Storage control cache resource management: Increasing diversity, increasing effectiveness

by D. A. Burton
B. McNutt

Efficient management of cached storage control resources has been important since the introduction of cached controllers in the early 1980s, and it continues to grow more important as technology advances. The need for cache resource management is due to the diversity of workloads that may coexist under a given controller. Some workloads may continually require the staging of new data into cache memory, with almost no benefit in terms of performance; other workloads may reap major performance benefits while requiring relatively little data staging. The sharing of resources among various workloads must therefore be controlled to ensure that workloads in the former group do not interfere too much with those in the latter. Management of cache functions is often viewed as the job of the host system to which the controller is attached. But it is now also possible for advanced controllers to perform such management functions in a stand-alone manner. Caching algorithms can change adaptively to match the workloads presented. This enables the controller to be ported across multiple platforms without dependencies on software support. This paper surveys the variety of techniques that have been used for cache resource control, and examines the rapid evolution in such techniques that is now occurring.

Introduction

Since the first cached DASD controllers were introduced in the early 1980s, users of such controllers have observed that data with poor locality of reference can interfere with the service provided to the remaining data under the

0018-8646/96/$5.00 © 1996 IBM

controller, because of the added demand that such data place on cache resources. Each cache miss typically causes a stage operation, in which the requested track is transferred into the cache. If the staging rate for a subset of the data served by a cache is high, this may cause high loads on the disks containing the data and on the staging paths. Such loads (especially high loads on the staging paths) can, in turn, interfere with the service provided to data that otherwise would cache well.

Contention can also occur for cache memory. In the extreme case, the cache can be "flushed" by a rapid succession of misses to data possessing no locality. In this case, new data that do not benefit from the use of cache replace older data that may have profited from cache storage. Contention for memory is usually not as extreme as in the example just described. It does, however, tend to limit the effectiveness of the cache in cases where data with poor locality are present, especially if the cache size is small.

Since contention for memory and staging path resources can interfere with the effectiveness of the cache, cached controllers must manage these resources so as to mitigate the effects of contention. Techniques for resource management have become increasingly sophisticated. Initially, the focus was on management via host-initiated actions. These included the selective suppression of caching for some volumes, and the use of Dynamic Cache Management (DCM) and Dynamic Cache Management Extended (DCME) on MVS systems. More recently, an important trend has been to incorporate more and more of the decision-making into the controller itself. This paper examines the range of techniques that have come into use, with a focus on adaptive strategies for use of cache memory.

The next section begins by sketching the techniques for resource management that are based on control by the host. These techniques, and the use of storage control cache in general, have typically been applied in large-system environments.

As cached controllers have become more common in small-system environments, the need to manage cache functions without the assistance of host programs has increased. The ability to adapt caching algorithms within the storage controller enables the cache to be ported across multiple platforms without dependencies on software support. The section on self-controlling caches describes various techniques that have been used to accomplish this in recent storage products such as the IBM 9340, the EMC^2 Symmetrix™, and the IBM RAMAC™.

Technologies such as RAID and log-structured arrays [1] place additional demands on the controller. These demands will further increase the complexity of the algorithms required to effectively manage the resources in a cached DASD controller. The section on trends and speculations attempts to predict the future impact of such technologies on cache resource management. The final section traces some of the common themes that are apparent in the diversity of approaches being surveyed.

Host-directed resource management: 1982 to the present

This section presents a brief history of cached storage controls for large systems. The emphasis is on storage controls designed for use with the MVS, VM, and VSE operating systems, and particularly on the use of these storage controls under MVS. We consider first the storage control hardware, then discuss the management of this hardware by the host.

• *Hardware resources*

Until very recently, cache hardware could be grouped into two separate families: cache for storing complete DASD tracks, and cache for storing individual records. We consider track cache first, since (until very recently) it has been the standard for use in general large-system environments.

Track caching

The first IBM DASD control unit featuring track cache was the 3880 Model 13 Direct Access Storage Control, introduced in 1982. This control unit provided a very small cache by today's standards, ranging in size from 4 to 16 megabytes. When a record not already contained in memory was referenced, the 3880 Model 13 staged this record into cache, as well as any other records that followed it on the same track. This method of staging has since been incorporated into all general-purpose IBM storage controls.

As has also become the standard practice, 3880 Model 13 managed its memory space using a simple least-recently-used (LRU) algorithm. This algorithm keeps a list of the data items stored in memory (the LRU list). The list is maintained using pointers, so that the positions of items on the list can be changed without requiring movement of the data. When a given data item is referenced, it is removed from its current position on the LRU list, if any, and placed at the top of the list. Space required for inserting new data items is allocated from the bottom of the list, thereby overwriting the oldest (the "least recently used") data. The effect of the LRU algorithm is to retain the most recently referenced data, up to the limit of the available memory.

The introduction of the 3880 Model 13 was accompanied by the appearance of cache "hints" in channel programs. These hints tell the controller how a particular access can best be handled. They are supplied to the controller as part of the Define Extent command;

this command, in turn, precedes the commands that call for actual transfer of the data. The hints supported by the 3880 Model 13 included *inhibit cache load, bypass cache*, and *sequential.*

Inhibit cache load is used for data predicted to exhibit poor locality of reference. The controller will check the cache for the presence of the data, but if they are not found, will operate directly from the device and will not place the data into cache memory. The *bypass cache* hint is more special-purpose. It is typically used by programs that are performing media maintenance and require direct access to the device, or sometimes by applications which transfer large amounts of data in a single channel program.

Sequential is used to indicate that access is expected to continue to subsequent tracks in sequential order from the current reference point. This hint enables the control unit to stay ahead of the program by prestaging data. In this way, all of the accesses associated with transferring a file (except the first) can be handled as hits. Also, the control unit can remove data from the cache after transfer of the data is complete, so as to save memory. The exact handling of the *sequential* hint varies with the control unit. The 3880 Model 13 prefetched a single track; subsequent control units have prefetched larger amounts of data.

A variation of the *sequential* hint, called *sequential prestage*, was added subsequent to the 3880 Model 13. This variation requests that the controller *not* remove the data after completion of the transfer. Instead, the data are allowed to age out of cache via the normal LRU mechanism. (By using first the *sequential prestage* hint, followed by *sequential*, it is also possible to retain the data in cache only until the second reference to the data, then allow the controller to remove the data following the second reference.)

The Model 13 lacked the ability to "fork" data. When data were read, they would be transferred from the device twice. The first read would satisfy the original request by transferring the data to the channel; a second read would then transfer the same data, plus the remainder of the track from the point of reference, into cache memory. When a write hit occurred in the Model 13, the data would first be written to the device, and would then be staged from the device into cache memory. This scheme could place significant loads on the available staging paths, of which there were fewer available than on current controllers.

Specific workloads could benefit a great deal from using the 3880 Model 13 cache; however, the device and path activity due to staging could also degrade performance for some types of data if cache hit ratios were not high enough (for the 3880 Model 13, hit ratios of 70 percent or higher were usually needed for effective operation). To allow control over which types of data should use cache, the Model 13 provided the capability to suppress cache functions at the volume level; i.e., cache could be "turned off" for any specified volume or set of volumes.

The IBM 3880 Model 23 Direct Access Storage Control, introduced in 1984, increased the maximum cache size to 64 megabytes [2, 3]. Also, this control unit introduced the capability to "fork" data: to place data into cache memory at the same time they are being transferred to or from the device. While a read miss operation was being serviced, the data could now be staged into the cache while being sent to the channel. Write hit operations could now update the cache copy of the data while the data were being transferred from the channel to the device.

The forking capability and added cache memory of the 3880 Model 23 greatly reduced the performance penalty associated with misses, compared to the Model 13. This broadened the range of workloads for which cache could be used effectively, and reduced (but did not eliminate) the need to selectively suppress cache functions at the volume level.

The IBM 3990 Model 3 Direct Access Storage Control, introduced in 1988, added several new features important to cache management. These included segmented cache memory, a larger cache (256-megabyte maximum), and (in 1989) nonvolatile storage (NVS).

The segmented memory management of the 3990 Model 3 allowed it to allocate cache in units less than a full track slot, more efficiently using available cache space. Prior control units had allocated track space in track slot sizes.

NVS allowed the 3990 Model 3 to store write data in memory, as a shortcut to writing them to disk (the write to disk could now occur later, after the completion of the I/O). Such a shortcut was not possible in the 3880 Model 13 or Model 23, since a power failure would have resulted in loss of the data before a permanent disk copy could be made. The NVS retained a duplicate copy of written data until the completion of the disk copy operation. The NVS copy was used only in the event of a power failure or a failure of the volatile cache memory. By drawing upon an emergency battery-based power supply, the NVS was able to retain data for 48 hours in the event of a power failure.

Two "hints" were introduced for controlling the use of NVS. The handling of writes, as just described, was provided by default. This capability was called "DASD fast write." The *inhibit DASD fast write* hint could be used to perform a write operation directly to disk. The *cache fast write* hint could be used when the use of *volatile* memory (rather than NVS) was acceptable (for example, *cache fast write* is often used for sort work files, since such files can be reconstructed if they are lost because of a power failure).

The 3990 Model 3 could provide handling at electronic speed, via DASD fast write, for any write to data already in the volatile cache. In the case of a write to data which

could not be found in cache, the 3990 Model 3 would stage the track so that the validity of the write operation could be verified.

The introduction of NVS and DASD fast write opened a new set of cache management issues revolving around the management of written data not yet copied to DASD (sometimes called "dirty" data). The 3990 Model 3 coordinated a highly granular NVS storage allocation scheme with a destage mechanism that could speed up or slow down as needed. These allowed the Model 3 controller to make the most out of a limited, four-megabyte NVS area.

The IBM 3990 Model 6 Direct Access Storage Control, introduced in 1993, extended the design of the 3990 Model 3 to larger sizes of cache and NVS. More interestingly from the point of view of cache resource management, the 3990 Model 6 became the platform for IBM's introduction of Record Cache for use in general-purpose environments.

Record caching

So far, our discussion of cache algorithms has focused on a track-based scheme for staging and memory management. For random or nearly random access patterns, a more finely grained management unit than the track is often effective. If record caching is used instead of track caching, the device busy times and path use per cache miss are reduced, as is the amount of cache memory required to store the staged data. Until very recently, such record caches have been provided only in very specialized environments. More specifically, such caches have traditionally been used for paging (the IBM 3880 Model 11 Direct Access Storage Control and Model 21 Direct Access Storage Control), and for high-throughput transaction processing performed by the Transaction Processing Facility (TPF) operating system.

When operating in these specialized environments, a record cache is used for known fixed-size records. The controller operates explicitly in record caching mode and does not support track caching. This usage is not appropriate in a general-purpose environment, but provides advantages if there is very little spatial locality in the data.

In cases of very little locality, a request for a particular record on a track is not likely to be followed by requests for other records on the same track. The overhead of staging the remainder of the track, and the space requirements to store it, are therefore likely to be wasted. Record caching avoids these forms of waste, while still allowing fast repeat access to individual records.

In March 1994, a record cache capability for general MVS environments (Record Cache I) was added to the 3990 Model 6. The use of Record Cache I was controlled by "hints," as discussed in the following section. Less than a year later, the Record Cache II feature was introduced. This permitted the 3990 Model 6 *itself* to control the use of record cache.

As part of Record Cache I, the capability to accept writes at electronic speed, even for data not in cache, was added to the 3990 Model 6. This was done initially via a host "hint" to indicate that there was no need to check the format of the affected track before accepting the write. The requirement for this hint regarding the format of the written data was also eliminated within a short time after the initial introduction of Record Cache I. The 3990 Model 6 now retains, in cache memory, a table of those tracks that conform to regular, stereotyped formats. These include the tracks of almost all databases. The controller accepts writes to such tracks at electronic speed, whether or not the track is in cache.

- *Host resource management*

The mechanisms for management of the hardware resources just outlined have been twofold:

- Use by the system programmer or others of volume-level cache suppression.
- Use by the host software of channel program "hints" to indicate to the controller which references should be staged in the event of a miss.

We now discuss in more detail the use of hints.

Ever since the introduction of the 3880 Model 13, hints have been used to choose the type of caching appropriate for specific, special applications such as sequential processing, sort, utilities, and so on. More recently, however, hints have been applied to the problem of dynamically controlling cache operation.

Dynamic control, in MVS environments that use System Managed Storage (SMS) file management, has been provided by the Dynamic Cache Management (DCM) function introduced in 1990, and by the Dynamic Cache Management Extended (DCME) function introduced in 1993. These software facilities use hints to indicate to the controller which references should be staged in the event of a miss. If a reference should not be staged, the *inhibit cache load* hint is given to the cache. The selection of data sets that should be supported by staging on a miss is based on the performance priority of each data set (as indicated by its SMS storage class), on the past effectiveness of the cache in supporting the data set, and on current controller load levels.

If a 3990 Model 6 storage control is present, DCME may also instruct the storage control, via channel program hints, to use record caching for some data sets. This is done if little locality appears to be present in the access pattern of the data set. At high subsystem loads, increasing use is made of record cache as a mechanism

for reducing the demands on storage control memory and staging paths.

For its part, the 3990 Model 6 now keeps a history of the patterns of locality exhibited by each track. If a given track seldom receives references to more than one of its records during a visit to the cache, the 3990 Model 6 stages the track in record mode even if not instructed to do so by DCME.

- *Summary*

Various cache management techniques have been successfully used for over a decade now. For most of this time, caches have been managed via host control. With hints in the Define Extent command, the host can collect cache statistics, analyze cache behavior, and control cache resources in real time.

Such real-time host control, however, is offered mainly in MVS environments running under SMS file management. Dynamic cache control using host hints is much more limited on non-MVS platforms or in MVS environments which are not yet taking advantage of SMS file management.

Self-controlling caches

As technology advances, control units are becoming more and more intelligent. The available computing power in control units has increased to the point that controllers are now capable of doing more than just reading and writing disks. A desire to provide cached controllers for non-MVS environments has led to the development of a variety of approaches to providing effective control of cache resources without host software support. This section surveys the approaches taken in various I/O subsystem offerings that have come on the market during the early 1990's (other than the 3990 Model 6, just described in the previous section). Our sampling includes the IBM 9340 Direct Access Storage Subsystem, the EMC^2 Symmetrix 4800 and 5500 Integrated Cached Disk Arrays [4, 5], and the IBM 9394 RAMAC Array Subsystem.

- *The 9340 subsystem*

The IBM 9340 Direct Access Storage Subsystem was introduced, in 1992, mainly for use in small, non-MVS environments, where use of storage control cache was not as prevalent as it is for large systems. Initially the controller built into this subsystem did not offer a caching function, but potential customers of the 9340 quickly made it clear that cache capability was now a requirement even in small-system environments. Also, it became apparent that a cached version of the 9340 would be useful in some large-system environments (particularly ESCON® environments). Cache capability was therefore added to the 9340 subsystem shortly after its introduction. The 9340 cache function, as well as the management of cache resources, was provided as a transparent function of the controller so as to avoid the need for host support.

The 9340 controller provided 64 megabytes of cache memory. A cache of this size requires good memory management or it can easily be overcommitted by data with poor locality of reference. Data which can use cache memory effectively must able to do so without being adversely affected by "cache-hostile" data (data with hit ratios so low that the resulting cache misses would flood the cache and clog the staging paths). The Adaptive Cache Management facility was incorporated into the 9340 subsystem as a way to provide the required cache control *and* the required transparency relative to host software.

The 9340 adaptive cache algorithm associated a particular caching behavior with a particular location on DASD. The controller maintained cache performance and utilization statistics, by device, for a series of fixed cylinder bands. This allowed the controller to identify cylinder ranges corresponding to data sets with poor locality of reference and to disable cache usage for those cylinder ranges.

The adaptive caching algorithm allowed the 9340 to operate efficiently in a wide range of environments, removing the need for manual tuning or host hints. Despite the small amount of available cache memory, the algorithm proved effective even in handling traditionally "cache-hostile" applications such as DB2®.

- *Symmetrix*

Another approach to the cache management problem was used by EMC^2 Corporation in their Symmetrix series (the Symmetrix 4800 Integrated Cached Disk Array [4] and Symmetrix 5500 Integrated Cached Disk Array [5]). The Symmetrix controllers provide plug compatibility with the IBM 3880 and 3990 controllers. The Symmetrix controller accepts count-key-data (CKD) channel commands of the type sent to 3880 and 3990 storage controls, then "emulates" these commands so that they can be performed on fixed-block architecture (FBA) devices attached to the controller via a SCSI interface. To accomplish the emulation function, all tracks are brought into the cache, even those with cache hints that would otherwise suppress staging of such tracks.

The Symmetrix philosophy has been to allow for the availability of very large cache sizes in the design. The Symmetrix series controllers were the first to offer the capability to configure a gigabyte or more of cache memory (other DASD vendors have since followed this example). The capability to install very large cache sizes provides Symmetrix controllers with the needed protection from data with poor locality, since a very large cache is more difficult to "flush." Also, if path or device loads become too high, there is a good chance that they can be

reduced (along with the miss ratio) by taking advantage of the cache size increases that are possible.

Extensive cache memory is also deployed in Symmetrix controllers to provide auxiliary tables that assist with CKD emulation. These tables incorporate a full description of all track formats (not just the tracks currently in cache). One benefit of such tables is that they allow the controller to validate write misses without having to perform a DASD access, thus avoiding the long service delays which would otherwise be required by the emulation scheme to perform a CKD write miss on the FBA DASD.

The Symmetrix series of controllers use *battery backup* to handle dirty data. This differs from traditional NVS in that, in the event of a power failure, batteries are used to operate the entire I/O subsystem, including both cache memory *and* the associated DASD, for long enough to make permanent copies of all dirty data. Thus, a battery-based backup for the power supply can be provided without requiring a separate area of memory, such as NVS, that can retain data for the duration of an extended power outage.

The Symmetrix controllers retain a single copy of dirty data in cache. The amount of dirty data which is accepted in the controller is adjustable so as to accommodate differing policies for the management of the string in the event of a power failure.

- *Two-level cache designs (RAMAC)*

Yet another approach to cache resource management was recently introduced with RAMAC. RAMAC involves the packaging of a RAID-5 disk architecture [6] in the form of a DASD *drawer*. All control functions needed for RAID-5 are incorporated into the drawer. The drawer uses FBA disks attached via a SCSI interface, but performs the emulation functions needed to make these appear to be CKD devices from the point of view of the storage control. Depending on the method of RAMAC attachment, the storage control may be a 3990 Model 3 or Model 6 (for RAMAC Array DASD) or a controller packaged in the same rack as the disks (for the RAMAC Array Subsystem).

Each RAMAC drawer incorporates 64 megabytes of cache memory. Because of the presence of drawer cache, a miss from the point of view of the storage control may still actually be a hit. This applies to both reads and writes; the RAMAC drawer cache is equipped with battery backup and can accept "dirty" data for later destaging.

An interesting aspect of the RAMAC design is that it imposes a sharply reduced performance penalty for staging operations. Sufficient data transfer capability is provided in the drawer so that *all four actuators* can simultaneously stage data, with no mutual interference. Therefore, track-level caching can continue to be provided in the drawer even for data that are unsuitable for storage control track caching because of lower path considerations.

The RAMAC Array Subsystem (with integrated DASD and controller) is an extension of the 9340 design, in which the drawer previously containing two 5.25-inch CKD volumes is replaced by a drawer with four 3.5-inch FBA devices. In addition to the cache storage provided in each drawer, a RAMAC subsystem rack can hold up to two controllers, each with up to two gigabytes of cache memory. For redundancy, dirty data are retained in both the drawer and controller caches.

The controller cache of the RAMAC Subsystem uses an adaptive caching algorithm which is very similar to that of the 9340, except that the algorithm incorporates selective record caching. Subsystem decisions about whether to apply track or record caching within a given cylinder range are guided by statistics indicating whether the I/O pattern typically includes requests for distinct records on a given track. If such requests are rare, the available cache memory can be used more efficiently by adopting record cache mode. Thus, the RAMAC subsystem is able to employ record cache, not just in those situations where it is needed to limit the consumption of storage control resources, but also in those situations where it may result in increased hit ratios.

RAMAC DASD provides a similar drawer for use as a RAID attachment to the 3990 Model 3 or Model 6 storage controls. The storage control algorithms of the Model 3 or Model 6 are retained essentially intact.

Trends and speculations

The overview of recent products, as just presented, exhibits a remarkable diversity of cache management strategies. It is apparent that, despite the declining cost of memory, management of memory and path resources continues to present an interesting challenge that requires evolving solutions. In this section, we attempt to organize the current trends in storage control resource management into an overall pattern, by examining the problems that the next generation of storage controls must address and the types of solutions that appear to be practical.

In the coming several years, I/O subsystems will continue to offer increasing amounts of both disk storage and cache memory. But the current pace of advances in disk technology is so rapid that technologies for cache memory may be hard pressed, in the next few years, just to keep up. Although the amounts of both in a typical I/O subsystem will continue to increase, we should not necessarily expect continued growth in the *ratio* of cache memory relative to disk storage.

At the same time, the demands placed upon cache memory and staging paths continue to increase. The resulting pressure on memory resources will thus continue to stimulate new memory management techniques. The

added need for memory comes about for a variety of reasons.

CKD emulation and RAID-3 or RAID-5 storage management both require more cache memory per I/O handled than do traditional disk subsystems. For practical purposes, these architectures (unlike traditional subsystems) demand that a cache be used to hold data while the required translations, merges, XOR operations, and so forth are carried out. *Universal staging* is therefore needed; all data must be brought into the cache, rather than selective data, as in traditional subsystems.

In the new disk management schemes just outlined, efficient handling of writes may impose additional memory demands. For maximum data integrity, "dirty" data must be duplicated to avoid the possibility of data loss due to cache memory failure. Delays in handling physical disk writes, due to CKD emulation or RAID-5 disk management, will also tend to force all writes to be accepted at electronic speed, *even if they are misses*. A storage control that accepts all writes in this manner must retain extensive information in the cache about the formats of *all* of the data stored on disk (not just the cached data), since a write cannot be accepted without verifying that it represents a valid update of the existing data format.

RAID-3 and RAID-5 disk subsystems spread the I/O activity of the data in a parity group across all of the disks of the parity group. This contrasts with conditions in a traditional disk subsystem, where an equivalent amount of load may fall predominantly on a single disk volume because of "skew" [7]. Since RAID-3 and RAID-5 subsystems mitigate skew, they make possible higher subsystem loads. Thus, disk arrays of these types may sometimes place higher demands on cache memory than traditional DASD, because of the higher loads that can be achieved.

The increasing sophistication of memory and path resource management strategies that is now occurring is not driven solely by increasing demands for memory, however. In addition, the rules of the game are changing. As just discussed, universal staging as required for some new types of subsystems precludes the option of not staging selected tracks. Traditionally, this option has been central to cache management. In addition, subsystem packaging is becoming much more flexible; a cache management strategy may have to account not for a single, centralized memory, but for a memory hierarchy that includes both controller and drawer caches. The following sections speculate about the implications of these trends in more detail.

- *Universal staging*

A storage control with universal staging cannot apply the most powerful technique that has traditionally been available to prevent overload of storage control resources; it cannot suppress staging of selected data. We now consider alternatives to selective stage suppression. Alternatives must be found both for the control of staging load and for the management of memory use.

Control of path load

In designing a disk subsystem to provide universal staging even for workloads with high miss rates and extreme staging requirements, one option is to overwhelm the problem with hardware. We might choose to build so much data transfer capability into the staging paths that the paths cannot be overloaded, even if all requests are misses. This happy state of affairs is actually the case for the *drawer* caches of RAMAC. But for a centralized cache that serves the entire subsystem, manufacturers do not yet appear to be at the point where this amount of transfer capability can be provided in a cost-effective manner. For the time being, at least, it is necessary for a well-rounded cache management strategy to incorporate some response to miss rates that the staging paths cannot sustain.

If all requested data must be staged, the control of staging load must be accomplished by ensuring that *only* the requested data are staged. Traditional cache designs have either staged the entire track on which the requested disk record appears, or else the portion of the track from the position of this record through the end of the track. By contrast, in future cache designs it is desirable to be able to selectively stage the individual disk record that has been requested, if this is necessary because of an overload of path resources.

It is likely that the subsystem served by a given cache contains a mixture of tracks that exhibit locality and can benefit from track caching, as well as tracks which have little locality and should be staged on a record basis if paths are overloaded. Therefore, we may have to pay a price if we need to switch the entire cache from track-staging mode to record-staging mode in order to conserve path resources. Such a switch of staging mode may sharply reduce the hit ratio because of its impact on data with track locality.

To maintain performance for data with track locality, make the most of limited path resources, *and* provide universal staging, data which can benefit from track caching must be staged on a track basis, while other data must be staged on a record basis. An example of this type of selective record staging is the Adaptive Record Cache algorithm described above for the RAMAC Array Subsystem.

Control of memory

The capability to selectively implement record, rather than track, staging for individual ranges of data also provides a way to conserve on the use of cache memory. This is

because an individual disk record typically requires an order of magnitude less storage than does a full track. Even with record caching, however, a rapid succession of misses to a specific range of data can still have the effect of "flushing" the cache.

To control the use of memory resources when all data must be staged, one alternative is to partition the cache memory [8]. This ensures that, if one memory area is flushed, the impact is contained within that specific area.

Another approach is to generalize the traditional concept of the cache LRU list. In such a generalized LRU (GLRU) list, insertion is allowed at points other than the top. If a specific range of data is currently exhibiting a high miss rate, but if data from this range must still be staged because of universal staging, all staged data from this range are inserted at a point near the bottom of the GLRU list. In this way, only other entries near the bottom of the list, whose probability of future hits is low, will be flushed.

Some interesting insights into the management of a GLRU list can be gained by applying a statistical model of cache locality called the *hierarchical reuse* model [9]. This model provides guidance as to the appropriate insertion points into the GLRU list by examining the corresponding *single-reference residency times* (the times required for data inserted at these points to reach the bottom of the LRU list and be removed from the cache).

The key assumption of the hierarchical reuse model is that data references are caused by a series of hierarchically related processes. For example, repeated references to a specific track may occur within the same subroutine, within different routines called by the same transaction, or as part of an overall task that involves several transactions. For this reason, the hierarchical reuse model predicts that the probability of a repeat reference to a given track in the immediate future is directly related to the length of time since the last reference (the former is assumed to be inversely proportional to the latter). This conclusion of the model validates well against I/O trace data. The degree of locality for a particular collection of data (the speed with which the probability of a repeat reference drops off with time) is reflected by the model parameter θ. For track caching, this parameter typically lies in the range $0.2 \leq \theta \leq 0.3$, with higher values reflecting stronger locality.

By applying the hierarchical reuse model to a set of distinct data sets or data ranges, each associated with a specific value θ, it is possible to determine the best GLRU insertion point for each data set or data range. As it turns out, the best insertion points are those for which the respective single-reference residency times of the various data sets or data ranges are proportional to the values θ. Moreover, the value of θ associated with a given data set or data range is easy for the storage control to estimate. According to the hierarchical reuse model, the complement of θ is equal to the ratio of the single-reference residency time, relative to the average residency time, for the corresponding data set or data range.

The advantage of cache memory control via the GLRU list, therefore, is that specific collections of data are retained in the cache no longer than they should be on the basis of their statistical behavior. A range of data with poor locality cannot flood the cache, since individual records or tracks from this range are not retained long enough to accumulate in cache memory.

- *New subsystem packaging*

The availability of two cache levels, as provided by RAMAC, creates a variety of new options and decisions for the storage control. The two memory levels have distinctly different performance attributes. Controller cache has the advantage of a shorter delay in placing data onto the channel, while drawer cache has the advantage of virtually unlimited staging capability.

The best way to deploy memory resources in these two caches, for use by specific data sets or data ranges, depends upon the access patterns of the data, the amounts of memory resource available in each cache level, and the overall levels of subsystem load and path use. For example, if path use is light, it may be desirable to load full tracks into the controller cache for data in a specific range, so as to take advantage of track locality by providing minimum service time when a track hit occurs. Under heavier loads, however, a more appropriate strategy for the same range of data may be to stage records into the controller cache, while servicing track hits (that are not also record hits) out of the drawer cache. In the future, increasingly sophisticated heuristics will undoubtedly evolve for making these types of decisions.

It is possible, however, to sketch a general strategy of operation that appears likely to work well for most data, especially under conditions of high controller load. This strategy is to create a division of labor between the two memory levels, so that each level attempts to provide different types of hits. The drawer cache is used to stage large quantities (full tracks or more) of data, thus providing hits if the reference pattern refers to several related records that are close to one another. The controller cache, by contrast, stores primarily individual disk records, which require much less storage than do full tracks. In this way, the controller cache provides much longer residency times than does the drawer cache, and obtains as many hits as possible because of these longer residency times. When operated in this way, a two-level cache hierarchy can virtually eliminate staging load as a performance bottleneck.

- *Evolving role of the host*

If future storage controls are able to implement memory and path management strategies as sophisticated as some of those just outlined, this raises the question of whether the host will be left with anything useful that *it* can do to promote efficient memory and path use. The answer is that "hints" provided by the host will continue to be valuable to the storage control; in the future, however, the nature of such hints is likely to evolve.

If we now enter "blue-sky" mode, it is not difficult to imagine the types of hints which could be used effectively by a very advanced storage control. Strategic, rather than tactical, information will be needed. Such information might include

- Data set boundaries, so that the "data ranges" discussed above can correspond to data sets or to other logical groupings of data as known by the application.
- Priority information; e.g., "data set XYZ is performance-critical," or "the response time objective of data set ABC is 12 milliseconds."
- Information about major events occurring in the host; e.g., "data set CICSDB has just been closed"; or "sequential data set LIST has just been opened for reads."

Some or all of these types of "hints," or even perhaps others, may evolve over the coming years.

Summary and conclusions

From the beginning of the period covered by our survey, the introduction of the 3880 Model 13 cached storage control in 1982, the need to manage the use of staging paths and memory has continued to make itself felt. This requirement has not disappeared in the years since 1982, despite the dramatic improvements in memory size and staging capability which have been provided in subsequent storage controls. Instead, storage control cache has gradually become, not just cheaper, but also smarter.

Initially, tactics for resource management revolved around the suppression of caching for specific volumes, and the use of host hints that allowed access methods to control cache as required for the type of access (random, sequential, media maintenance, and so forth). The power of host hints to achieve dynamic cache resource control was considerably enhanced when, in 1990, MVS systems provided the capability to manage such hints using SMS.

The newest generation of storage controls are now smart enough that they can both use host hints and make their own independent decisions. This is particularly important for storage controls that serve smaller systems, in which SMS management is not in use.

We have surveyed a variety of management strategies that have been adopted by the most recent generation of storage controls. The IBM 9340 acted adaptively to suppress staging of selected cylinder ranges, thus making the most out of a relatively small memory; the EMC^2 5500 relies mainly on large memories, without the capability to suppress staging activity; the IBM RAMAC Subsystem dynamically selects among track staging, record staging, and stage-suppression in the storage control while always staging into the drawer.

Despite the diversity of strategies, some ground rules are evident. In the future, it will increasingly be the job of the storage control to manage the use of its own resources. Information from the host will tend to be used for strategic insights (e.g., priorities), rather than directly to control cache functions.

Also, advanced subsystem architectures such as RAID-3, RAID-5, and log-structured subsystems with compression will virtually demand that cache be used for servicing all I/Os. This means that innovative methods must be found for reacting to overloads of the staging paths or memory. These may include record caching, or the insertion of data at intermediate points in the LRU list.

Finally, storage controls must become expert at statistical data gathering and analysis. Detailed statistics must be maintained and analyzed, either at the level of ranges of data or for individual tracks. Future gains in I/O performance will be made possible, not just by advances in disk, memory, and processor technology, but also by the effective gathering and use of such statistics.

References

1. M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Trans. Comput. Syst.* **10,** No. 1, 26–52 (February 1992).
2. "An Analysis of IBM's 3880 Controller: Paving the Way for the Management of Active Data," *Report No. IDC-3011*, International Data Corporation, Framingham, MA, December 1986.
3. "Issues in Large Scale Storage," *Report No. IDC-2919*, International Data Corporation, Framingham, MA, April 1986.
4. *Symmetrix 48XX Series User Guide*, EMC^2 Corporation, Hopkinton, MA, February 1992.
5. *Symmetrix 5500 Product Description Guide*, EMC^2 Corporation, Hopkinton, MA, 1993.
6. D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of ACM SIGMOD 88*, Association for Computing Machinery (ACM), New York, June 1988, pp. 109–116.

7. B. McNutt, "An Empirical Study of Variations in DASD Volume Activity," *Proceedings of the Computer Measurement Group*, December 1986, pp. 274–283.
8. D. Thiebaut, H. S. Stone, and J. L. Wolf, "Improving Disk Cache Hit-Ratios Through Cache Partitioning," *IEEE Trans. Comput.* **41,** No. 6, 665–676 (June 1992).
9. B. McNutt, "A Simple Statistical Model of Cache Reference Locality, and Its Application to Cache Planning, Measurement, and Control," *Proceedings of the Computer Measurement Group*, December 1991, pp. 203–210.

Received February 20, 1995; accepted for publication March 14, 1996

David A. Burton *IBM Storage Systems Division, Tucson, Arizona 85744 (burton@vnet.ibm.com).* Mr. Burton is a senior engineer with the advanced controller development group. He is currently the team leader for the cache component of the microcode development team. He received a B.S. degree in electrical engineering from California State University, Chico, in 1988. Mr. Burton joined IBM in 1988 at the development laboratory in San Jose, California, where he worked on the system adaptor and caching functions for DASD control-unit microcode development. He is the author of several patents and technical papers in the area of cache management algorithms for DASD storage control units.

Bruce McNutt *IBM Storage Systems Division, 5600 Cottle Road, San Jose, California 95193 (b_mcnutt@vnet.ibm.com).* Mr. McNutt is a senior engineer–scientist working in the area of DASD system performance evaluation. He received his B.S. degree in mathematics from Stanford University and his master's degree in electrical engineering and computer science from the University of California at Berkeley. Mr. McNutt joined IBM in 1983 and has specialized in DASD performance and workload characterization since that time. His 1990 paper "DASD Configuration Planning: Three Simple Checks" received the award for best management paper at that year's conference of the Computer Measurement Group (CMG). In 1987, as described in subsequent papers, Mr. McNutt originated the "multiple workload" approach to planning for storage-control cache memory (planning based upon an objective for cache residency time), which is now in widespread use.