fig. 2(a) is an access pattern known in advance. Fig. 2(b) shows the striping units of I/O requests distributed among tape libraries. Fig. 2(c) shows a prefetching schedule and updated priorities.
4.2 Parallel Data Transfer from hard disk drives to solid state disks

The performance of large-scale storage systems will be substantially improved by employing a number of solid-state disks that can help in reducing the number of hard disk accesses. We develop a parallel data transfer algorithm called PRE-DS that prefetches data set from the disk system to the solid state disks. The first component in PRE-DS is a SSD partitioning algorithm (or PaSSD for short), which dynamically partitions an array of SSDs among HDDs in such a way to maximize I/O performance. The basic idea of PaSSD is motivated by the observation that I/O workload is not uniformly distributed among parallel hard disks. The solid-state disks will be allocated dynamically depending on the popularity, size of content, and access patterns. Taking these factors into consideration, two approaches are implemented in PaSSD to assign weights for contents and associate a solid-state drive to each content. The first approach is called Content-Popularity-Based Weight Assignment, in which the weight of a new content is assigned based on its popularity of other relevant contents that already have measured popularities. If a large number of users have requesting content regularly, the content should be fetched and cached on a solid-state drive based on the measured weight. Weights will be adjusted dynamically and regularly to reflect any changes on the fly. The second approach is called Collaborative-Popularity-Based Weight Assignment. In this approach, we specify popularity weights by considering correlations among requested contents. For example, if user u1 requested content c1 and c2, and user u2 requested content c2, then c2 will receive a higher weight because there is a strong likelihood that both u1 and u2 will request c2. PaSSD will keep assessing contents and estimating the weights on a regular basis in a dynamic environment. After popularity weights are assigned using the above two approaches, PRE-DS will decide to forward requests to either SSDs or HDDs. Furthermore, PRE-DS will be used to transfer data from HDDs to SSDs if requested popular data blocks are not available in the solid-state drives, and if transferring that data block does not cause a data block with higher priority to be evicted from the solid-state disks. To improve I/O performance, PRE-DS strive to fetch as many popular data blocks as possible from all hard disks and store these popular data in the high-speed solid-state disks. Fig.3b describes the PRE-DS algorithm.

5 MATHEMATICAL MODEL FOR HYBRID STORAGE SYSTEMS

In this section, we describe the analytical model to access the proposed PreHySys algorithm. The mathematical model consists of three parts: the server access model, the access time without prefetching, and the access time with prefetching. The ultimate goal of this model is to provide criteria that can mathematically evaluate the performance of our prefetching algorithm. More specifically, we want to maximize the average access time improvement S, where S is defined as follow:

\[ S = t - \bar{t} \]  

(1)

Where \( t \) and \( \bar{t} \) are the average access time when prefetching is not carried out and when prefetching is carried out respectively.

In the server access model, we consider multiple users accessing the network through the ftp server. In our analysis, we consider M/G/1 round robin queuing system. In this system, the average time to finish a job, necessitate a service time \( x \), is calculated as follows:

\[ r = x/(1 - \rho) \]  

(2)

Where \( \rho \) is the system utilization.

Let the average size of an object be \( s \), let \( \bar{s} \) be the size of an object located in the disk system, and let \( s' \) be the size of an object located in the tapes storage where \( s = \bar{s} + s' \). This means that part of the object could be located in the disk system, and the other part is located in the tape storage. In this case, the upper level and the lower level prefetching algorithms need to be called. If \( \bar{s} = 0 \) then the data object is located in the tape storage and the lower level prefetching algorithm will be called. Otherwise, if \( s' = 0 \) then the data object is located in the disk system, and the upper level prefetching algorithm will be called.

Let \( \bar{b} \) be the bandwidth in the upper level hybrid storage system (i.e. the bandwidth from the disk system to the solid-state-disks). Let \( b' \) be the bandwidth in the lower level (i.e. the bandwidth from the tape storage to the disk systems). We assume that \( b' > \bar{b} \). The average service time \( x \) is defined as:

\[ x = (\bar{s} + s')/(\bar{b} + b') \]  

(3)

Requests at arrival rate of \( A \) are issued by the users. Based on the assumption that the arrival rate is not affected by prefetching which means that prefetching is totally transparent to the users. Without prefetching, a proportion \( h_i \) of the users requests results in a hit in the solid-state disks. This means that this proportion is served by the solid-state disks. The failure ratio is \( f_i = 1 - h_i \) which means that the
requests are located in the disk systems and/or the tapes. The proportion \( h_d \) results in a hit in the disk system which means that \( h_d \) proportion of the user’s requests are served by the disk system. The complementary value \( f_d = 1 - h_d \) means that requests are served by the tape system.

We consider the case where prefetching is not performed. Using (2) and (3) the average retrieval time of the requests from the disk system is

\[
r'_d = \frac{s'}{b'(1 - \rho'_d)}
\]

(4)

The requests are issued by the users at rate \( \lambda \).

Where

\[
\rho'_d = \frac{h_d \lambda s}{b}
\]

(5)

Thus,

\[
r'_d = \frac{s'}{b - h_d \lambda s}
\]

(6)

The average retrieval time of the requests from the tape storage is

\[
r'_s = \frac{s'}{f_d s' b'}
\]

Where

\[
\rho'_s = \frac{h_d \lambda s'}{b'} + \frac{f_d \lambda s'}{b'}
\]

(7)

Thus,

\[
r'_s = \frac{s' b'}{b' - \lambda (b'h_d \lambda s + b f_d s')}
\]

(8)

we assume that the access time is 0 when the requests are located in the solid-state disks. The access time for the requests that are located in the disk system and in the tape storage are \( r_d \) and \( r_i \) respectively. Hence, the average access time is

\[
t' = h_s \cdot 0 + h_d r_d + f_d r_i
\]

\[
= h_d r_d + f_d r_i
\]

\[
= \frac{h_d \lambda s}{b - h_d \lambda s} + \frac{f_d s' b_d}{b' - \lambda (b'h_d \lambda s + b f_d s')}
\]

(9)

A. Access Time with Prefetching

We calculate the access time for requests considering prefetching. For each user request, an average of \( n(F) \) items will be prefetched where

\[
n(F) = \overline{n}(F) + \overline{n}(F)
\]

(11)

Where \( \overline{n}(F) \) is the average number of items to be prefetched from the tapes to the disks system, and \( \overline{n}(F) \) is the average number of items to be prefetched from the disks to the solid-state disks.

We assume that all items that will be prefetched have probability \( p \) where

\[
p = p_1 + p_2
\]

(12)

Where \( p_1 \) is the probability of items to be prefetched from the tape storage to the disk system, and \( p_2 \) is the probability of items to be prefetched from the disk system to the solid-state disks. For simplicity, we assume that the items to be prefetched from the tapes to the disks have the same probability, and the items to be prefetched from the disks to the solid-state disks have the same probability.

When prefetching is performed on the lower level (i.e. from the tape storage to the disk system), the hit ratio in the disk system will be increased by the number of the prefetched items. Thus, the hit ratio in the disk system is expected to rise to \( \hat{h} \), where

\[
h = h_d + \overline{n}(F) p_2
\]

(13)

When the data objects are to be prefetched from the disk system to the solid-state disks, the hit ratio in the solid state-disks is expected to rise to \( \hat{h}' \) where

\[
h' = h_s + \overline{n}(F) p_1
\]

By using prefetching, not only the values of the hit ratio will be affected, but also the system utilization, the retrieval time, and the access time will be affected. One important factor of prefetching is which items in the disk system or in the solid state disks will be evicted in case they are full. To solve this problem, it is necessary to model the interaction between prefetching and the replacement policy.

The rate of users’ request that require demand fetches for the hybrid storage system is \((1 - h)\lambda + (1 - h')\lambda'\). Now the ftp server must serve both on demand fetches ad well as prefetching. Thus, the effective rate of the requests is

\[
(1 - h)\lambda + (1 - h')\lambda' + \overline{n}(F) + \overline{n}(F)
\]

(14)

Based on (14), the system utilization if prefetching is done on the upper level system (i.e. from the disk system to the solid-state disks) is
\[ \rho_d = \frac{(1-h+\bar{n}(F))\lambda s}{\bar{b}} \]  

Using (2) and (3), the average retrieval time of the data objects from the disk systems to the solid-state disks is

\[ r_d = \frac{\bar{s}}{\bar{b}(1-\rho_d)} \]  

Using (15), we get

\[ r_d = \frac{\bar{s}}{\bar{b} - (1-h' + \bar{n}(F))\lambda s} \]  

(17)

The system utilization when the data objects are prefetched from the tape storage is calculated as follows

\[ \rho_t = \frac{(1-h + \bar{n}(F))\lambda s + (1-h' + \bar{n}(F))\lambda s'}{b'} \]  

(18)

The average retrieval time of data objects to be retrieved from the tape storage is

\[ r_t = \frac{s' \bar{b}}{b'(1-\rho_t)} \]  

Using (18) we get

\[ r_t = \frac{s' \bar{b}}{bb' - (b'(1-h+\bar{n}(F)))\lambda s + \bar{b}(1-h'+\bar{n}(F))\lambda s'} \]  

(20)

Thus, the average access time when prefetching is performed:

\[ t = h \cdot 0 + (1-h') \cdot r_d + (1-h-h')r_t \]

\[ = \frac{1-h - \bar{n}(F)p_l}{\bar{b}(1-h_l - (1-p_l))\lambda s_l} + \frac{f_d - \bar{n}(F)p_z - \bar{n}(F)p_z s' \bar{b}}{bb' - b'(1-h_d + \bar{n}(F))(1-p_z)\lambda s + \bar{b}(1-h_z + \bar{n}(F))(1-p_z)\lambda s'} \]  

(21)

Substituting (10) and (21) in (1) which defines the access improvement when using prefetching, we get

\[ S = \left\{ \frac{1-h_l - \bar{n}(F)p_l}{b(1-h_l - (1-p_l))\lambda s} + \right\} \]

\[ \frac{f_d - \bar{n}(F)p_z - \bar{n}(F)p_z s' \bar{b}}{bb' - b'(1-h_d + \bar{n}(F))(1-p_z)\lambda s + \bar{b}(1-h_z + \bar{n}(F))(1-p_z)\lambda s'} \]  

(22)

6 TESTBED

To test the performance of the multi-layer prefetching storage, we will build a testbed. The testbed platform is a personal computer running Linux equipped with Intel Pentium Dual Core 3.2 GHZ processor, 1GB of main memory, the hard drive is Samsung SATA disk, the solid state drive is 32GB Transcend SSD. Depending on the access pattern known in advanced, we measure the performance of the system with and without prefetching. Our primary metric for measuring the system performance is the overall completion time, the average time to read a block, and the average disk access time. However, our ultimate goal is to test the success of our multi-layer prefetching algorithm.

Figure 3 plots the results for five values of the average number of prefetched items against the access improvement where the data size is 5MB; the value of the probability is 0.4, the arrival rate is 30MB/sec. Figure 3 reveals that when the average number of prefetched items increases, the average access time improvement \( S \) (please see equation 1) of the system will increase.

7 SIMULATION AND EXPERIMENTAL RESULTS

To simulate our algorithm, we use Microsoft Access which provides a good foundation for the database technique called data mining. It uses “objects” to help the user list and organize information, as well as prepare specially designed reports. Just as a spreadsheet uses a table with rows and columns, Access has an array of data “containers”
(views, filters, etc.) that elevate one-dimensional lists into the powerful realm of data management. Those "objects" are the database table, form, query, and report.

The table is the "warehouse" of the information you store in your database. The rows and columns of the spreadsheet become your records and fields. A record is one individual item of information that consists of fields. The database user must be able to easily perform quick filters in a form or a table. (A "filter" is a look at just one specified category of information, e.g., all customers who purchased less than a specified amount). Clicking on the “advanced filter” quickly either in the form or table view allows the user to “drill” into the data to find specified information in our project. A filter is really a sort of "query on the fly" for a quick look at limited parts of the database.

The Access Forms are simply a unique way of viewing the information already in your table. Tables with hundreds of records and many fields can be wide and deep. Forms help us display, view, and use all or some of the information in the table. Forms also are a more convenient way to enter new data, and access existing records. Remember: whatever deletions, changes or record additions you make on a form is fed into or removed from the table. Query is another most important part of our work without the trusty query, our database would be as unwieldy as an overstuffed spreadsheet table. We would not be able to see a group of “trees,” because the “forest” of information in both urgent and non-urgent data would spread wide and deep. We need the power of queries to bring order and focus to our inventory of hard-earned data. It is important to remember that a query is nothing more (or less) than a restricted view of the big data table. Any changes made to the big table are also reflected in the query table and objects (forms and reports) that depend on the query. Queries, in other words, are restricted, or truncated, view of the overall database. A saved query in an MS Access database is like the web bookmark. In a saved query, we have instructed Access to reserve a specific view of the information either in a table or a form designed especially for the query. Database managers frequently use queries for special form views so that data entry people can only see, use or edit part of the database. The power of the MS Access database is not always in its robust ability to hold tons of data. It is through its ability to “drill down” which makes Data Mining more efficient.

In our simulation, we use two data sets which are urgent and non urgent. We use the Market Basket Analysis Data Mining Technique which is a modeling technique based upon the theory that if you buy a certain group of items, you are more or less likely to buy another group of items. For example, if you are searching to buy a mobile device then all the related items to mobile will be displayed such as screen guards, pouch, etc. Therefore, serves as an efficient and effective technique to prefetch the data. We make use of the combo box tool provided by MS access and in the back end we write the code to retrieve the data in visual basics.

![Figure 4: The Response time with and without prefetching](image)

![Figure 5: The retrieval time with prefetching](image)

![Figure 6: The retrieval time without prefetching](image)
Figure 4 shows the rate of data transfer rate with and without prefetching. We can notice here that relative prefetch of 5 consecutive data transfers achieves nearly a 2x improvement over the normal data transfer without prefetching with similar growth curves.

Figure 5 and figure 6 depict the data retrieval rate with and without prefetching. In figure 5, the system idle time is equal to its response time where as in figure 6 the idle time decreased and the response time decreases which results in improvements in the data retrieval rate.

Figure 7 plots results for five values of the average number of prefetched items against the access improvement where the data size is 10 MB; the value of the probability is 0.4. Fig. 4(a) and figure 4(b) show the access improvement when the arrival rate is 30MB/sec and 80MB/sec respectively. Figure 4(a) (b) also reveals that when the average number of the prefetched items increases, the access improvement will increase as well.

8 CONCLUSION

The use of large scale parallel disk systems continues to rise as the demands for data-intensive applications with large capacities grow. Traditional storage systems scale up storage capacity by employing more hard disk drives, which tends to be an expensive solution due to ever increasing cost for HDDs. With the new evolutionary technology on storage components like solid-state disks, hybrid storage systems, which contain both high-end storage components (SSDs and HDDs) and low-cost storage components (tape), will be an ideal solution for next generation of data-intensive applications at petascale or extrascale level.

In hybrid storage systems, judiciously transferring data back and forth among SSDs, HDDs, and tapes is critical for I/O performance. We propose a multi-layer prefetching algorithm (PreHySys) that can reduce missing rate of high-end storage components thereby reducing the average response time for data requests in hybrid storage systems. To validate PreHySys, we also build an analytical model that can mathematically evaluate the performance improvement (i.e., the average access time improvement) when prefetching is carried out.

The contributions of this research are two folds. First, to the best of our knowledge, this is the first time multi-layer prefetching techniques are proposed in the context of hybrid storage systems. Second, we present the first mathematical model to evaluate the prefetching algorithms for hybrid storage systems. For future work, we will further improve our PreHySys algorithms and analytical model by applying them to real world hybrid storage systems with data-intensive applications.

ACKNOWLEDGMENT

The work reported in this paper was supported by the US National Science Foundation under Grant CNS 1048432 and Texas A&M University-Kingsville under start up fund.

REFERENCES


