DB2 Version 3: A Closer Look (Part 2)
By Craig S. Mullins

As you read this article, DB2 Version 3 should be generally available for customers to implement. The last issue of PLATINUM SYSJOURNAL explored two features of this new version—data compression and extended bufferpool support. In this issue we will continue this exploration with a closer look at two more DB2 V3 features—hiperpools and query I/O parallelism.

This article will provide an analytical examination of these features, instead of the descriptive information provided by the IBM manuals. Some level of understanding of these features is assumed.

Hiperpools

By providing DB2 V3 with 60 bufferpools that can be dynamically modified, IBM has contributed greatly to the options available for tuning DB2. But the expanded number of bufferpools and their greater usability is only half of the story when implementing DB2 Version 3 buffering. IBM has also provided a new buffering feature called Hiperpools.

Hiperpools can be considered extensions to the "regular" bufferpools, which are now referred to as virtual pools. Working in conjunction with the virtual pools, hiperpools provide a second level of data caching. In DB2 V3, when "old" information is targeted to be discarded from (*or, moved out of*) the bufferpool, it will be moved to the hiperpool instead (if a hiperpool has been defined for that bufferpool).

Only clean pages will be moved to the hiperpool, though. Clean pages are those where the data that was modified has already been written back to DASD. No data with "pending" modifications will ever reside in a hiperpool.

Each of the 60 virtual pools can optionally have a hiperpool associated with it. There is a one to one relationship between virtual pools and hiperpools. A virtual pool can have one and only one hiperpool associated with it, but it may have none. A hiperpool must have one and only one virtual pool associated with it.

Hiperpools are page addressable, so before data can be accessed by an application it must be moved from the hiperpool to the virtual pool (which is byte addressable). Hiperpools are backed by expanded storage only, whereas virtual pools are backed by central storage, expanded storage, and possibly DASD if paging occurs. Keeping this information in mind, consider using hiperpools instead of specifying extremely large virtual pools without a hiperpool.

When you specify a virtual pool without a hiperpool, you are letting MVS allocate the bufferpool storage required in both central and expanded memory. If possible, specify a virtual pool that will completely fit in central storage and a hiperpool associated with that virtual pool. The DB2 buffer manager will handle the movement from expanded to central storage and should be more efficient than simply implementing a single large virtual pool. Of course, you will need to monitor the system to ensure that the virtual pool is utilizing central storage in an optimally efficient manner.

You may also want to consider tuning your hiperpools for specific types of processing. For example, if you know that the majority of your sequential prefetch requests will never be accessed again, then you may want to tune your hiperpools to avoid sequential data. You can accomplish this by setting the HPSEQT parameter to zero (0) using the ALTER BUFFERPOOL command. This will ensure that only randomly accessed data will be moved to the hiperpool.

To utilize hiperpools you must be using MVS/ESA 4.3 on an ES/9000 Model 511 or 711 series processor with ADMF (Asynchronous Data Mover Facility). The total of all hiperpools defined can not exceed 8 gigabytes.

I/O Parallelism

Another new feature employed by DB2 Version 3 is the ability to execute queries in parallel using a technique called query I/O parallelism. Generally, I/O parallelism enables DB2 to execute parallel read engines against partitioned tablespaces. There are instances where I/O parallelism can be to access non-partitioned tablespaces, but those situations are outside the scope of this article.

At optimization time, DB2 can be directed to consider parallelism by:
- specifying DEGREE(ANY) at BIND time for packages and plan
- issuing the SET CURRENT DEGREE = "ANY" statement inside the program for dynamic SQL

Four new PLAN_TABLE columns will indicate whether I/O parallelism is in effect for a particular query. These columns are shown in Figure 1.

Figure 1. New PLAN_TABLE Columns

Column Name	Column Type	Column Definition
ACCESS_DEGREE	SMALLINT	Indicates the degree of parallelism for this portion of the query. Degree of parallelism is synonymous with the number of parallel I/O streams that will be utilized.
ACCESS_PGROUP_ID	SMALLINT	Identifies the grouping of rows in the PLAN_TABLE that are associated with a particular I/O stream.
JOIN_DEGREE	SMALLINT	Indicates the join degree of parallelism when joining composite table with new table.
JOIN_PGROUP_ID	SMALLINT	Identifies the grouping of rows in the PLAN_TABLE that are associated with the parallel join.

I/O parallelism can significantly enhance the performance of queries against partitioned tablespaces. By executing in parallel, elapsed time will usually decrease, even if CPU time does not. This will result in an overall perceived performance gain because the same amount of work will be accomplished in less clock time.

The types of queries that will benefit most from I/O parallelism are those that:
- access large amount of data, but return only a few rows
- use column functions (AVG, COUNT, MIN, MAX, SUM)
- access long rows

To prepare for query I/O parallelism follow these suggestions:

- For partitioned tablespaces: move each partition of the same tablespace to a separate DASD volumes. Failure to do so will impact performance because I/O parallelism will cause concurrent access to separate partitions and, if those partitions co-exist on the same device, increased head movement will usually negatively impact performance.
- For non-partitioned tablespaces: consider partitioning tablespaces that are accessed in a read only manner by long-running batch programs. Of course, very small tablespaces are rarely viable candidates for partitioning, even under DB2 Version 3.

A final caution: increased parallel activity could cause DB2 buffers to be more active. Consider a single query that uses sequential prefetch to access an entire tablespace with 4 partitions under DB2 V2.3. After converting to DB2 V3 and re-binding specifying DEGREE(ANY), DB2 decides to use I/O parallelism with a degree of 4 (i.e. 4 concurrent I/O streams). This will cause 4 concurrent MVS I/O subtask SRBs to be initiated to satisfy the request. Instead of 1 task accessing the bufferpool (V2.3), there will be 4 (V3). As the number of concurrent parallel queries increases, bufferpool thrashing may occur. Of course, if resources are unavailable, DB2 can choose not to use I/O parallelism, even if it is specified in the PLAN_TABLE.

Synopsis

This article has analyzed only two of the numerous features that will be available with DB2 Version 3. Future SYSJOURNAL articles will take a look at some of the other new features, but until then remember this: as more administrative and development options become available with DB2, your organization will need to develop an effective strategy for the optimal implementation of new techniques. Nothing comes without a cost. The appropriate mechanism for examining new features is a cost/benefit analysis.

New features usually provide a benefit in terms of:
- better performance
- better resource utilization
- easier administration
- easier use

However, costs for new features typically include one or more of the following:
- additional consumption of valuable system resources (CPU, DASD, I/O)
- requirements for new hardware
- cost of implementing new standards and procedures

Analyze each new feature and determine if the benefit outweighs the cost. Generally, you will find that the benefits of new DB2 features will outweigh their costs if proper management and focus is maintained.