MYSTERIES OF BUFFER OPTIMIZATION SOLVED!

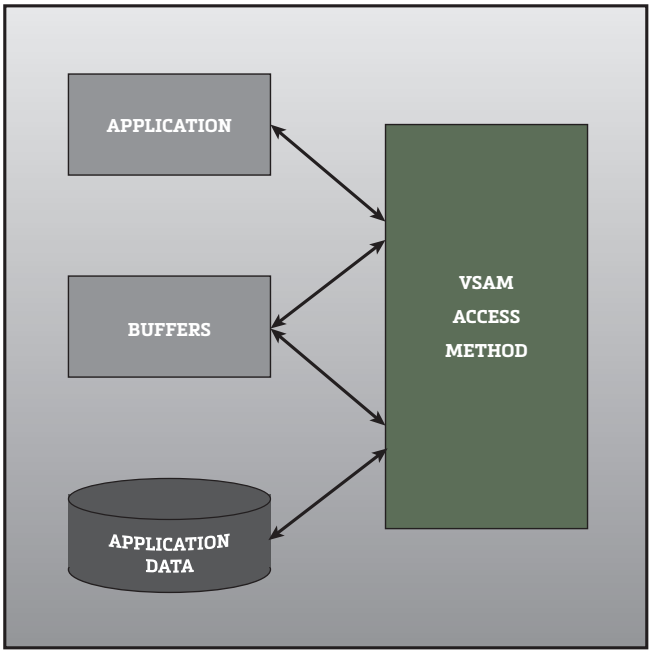
By Bill Hitefield, Dino-Software

In some respects, IBM's access methods (especially VSAM) are like software black boxes – you pass a request to them, they go off into the Enchanted Forest, and sometime later, data is either retrieved or written by your application. As application developers, we have some modicum of control over when, why, and how often we make requests to the access method. It is that second part of the process, the "sometime later," that we view as a part of application performance that we seemingly have no means of influencing.

For most applications, if the "sometime later" variable were lessened, the aggregate overhead experienced by an application would decrease.

The access method and buffers

When the access method processes the data set on behalf of the application, it builds a buffering environment. Think of buffers as "temporary housing" for data records. The access method uses this temporary housing as a staging area for the data until it is needed for the program.



As application owners, we know how our application accesses the data (sequentially, randomly, or some combination). If we combine that knowledge with knowledge of how the access method avails itself of the buffers in each of those access types, then we can define a unique buffering environment for each data set accessed by our application, and thus ensure optimal performance by the access method.

Many people already do this. There are multiple sources that make that correlation (for example, IBM's "VSAM Demystified," Redbooks SG24-6105). Those sources allow them to determine the needed buffering environment and then make the needed alterations to their system (JCL, …) to define that specific environment.

Improvements in application execution time can be quite dramatic. In some instances, over 90% of the EXCP SVCs issued against a data set are not just optimized, they are eliminated. This can result in tremendous savings.

This sounds like something that everybody should be doing, doesn't it? If so, then why don't they?

For an IT site, the task does not end with that one set of definitions. Applications change over

time, data sets change over time, and the access pattern that the application uses changes over time. In their due diligence, the site then must periodically monitor those definitions to ensure that they are current and that they in fact provide the needed relief.

The manpower and time required to initially perform this task and then monitor the results quickly becomes something that is not insignificant. This causes most sites to lose heart in this ongoing contest with the access method. A good buffer optimizer such as VELOCI-Raptor can virtually eliminate this concern.

The buffer optimizer's place in the process

This is where a buffer optimization product, such as VELOCI-Raptor (also known as VR), performs its magic. When an application opens a data set, VR will combine its knowledge of access method behavior, the access declarations made by the application, along with the current status of the data set (number of records, CI-size, and so forth), and will determine the best buffering environment for that particular access to the data set.

Since all of this information is used each time the data set is opened, there is no need to have an ongoing task to monitor the performance definitions that were (without the buffer optimization product) entered manually.

The buffering product does what a performance specialist could have done and does it automatically each time a data set is opened, for every application, day or night, over and over, even on weekends. The manpower constraint facing most sites is now no longer a concern.

Going deeper - the buffer optimizer and IBM's buffering schemes

IBM allows for the use of different buffering technologies. The default is NSR – Non-Shared Resources. While this buffering scheme excels in sequential access to a data set, it seems to merely tolerate random access. IBM also offers LSR – Local Shared Resources. This buffering scheme is pretty much the opposite of NSR in how it responds to applications. It excels in random access to a data set, and it tolerates sequential access.

For a performance specialist to implement LSR, the approach is not quite as straightforward as the process described earlier. The implementation of LSR requires more internal knowledge of the application. LSR has certain expectations and requirements that must be met by the requestor of its services. For all of its beauty in easily processing random requests, LSR is fairly stringent in its implementation.

In their implementation of LSR, this means that the performance specialist must be diligent not only in selecting the proper applications but also in making the needed changes to implement the use of LSR for an application.

Buffering products like VELOCI-Raptor are designed to do just that. When the data set is opened, they will determine whether the optimal scheme is NSR or LSR. If LSR is selected, VELOCI-Raptor will provide the necessary interface to ensure that the application abides by the restrictions set by LSR itself.

VELOCI-Raptor then extends this support by determining whether or not the application abides by its own declared intentions. For example, the application may declare "ACCESS IS RANDOM" when the data set is opened, yet it may process the data set sequentially. If so, the buffering

strategy defined either by the performance specialist or by the buffering product is no longer optimal. VELOCI-Raptor monitors that application's access and can determine when a different buffering scheme may have been more beneficial – and then implement that buffering scheme.

A final consideration – communicating with the buffer optimizer

For a site to make effective use of any buffer optimization solution, it must be able to communicate its wishes to the chosen solution. What candidates are to be selected? Once selected, what is to be done with them?

Communication with any buffering solution can be broken down into several levels of user involvement as follows:

1. Low-level

Some solutions require guidance at a very low level, resulting in a high level of input from the user. Not only must the user identify the candidates for optimization, they must also define (and monitor) the type of optimization that is to be performed. At this level, the buffering solution does not make that determination.

An example of this would be the use of JCL parameters to implement a rudimentary buffering solution. The user must not only select which candidates are to be processed, they must also implement that selection in each place that candidate is referenced.

While this method may get the job done, it requires a great deal of effort and sophistication on the part of the user.

2. Mid-level

Solutions at this level require the same sort of definitions (i.e., identification of the candidates and a selection of the optimization type) but require them to be issued only once. The buffering product will then apply the requested actions each time the candidate data set is accessed.

Some buffering solutions may allow you to define buffering options that can be associated with various criteria. You can specify a specific option that is to be used for a certain data set, or you can specify a solution that is to globally apply to a selected storage class.

While this is an improvement over the previous level, the user still must make a determination as to which buffering solution is to be implemented.

3. High-level

At this highest level, the best of all features is uniquely embodied within VELOCI-Raptor. VR exclusively provides:

- The user's only involvement is simply the identification of candidates.
- The product is able, in and of itself, to determine the optimal buffering solution based upon real-time criteria 24/7/365.

At this level, users only need to concern themselves with the "what" of their activities, not the "how." This removes a tremendous workload from the user.

4. Conclusion

In literature lore, most mysteries end with "the butler did it." In the mystery of buffer

optimization, the answer is "the buffering product did it." A good buffering product does what a performance specialist can do, but does it automatically – and does it all the time.

VELOCI-Raptor was designed with the premise that an application is not static for long. Something will change, and that something will impact performance. VR is uniquely prepared for that, will detect it, and will make the needed adjustments to the optimal buffering scheme.

VELOCI-Raptor has a mantra – "The fastest I/O is the one you do not have to issue." Its goal is to provide a buffering scheme such that a majority of application I/O requests can be satisfied via "temporary housing" – the buffers.

When considering a buffering product, think about the following:

- The product should have a realistic and intelligent approach to buffer optimization. It should avail itself of all possible solutions and not limit itself to any particular one.
- The product should automatically respond to dynamics in your operating environment. You should not have to monitor its selections.
- All the product should require of you is the identification of candidates. It should be sophisticated enough to determine the optimal solution on its own.

I hope that by now the mystery of the "buffer optimizer" has been removed and we now have a clear picture of the need for something that will help us optimize the buffering environment in which VSAM works with our applications. Our understanding of the role of a buffer optimizer now helps us determine whether or not one is right for us. We also have some new insight as to how to differentiate the good optimizers from the ordinary.

Is a buffer optimizer right for you? I would offer that there is a high probability that you should consider one. If so, I would suggest that you give VELOCI-Raptor a good look. You will be happy you did. No mystery involved.