Abstracts: A Latency-Hiding Technique for High-Capacity Mass-Storage Systems

Abstract

Extraordinary advances in digital storage technology are rapidly making possible cost-effective, multiple-terabyte information retrieval systems. The latency and bandwidth of these technologies are typically much worse than what users of computer systems are accustomed to. Unfortunately, traditional techniques of reducing latency and improving bandwidth, caching and compression, by themselves will not work well with the access patterns that we anticipate for these high-capacity systems.

We introduce and define a new storage management technique, called *abstracts*. An abstract is an extraction of the "essential" part of the data set. It is created using some combination of averaging, subsetting, rounding, or some other method of condensing the data. An abstract's composition is heavily dependent on the context in which it is used. Each data set can have multiple abstracts associated with it, each of which can be used to answer a different subset of the possible queries that might be posed about the data. When we are able to answer a query from an abstract, effective bandwidth increases, because we transfer much less data through the storage system. The counter-intuitive result is that abstracts on robot-based tape storage systems can have lower latency than full data sets on magnetic disks, because the inherent latency disadvantage of tertiary systems can be overcome by the reduction in transfer time due to the smaller transfer size. Moreover, because many abstracts can fit in faster storage in the space occupied by a single unabstracted data set, users can get the effect of magnetic disk latencies for very large objects.

To evaluate the potential of abstracts, we examine four common queries as well as a detailed case study. We also study the statistical characteristics of several data sets in an effort to identify classes of abstracting functions.

1 Introduction

I/O device designers are combining robots with magnetic tapes and optical disks to create peripherals containing many terabytes for an order of magnitude less than the cost of the corresponding amount of magnetic disk. Table 1 motivates our work by illustrating the dramatic difference in cost and performance between secondary storage (magnetic disk) and tertiary storage (optical disk and tape libraries). The relative prices given for the tape libraries and jukeboxes may even be too high, because much of the cost of such systems is in the robots, which are likely to become cheaper as the technology matures and becomes more popular. Computer systems designers are thus presented with the challenge of creating inexpensive systems which give the illusion of large amounts of fast storage.

TABLE 1. Typical media costs, capacities and latencies for various storage technologies.

Storage technology	Total Capacity (GB)	Total Cost ($)	Cost/ Capacity ($/GB)	Media Capacity (GB)	Latency (sec)
Magnetic Disk	1 GB	$2500	$2500	1 GB	0.010 sec.
Sony Optical Disk Jukebox	300 GB	$120,000	$400	6 GB	5.5 sec.
Dilog DAT Stacker	10.4 GB	$5,480	$527	1.3 GB	75 sec.
Exabyte 120 Tape Library	500 GB	$40,000	$80	5 GB	100 sec.
Metrum RSS-600 Tape Library	8700 GB	$540,000	$62	14.5 GB	50 sec.

What is the opportunity given this extraordinary increase in cost/capacity? A mundane answer is simply to increase the number of traditional files. Computer systems that may today consists of tens of thousands of small UNIX files would evolve to systems containing tens of millions of small UNIX files, perhaps turning the UNIX command “rm” into a relic. While such a use is inevitable, we believe applications outside of traditional computer science may make more imaginative uses of these new storage systems.

One potential use of huge storage systems is to provide on-line access to images, either copied from the real world or created by computer simulation. As one example, the ability to do this on a massive scale has been identified as a barrier to the development of models to predict the impact of global human activity on the planet Earth, one of the “Grand Challenges” for computing in the 1990s [CPM91]. Specific issues include the greenhouse effect, ozone depletion, scarcity of potable water, deforestation, and the increasing toxicity of the atmosphere. One responsibility of scientists who work on global change is the development of public policy, particularly with respect to costly remedies to control the impact of human enterprise on the global environment. Surprisingly, it is difficult to predict the long-term effects of even well-documented changes, because the understanding of variations caused by nature is so poor.

Researchers in this area have found that serious problems in the data systems available to them impede their ability to access needed data and thereby do research. They require a massive amount of information to be effectively organized in an electronic repository. They also require ad-hoc collections of information to be quickly accessed and transported to their workstations. The hardware and software currently available are totally inadequate to support the needs of this community [STO91].

For example, NASA’s Earth Observing Satellite system will soon be beaming data to ground stations at 1.9 terabytes per day [DOZI91]; over the projected fifteen year lifetime of the project, the total volume of the archive will be over 10 petabytes. As another example, individual simulations of climate conditions can produce more than 70 GB of data per run today. [MECH91] Keeping these data sets on magnetic disk would be prohibitively expensive. Robotic tape libraries and optical disk jukeboxes have the potential for cheaper storage, but the latency of these technologies are typically much worse than traditional magnetic disk. Further, we anticipate that interactive response times will be needed in response to queries about these data sets, rather than response times measured in minutes [STON91]. For instance, scientists would like to view animated sequences of visual images based on the data generated by their climate model.

The problems faced by global change researchers are faced by other users as well. Most of the “Grand Challenge” problems share these characteristics, i.e. they require large amounts of data, accessed in diverse ways from a remote site quickly, with an electronic repository to enhance collaboration. Moreover, these issues are also broadly applicable to the computing community at large. Consider, for example, an automobile insurance application. Such a company wishes to store police reports, diagrams of each accident site, and pictures of damaged autos. Such image data types will cause existing data bases to expand by factors of 1000 or more, and insurance data bases are likely to be measured in terabytes in the near future. The result is that the same latency problems will appear [STON91].

We believe that traditional approaches to managing storage hierarchies are inadequate when applied to massive storage systems. In particular, caching and compression will not reduce latency sufficiently to enable interactive systems using high-capacity tertiary memory devices. This paper presents a new technique, called *abstracts*, which overcomes many of the limitations of traditional storage management approaches. Moreover, abstracts can be combined with compression and caching to yield much faster response times.

An abstract is an extraction of the essential information in a data set; its composition is a function of the usage intended for the data. The key observation is that while queries posed to these data sets will often touch large portions of the data, the answers themselves may be tiny: “Which regions of this satellite image have snow?” Abstracts exploit this observation to improve the performance of storage systems that include high-capacity tertiary devices.

The benefits of abstracts depend strongly on the access patterns which occur to the data. To evaluate the potential of abstracts, this paper examines four common queries and presents a detailed case study of the use of abstracts to answer queries of interest to global change scientists. We also study the statistical characteristics of several data sets in an effort to identify classes of abstracting functions.

In order to further quantify the benefits offered by abstracts, we are developing a system incorporating the ideas we have introduced in this paper. Global change researchers in the Sequoia 2000 project at U. C. Berkeley, U. C. Los Angeles, U. C. Santa Barbara, and U. C. San Diego will be the user community for this project.

The remainder of this paper discusses these issues relating to improving performance in massive information retrieval systems in more detail. Section 2 evaluates traditional storage management approaches when applied to this environment. Section 3 defines abstracts and gives some indication of the performance boost that can be expected from the technique. Section 4 offers examples of some typical user-level operations in this environment, and shows how abstracts can be applied to accelerate these operations. Section 5 discusses statistical issues in abstracting data. Section 6 gives a detailed analysis of an abstracted data set and the performance of a query which we tested against it. Finally, Sections 7 and 8 offer some conclusions and indicate our plans for future work.

2 Traditional Approaches

In this section we argue that caching and compression by themselves will not work well for managing massive storage systems. Combining caching and compression with abstracts, however, can improve this situation.

2.1 Caching

Caching improves performance by storing frequently used data in fast, expensive memory. The performance of a cache depends strongly on both the hit rate and the miss penalty. Both work against the effectiveness of caching for managing tertiary storage.

Caches achieve high hit rates by exploiting localities in the access patterns exhibited by users and programs: a piece of data which has just been read is likely to be read again soon, and its neighbors are likely to be read soon too [SMIT82]. Localities of reference do not help, however, when the working sets are far larger than the cache itself. For instance, while earlier studies have shown that file caching is effective, these studies have concentrated on Unix-based systems comprised mostly of many small files and only a few larger files [OUST85] [BAKE91]. These studies suggest that when files become large enough that only a few files can be stored in memory, caching will be significantly less effective. One of these studies [BAKE91] specifically blames worse than expected cache performance on the increasing use of large files in Unix systems.

We believe that the cache hit rates for many scientific applications will be quite low because of the massive data sets involved and because queries will commonly touch significant portions of the data sets. We give several examples of this later in the paper; rendering images to a screen is one illustration. If a 300 megabyte image (the size of one Thematic Mapper Satellite image) is loaded from tape into fast memory in order to display it on a 1 mega-pixel screen (at 1 byte per pixel), then only 0.3 percent of the information transferred was needed, and 300 bytes were moved (and caused other cached data to be displaced) for every useful byte.

Partial file caching, or caching only the most frequently referenced portions of each file, does not address this problem. One way to view this is that a cache stores an exact version of a piece of the data set; scientific queries often call for a summary or inexact view of the entire data set.

Further, each miss in a cache backed by tertiary storage is exceedingly expensive. As we discussed earlier, an access to tertiary storage – initiated as a result of the miss – requires tens of seconds just to reach the first byte. The cost of fetching a block of data from tape is dominated by the mechanical delays associated with loading and seeking the tape. Fetching small, widely dispersed blocks of a data set from tape will thus be essentially as time-consuming as bringing in the whole data set.

The combination of low hit rates and long miss penalties suggest that accesses will be sent to tertiary storage far too often for interactive response time to be maintained.

2.2 Compression

Compression takes advantage of patterns in data by removing redundancies and, in effect, squeezing the data to maximize the information contained in each byte. Video, images, and satellite sensor data are considered good candidates for compression because of natural regularities in the data.

Compression can be lossless (i.e., the decompressed data is identical to the original) or lossy (some information is lost). Lossy compression generally yields much higher compression ratios than lossless. For some applications, such as image rendering, this is a reasonable trade-off. However, both lossy and lossless compression are typically used to improve capacity, rather than to reduce latency. For instance, Cate and Gross store compressed versions of files on disk, and cache uncompressed data in main memory [CATE91]. This has the effect of increasing the amount of data which can be stored on disk. This is of relatively little benefit, however, with the exceptionally high capacities available in tertiary storage. By contrast, we propose storing the original data on tape and caching the abstracted data on disk and in memory.

Lossless compression typically reduces the size of data objects by a factor of 2-5. [ZIV77] [LELE87]. This allows the storage system to transfer more information with fewer bytes, thus making more effective use of available bandwidth. [BURR91] But compression alone does nothing for latency; the many seconds consumed loading a tape and seeking to a location on that tape are not eliminated simply by shrinking the size of the data to be read from the tape once the first byte is found.

If compression is combined with caching, things look a little better. A cache of a given size can hold more compressed data than uncompressed data. But again, the sheer size of the data sets which we expect to see on a massive storage system limits the effectiveness of a cache of reasonable size. A 70 GB data set compressed to 20 GB will still have difficulty fitting on a 1 GB disk!

Lossy compression gives much better size reductions; the important issue is which parts of the data set it is acceptable to discard. For images that are to be viewed, we can take advantage of the human perception system's ability to compensate for the loss of certain aspects of the image. Colors can be blended or shapes made vague, and people will still be able to discern the contents of the image. This is especially true of moving video, where the eye naturally interpolates between frames. Both the MPEG standard for moving video [LEGA91] and the JPEG standard for still images [WALL91] exploit these elements of visual compressibility. But this doesn't say which information can be lost with data types other than images, nor does it work with applications other than image rendering. The information that is lost in a lossy compression algorithm designed for images may be crucial to a data set of some other type, like satellite sensor data or modelling output.

3 Abstracts

In this section we introduce and define a new technique, called *abstracts*, for improving the latency of massive information retrieval systems which include tertiary storage devices. We explain how abstracts are created and used, and we describe their potential performance benefits. We then discuss how caching and compression can be made far more effective when used in conjunction with abstracts.

3.1 Definition

An abstract is an extraction of the "essential" part of the data set. It is several orders of magnitude smaller than the original data set which it represents. It is created using some combination of averaging, subsetting, rounding, or some other method of "condensing" the data. An abstract's composition is heavily dependent on the context in which it is used. It can be considered a precomputed answer to an anticipated query, or precomputed data that helps answer an entire class of queries. Each data set can have multiple abstracts associated with it. Each abstract is a separate view of the data, and thus can be used to answer a different subset of the possible queries that might be posed about the data.

Because an abstract is so condensed, the result of a query answered by an abstract may be an inexact approximation of the result which would have been obtained with the original. A query runs much more quickly when sent to the abstract, as we will discuss in detail in the next sub-section. Abstracts thus may sacrifice accuracy for performance. In many of the applications which use large data sets, this is a reasonable trade-off. For example, the user may ask a question such as “Which regions of the earth have above-normal temperatures in this model output?” Obtaining an answer accurate to, say, a 50x50 km square within a small number of seconds (a reasonable interactive response time) may be preferable to waiting several minutes for an answer accurate to a 1x1 km square.

The idea of extracting information from files to accelerate certain types of access is becoming increasingly common. One example is semantic file systems, which pre-compute indices on the contents of files to allow users to quickly find files of interest in a large file system [GIFF91]. Examples of indexed attributes are keywords (for mail messages) and procedure names (for program source code.). Lossy image compression is another example of a specific type of abstract. In this case, the data is viewed on a computer screen, and the resolution and size of the display indicate the tolerable data loss. Abstracts generalize the idea of application-specific information extraction, offering condensed views of the data for arbitrary queries.

3.2 Potential Benefits

Abstracts can help to improve both effective bandwidth and latency to data. Provided we are able to answer the user’s question from an abstract, rather than having to go to the original data set, the benefits are twofold: Effective bandwidth increases, because we transfer much less data through the storage system (an abstract is far smaller than the original); and latency decreases, because many abstracts can fit in faster storage in the space occupied by a single unabstracted data set.

This double-barreled performance boost is the reason abstracts are defined to be so much smaller than their original data sets. With such drastically condensed data, we can overcome many of the obstacles to caching in a massive data storage environment. A typical query could easily take hours if it is run on a large data set that is stored on tape. Even if the large data set is cached on disk (unlikely, for reasons discussed earlier), the query could take many minutes, but retrieving an abstract from disk would supply the answer in fractions of a second.

We simulated the behavior of several storage devices to measure the potential performance improvements from abstracts. We assumed a 1000:1 size reduction, which we show to be feasible in Section 4. We ran simulated queries against a 70 GB data set (similar to a climate modelling output) and its 70 MB abstracts, and against a 300 MB data set (Thematic Mapper satellite image) and its 300 KB abstracts. We used the measured performance of the Exabyte 8500 tape drive in an Exabyte 120 Tape Library [CHER91] as the tertiary storage, and a composite disk drive technology as secondary storage. The results, summarized in Table 2, indicate a stunning speedup: factors of more than 3000 are possible!

TABLE 2. Time to execute a query when: (a) full data set is read from robotic tape library; (b) full data set is read from disk; (c) abstract is read from robotic tape library; and (d) when abstract is read from disk.

Query	Full Data Set Size (MB)	(a) Full Data Set on Tertiary (sec)	(b) Full Data Set on Disk (sec)	Abstract Size (MB)	(c) Abstract on Tertiary (sec)	(d) Abstract On Disk (sec)	Speedup (a)/(d)
Climate model output	70,000 MB	150,000 sec.	35,000 sec.	70 MB	220 sec.	35 sec.	4286
Thematic mapper image	300 MB	700 sec.	150 sec.	0.3 MB	75 sec.	0.2 sec.	3500

Table 2 demonstrates several points:

- Both bandwidth and latency are a problem for large data sets. It can take hours (or even days) to fetch a full data set from tape. (150,000 seconds = 1.75 days)
- Even when abstracts are stored on tape they offer performance benefits. In fact, abstracts on tape can be retrieved more quickly than the full-size data set stored on disk by factors of 2 to 160. The inherent latency disadvantage of tertiary storage systems can be overcome by the reduction in transfer time due to the smaller transfer size.
- Performance is even better when abstracts are cached on disk. Further, we can store many more abstracts than full-size data sets on disk, as we show in Table 3.

The dramatic speedup potential of abstracts indicates that hit rates on abstracts need not be especially high to improve throughput significantly. A hit rate as low as 60 percent (a value measured for some file caches [BAKE91]) would more than double the performance seen by the user, reducing average response time to a Thematic Mapper image query from 700 seconds to 280 seconds.

.

TABLE 3. Expected capacity of storage levels, and (approximate) number of 70 GB data sets and 70 MB abstracts which can fit in each level of a typical storage hierarchy

Storage technology	Expected capacity (GB)	Number of data sets	Number of abstracts
DRAM	1 GB	0.14	14
Magnetic disk	20 GB	0.3	300
Optical disk	500 GB	7	7000
Tape	10,000 GB	140	140000

Creating an abstract is equivalent to running a query over a full-size data set. As we have seen, this is highly time-consuming. If we have to create new abstracts frequently, then we incur a heavy performance penalty. Ideally, abstracts could be generated as a part of the process of bringing the data into the system [BERS88] [GIFF91], reducing cost by allowing many abstracts to be created without additional access to tertiary memory. As the access patterns from users change, however, we will inevitably need to re-evaluate the extent to which existing abstracts are able to intercept queries and avoid fetches from tertiary memory. The process of creating abstracts must adapt to the way users are using the data. We discuss our plans for measuring this in later sections.

4 Query Patterns

Of course, we can achieve the performance advantages of abstracts only if they are useful in answering users' queries. To evaluate this, we examined how users will make use of tertiary storage. In this section we discuss four common queries, providing an appropriate abstract for each. Note that we might store several different abstracts for a single data set. This is acceptable because tertiary storage costs are so low.

4.1 Data Mining

Many common queries of geographic information can be classified as requests for *meta-data*, information about a data set. For example, a user may want to know which of a set of satellite images has a particular characteristic, such as less than 50 percent cloud cover. It may be useful to know which regions of a data set conform to some specification; say, the areas of an image which have snow in them. Or it may be useful to have a histogram of the frequency of each data value over a region. The common characteristic is that these queries touch a high volume of data, but the generated answer is tiny by comparison.

In this case the abstract might be a low-resolution version of the images, over which an approximate cloud-cover calculation may quickly be made. If percentage cloud-cover is anticipated to be a common query, that percentage itself is a potential abstract of the data, reducing 300 MB of data to a few bytes of information. In the next section, we give a detailed evaluation of an abstract that can answer queries of this form.

4.2 Image Rendering

Users will frequently render a data set (image or otherwise) to the computer screen in order to view it as an image. This necessarily limits the size of the data which is of interest to the number of bits which can be usefully displayed on a monitor, typically 1-3 MB (or less, if compression is applied to the image), regardless of the size of the original which is being rendered. Thus, when a 300 MB satellite image is displayed, only a tiny portion is actually shown. We might store the rendered image, suitably compressed, as an abstract of the data.

The potential compression ratio is affected by such factors as the size of the window on the computer screen and the size of the original data set. Thus, a simple abstract would be to store the data that can be visually seen on a given display. More generally, we would like to be able to render images of regions of the data (see browsing), and in different window sizes or display types. Storing averages at various levels of granularity might be an effective way to address this.

4.3 Browsing

Users may frequently wish to browse data both in image form and in summary formats such as histograms. A user will browse a data set when he or she is not sure which portion of the data is of interest (or whether the data is of interest at all), and wants an overview. At some point, the user is likely to identify areas of the data in which the full-precision data is required; a fetch from tertiary storage may then required. A great deal of data is touched as a part of the browsing operation, but very little of it is actually needed in the result. Abstracts again can be of value in reducing the volume of data fetched and migrated from tertiary memory.

An example of a browsing operation might involve scanning the output of a climate model to find interesting weather-related phenomena which may occur anywhere over the surface of the earth. Once the user finds an area worthy of detailed study, the user can request the original (uncompressed) data for that region, and will only have to tolerate the slow fetch from tertiary storage for a small part of the data set. The abstract might be constructed as follows: each data point could be reduced from an 8-byte floating point number to a 1-byte integer, and 10 by 10 blocks of data points could then be averaged into a single point. This would yield a factor of 800 compression.

4.4 Transects

Some data, like the climate modelling output, is in the form of large multi-dimensional arrays (cubes) of data. Scientists using this information often extract slices of these cubes along various axes. For example, a user may wish to observe the simulated temperature patterns along some specific longitude at the same time of year for several consecutive years. Queries of this form are called *transects* by earth scientists.

These queries may call for widely dispersed pieces of the data set. Even if each piece is small, the high latency to read even a single byte from tape means that reading only the needed data may be no faster than reading the entire data set. Manipulating the layout of the data can help certain kinds of transection operations. For example, exchanging row-major for column-major organization of an array might ensure that a specific type of transect always accesses data which is contiguous. But the potential of this technique for improving access time is limited: only one type of transect can be served this way. In data cubes with more than 3 dimensions (e.g. time and the three spatial axes), transections might occur over more combinations of dimensions than can be served by a particular data layout.

Abstracts can be constructed to assist with transecting operations. For example, we might sample slices at regular intervals, and compress these slices to create an abstract which could answer queries similar to the example given above. Answers to questions like “What is the wind velocity during January of each year that we simulated?” are quite small; if such questions are anticipated, abstracts can capture their answers and help to avoid accesses to full-size data sets. One possible transect would involve averaging over 5 x 5 x 5 cubes of data points, and sampling every eighth time slice, giving a total size reduction of 1000.

5 Abstracting Functions

A single abstracting function for all data types would be no more applicable than a single lossy compression algorithm. Many applications can tolerate approximations in answers to their queries, but a single approximation would not be appropriate to all applications. A necessary step in building an abstracting system is the creation of appropriate abstracting functions.

On the other hand, it is reasonable to expect that some abstracting functions will work on more than one data type or in more than one environment. For example, image rendering is common enough that abstracting functions intended to optimize this operation may productively be applied to most of the data sets stored in a mass storage system. We have studied the statistical characteristics of several data sets in an effort to identify other classes of abstracting functions.

5.1 Natural Resolution

When an image is displayed at increasingly coarse granularity, eventually the viewer will be unable to distinguish the features in that image which are important to him. Similarly, an abstract which averages (or samples) over regions which are too large may not contain sufficient detail to provide useful information. On the other hand, if the "resolution" of the abstract is finer than necessary, then the overhead of caching and using the abstract is increased, and the abstract is less efficient than it might be. Establishing an optimal resolution for a data set that balances these conflicting objectives would thus be of some value.

The desired level of granularity is a function of the use of the data and by the nature of the data itself. We have found that the natural phenomena represented in our data sets tend to be auto-correlated. Autocorrelation refers to the degree with which a given element's value can predict the values of neighboring elements in a data set. In a sense, this is like natural locality: snow at point x at time t implies a high likelihood of snow at the neighbors of point x, and snow at point x at time t+1.

The auto-correlation of a data set can be used to predict a resolution that is likely to be of interest in a sizeable fraction of queries. Figure 1 shows how the variance between pairs of data points gradually increases with the *lag* (distance) between the points in the Thematic Mapper image used in our case study in Section 6. Low variance at a given lag (say, n meters) implies that a point can effectively predict other points that are n meters away from it. High variance implies the opposite, that a point n meters away can't be predicted. The graph in Figure 1 is called a *variogram*, and indicates the degree of correlation between points at a given distance. Reducing the resolution of the image through averaging (or some other means) is equivalent to moving further to the right on the variogram.

We can use the variogram of an image to estimate appropriate default levels of compression for that image. Compression factors which are below the knee (called the *sill* by earth scientists) of the variogram are reasonable first attempts at abstraction. Combining pixels with low variance between them causes relatively little information to be lost, as the pixels are likely to be similar to one another and to the computed average. If we compress an image by merging pixels whose mutual variance is small according to the variogram, we may still have a useful image. Combining pixels whose mutual variance is high discards more data.

FIGURE 1. *Variogram* of data for three of the six bands in a Thematic Mapper satellite image. Low variance (on the Y axis) indicates strong correlation between pixel values. As the distance (*lag*, along the X axis) between points grows, the correlation declines; thus the variance increases. Data with low variance at a given resolution may be abstracted to that resolution with little loss of accuracy.

6 A Case Study: Snow Cover in the Sierras

The resolution or precision of the data in an abstract is much lower than that of the data set from which it is derived. However, queries posed to the abstract can still give remarkably accurate results. We illustrate this claim with an analysis of a snow-finding query which we ran against both an abstracted and full-size satellite image. The original data is a 75 MB image of east-central California in January 1986 taken by the Thematic Mapper Satellite. The image consists of six monochromatic 4200 x 2982 pixel recordings, each taken at a different optical frequency. A pixel represents a square 28 meters on a side. Each of the six frequencies at which an image is stored is called a *band*. Band 2 (520-600 nm) of the image is shown in Figure 2.

The abstract was constructed by averaging 11 x 11 blocks of pixels, leaving one value where there had been 121. Naturally, this yields a 121:1 size reduction to a 622 KB data set. In addition, the nature of the query allowed us to ignore three of the six bands in the data set. Thus, the abstract was 311 KB. The Unix compress utility was then used to reduce the abstract to 230 KB. This gave a final reduction of 326:1.

The scenario that we tested calculated the portions of the image which depict snow cover. A pixel is believed to show snow if its measured radiance satisfies each of the following criteria [DOZI89]:

$$r1 > 0.16 \quad (1)$$

$$r5 < 0.25 \quad (2)$$

$$(r2 - r5) / (r2 + r5) > 0.4 \quad (3)$$

r*n* is defined to be the normalized reflectance (between 0 and 1) of a pixel in spectral band *n*.

These tests are computationally simple, and are really a special case of a more generic abstracting mechanism: selection by arithmetic preprocessing and boolean combinations of pixel values. For example, a normalized difference of infrared and visible reflectance ((IR - vis) / (IR + vis)) is used to map vegetation [JUST85], and could be used as an abstracting function to find vegetated areas.

Researchers studying climatic phenomena use snow-cover information to predict such phenomena as patterns of snowmelt and water runoff. A user might ask an abstracted Thematic Mapper image to locate snow (a browsing operation), and then follow that with a request for detailed information about the areas that contain snow for a small region of the image. The abstract is much smaller than the full-size data, but shows the boundaries of the snow-covered areas of the image almost as well as the original.

Figures 2-6 are at the end of the paper. Figure 3 shows the result of the query run against the original image. In order to show more clearly the boundaries of snow in a small area, in Figure 4 we focus on a small region of the image.

We ran the same query to find snow in the abstracted data set, and then we retrieved the abstracted data wherever snow was found. We masked off the abstracted image where snow was not found in that image. Figure 5 shows the result for the focus area of the image.

In Figure 6, we use the snow pattern (similar to that shown in Figure 4, but derived from abstracted data) to selectively eliminate regions of the original data set that do not have snow. Thus, Figure 6 computes essentially the same query that Figure 4 shows, but its mask is based on abstracted data. This might be the result of a request by the user to zoom in on an area of interest to get full-resolution data in snow-covered areas.

The images in Figure 4 and Figure 6 are quite similar, both visually and numerically. Significant differences, if present, would indicate that for this query, 11 x 11 pixel averaging is an inappropriate abstracting technique. Because the images are so alike, a user browsing the abstract would see greatly improved performance (because the abstract is so small, and may be in cache) and little degradation of accuracy.

Table 4 gives an indication of the accuracy of the abstracted images at a variety of averaged block sizes, from 3 x 3 to 35 x 35. Even at a compression ratio of more than 1000, only about ten percent of the pixels in the image are incorrectly computed to contain snow. At a more reasonable compression ratio of 121 (the 11 x 11 block size used earlier), less than seven percent of the pixels were in error. A query like “Where is the snow?” can be run against a highly compressed abstract with little loss of data.

A related query which users may run is “What percent of this image is snow-covered?” Improving the performance of queries of this form is of considerable value. In fact, a user may apply this question to each of the images available, to identify images which are of interest. An image with negligible areas of snow may not be useful in a study of snowmelt runoff, for example. We ran this query against the Thematic Mapper image and abstracts in which we averaged over blocks of pixels, again ranging in size from 3 x3 to 35 x 35. The results, also shown in Table 4, indicate that abstracts offer excellent accuracy for this query at compression ratios up to more than 300, again with a substantial performance boost.

TABLE 4. Accuracy of responses to queries for several levels of resolution in abstracted image. The queries are (a) “Does this pixel depict snow?” and (b) “How much of this image is snow?” In (a), 100% correct would indicate no loss of information in the abstracting process. In (b), the small deviations from the snow cover found in the original image (21.3%) indicate little inaccuracy.

Step size	Compression factor	(a) Percent of pixels correct	(b) Percent of image found snow-covered
Original	1	100 %	21.3 %
3 x 3	9	96.7 %	21.6 %
7 x 7	49	94.4 %	22.5 %
11 x 11	121	93.1 %	22.7 %
19 x 19	361	91.4 %	24.7 %
35 x 35	1225	89.4 %	39.8 %

7 Future Work

Without practical experience with real users posing queries to real data sets, we can only speculate on the effectiveness of using abstracts in a massive storage system. We are implementing our ideas on the Sequoia 2000 information server, a 10 terabyte data repository to be used by global change researchers from UC Berkeley, U. C. Los Angeles, U. C. Santa Barbara, U. C. San Diego, and the California Department of Water Resources, among others. We expect to answer the following questions from this prototype:

1. How high are hit rates? That is, how often can queries be answered by an abstract rather than requiring a scan of the full-size data set? And how often are the abstracts found on disk? Our simulations suggest that hit rates as low as 60 percent would more than double performance.

2. What types of queries do users actually run? This will evolve as people become accustomed to the system, and apply it to new tasks.

3. Can we produce abstracts during the initial loading of data into the storage system, thereby lowering the cost of their production? What is the incurred overhead of creating abstracts?

4. How do we identify useful abstracting functions for particular applications and users? A general abstract (e. g., a low-resolution picture of data) costs little to make, and may be useful to many people. A more specific abstract, such as “This image is 15 percent snow,” may take more computation time to create and is narrower in its usefulness, but is a perfect, lossless, tiny abstract when it can be used appropriately.

8 Conclusions

New tertiary storage technologies are now allowing systems designers to create information retrieval systems with capacities of many terabytes. The latencies of these technologies are higher than can be tolerated by users of the interactive applications that will run on these systems. We have introduced a new storage management technique, called abstracts, which compensates for the performance deficiencies of tertiary storage devices. Our preliminary experiments indicate that the throughput improvement offered by abstracts can be dramatic; it leads to the surprising result that an access to an abstract on tertiary storage is faster than an access to the full data set on secondary storage. When combined with caching and compression in a hierarchical storage system including magnetic disks, the benefits of abstracts on user response time are even more pronounced, giving the user the illusion of terabytes of storage with the speed of magnetic disks. Our case study suggests that abstracts will be useful for global change scientists, and we believe that other applications requiring massive storage will benefit as well. We expect that a prototype system which implements these ideas, now under development, will corroborate our preliminary results.

9 References

[BAKE91] Baker, M. G., Hartman, J. H., Kupfer, M. D., Shirriff, K. W., Ousterhout, J. K., “Measurements of a Distributed File System,” *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pp. 198-212, October 1991.

[BERS88] Bershad, B. N., Pinkerton, C. B., “Watchdogs: Extending the Unix File System,” *USENIX Association 1988 Winter Conference Proceedings*, February 1988.

[BURR91] Burrows, M., Jerian, C., Lampson, B., Mann, T., “On-line Data Compression in a Log-Structured File System,” DEC Systems Research Center Technical Report, 1991.

[CATE91] Cate, V., Gross, T., “Integration of Compression and Caching for a Two-Level Filesystem,” *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 200-209, April 1991.

[CHER91] Chervenak, A., “Performance Modeling of Helical Scan Tape Devices,” *submitted for publication*, University of California, 1991.

[CPM91] Committee on Physical, Mathematical, and Engineering Sciences, “Grand Challenges: High Performance Computing and Communications,” *Office of Science and Technology Policy*, Washington, D. C., 1991.

[DOZI89] Dozier, J., “Spectral Signature of Alpine Snow Cover from the Landsat Thematic Mapper,” *Remote Sens. Environ.*, vol 28, pp. 9-22, 1989.

[DOZI91] Dozier, J., Ramapriyan, H. K., “Planning for the EOS Data and Information System (EOSDIS),” *Global Environmental Change*, R. W. Corell and P. A. Anderson (eds.), *NATO ASI Series*, vol. 1, Springer-Verlag, Berlin, 1991.

[GIFF91] Gifford, D. K., Jouvelot, P., Sheldon, M. A., O'Toole Jr., J. W., "Semantic File Systems," *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pp. 16-25, October 1991.

[JUST85] Justice, C. O., Townshend, J. R. G., Holben, B. N., Tucker, C. J., "Phenology of Global Vegetation Using Meteorological Satellite Data," *Int. Journal of Rem. Sens.*, vol 6, pp. 1271-1319, 1985.

[KATZ92] Katz, R., Patterson, D., Chervenak, A., Fine, J., Miller, E., "An Approach to Cost-Effective Terabyte Memory Systems," *Proceedings of IEEE Compcon*, February 1992.

[LEGA91] Le Gall, D., "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, vol. 34, no. 4, pp 46-58, April 1991.

[LELE87] Lelewer, D. A., Hirschberg, D. S., "Data Compression," *ACM Computing Surveys*, vol. 19, no. 3, pp. 261-296, September 1987.

[MECH91] Mechoso, C. R., Personal Communication, October 1991.

[OUST85] Ousterhout, J. K., Da Costa, H., Harrison, D., Kunze, J. A., Kupfer, M., and Thompson, J. G., "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," *Proceedings of the 10th Symposium on Operating System Principles*, pp. 15-24, December 1985.

[SMIT81] Smith, A. J., "Long Term File Migration: Development and Evaluation of Algorithms," *Communications of the ACM*, vol. 24, pp. 521-532, 1981.

[SMIT82] Smith, A. J., "Cache Memories," *ACM Computing Surveys*, vol. 14, no. 3, September 1982, pp. 473-530.

[STON91] Stonebraker, M., Dozier, J., "Large Capacity Object Servers to Support Global Change Research," Sequoia Technical Report 91/1, 1991.

[WALL91] Wallace, G., "The JPEG Still Picture Compression Standard," *Communications of the ACM*, vol. 34, No. 4, pp. 30-44, April 1991.

[ZIV77] Ziv, J., Lempel, A., "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, Vol. 23, pp. 337-343, May 1977.

10 Figures

FIGURE 2. Band 2 of a Thematic Mapper satellite image from January 1986, showing east-central California. The six bands of the image total 75 MB.

FIGURE 3. Snow-covered regions of Figure 2. Areas which have no snow in the original image are masked off (black).

FIGURE 4. Snow near the Kern River (small area of Figure 3). Areas which have no snow are masked off (black).

FIGURE 5. Masked areas of abstracted data set showing snow near the Kern River. The mask was based on averaged pixel values in bands 2, 3, and 5 of TM image.This image might be used itself as an abstract for browsing the snow-covered areas in an image.

FIGURE 6. Masked areas of original data set showing snow near the Kern River. The mask was based on averaged pixel values in bands 2, 3, and 5 of TM image. This image's similarity to Figure 4 indicates that little accuracy was lost in finding snow based on the averaged pixels rather than the original image.