IBM Database 2 performance: Design, implementation, and tuning

by J. M. Cheng
C. R. Loosley
A. Shibamiya
P. S. Worthington

The larger and more complex a relational data base, the more efficient the data base management system must be to maintain an acceptable level of performance. The design and implementation of IBM Database 2 (DB2) have been aimed toward this objective. Techniques for achieving this key objective in DB2 are the subject of this paper. Presented are performance-related strategies in query processing and performance-related design tradeoffs. Data base and application design options and their resolution for optimum performance are also discussed. Also presented are techniques to maintain performance by application monitoring and tuning and DB2 system tuning.

IBM Database 2 (DB2) is a relational data base management system[1,2] that supports both repetitive transactions and ad hoc queries against large data bases. In such a system, achieving acceptable performance for the different types of work is a key design objective. Performance objectives for DB2 centered around maximizing the level of transaction concurrency while minimizing

- Number of instructions executed by the CPU.
- Number of Input/Output (I/O) operations.
- Time to perform the I/O operations.
- Failures due to resource overcommitment.

This paper describes how DB2 achieves these objectives and how a DB2 user can monitor and tune the performance of DB2.

The first part of this paper describes performance considerations in the design and implementation of Structured Query Language (SQL) query optimization, the use of compilation instead of interpretation in the processing of SQL, and the performance gain in dividing the processing into separate compilation and execution phases. Performance-related tradeoffs made in the design of DB2 components are then discussed. Specific topics include the data base buffer pool algorithms, the use of virtual storage in multiple private address spaces, and the DB2 sort implementation. Next, the options provided to a DB2 user to optimize the performance of a data base and application are described. On the data base side, efficient uses of DB2 tables and indexes are discussed. On the application side, the uses of dynamic and static SQL, and various data base locking options are reviewed. DB2 utility options are also discussed. The final part of this paper is devoted to the monitoring and tuning of DB2. Sections cover application monitoring, use of information from the DB2 catalog tables, DB2 storage management options, and the DB2 statistics facility. Sample reports showing DB2 accounting and statistics data are presented, and some key data items are reviewed.

Performance-related strategies in query processing

Several strategies are employed to optimize the performance of SQL statements. These strategies are

described under the headings *SQL query optimization, Automatic access path selection,* and *Bind and SQL compilation.*

SQL query optimization. SQL is a language that gives a user the power to concentrate on specifying what is to be retrieved, rather than how the information is to be retrieved.[3] Query optimization is very important for a successful relational data base management system. Many optimization techniques have been published[4-7] or implemented in existing systems.[8-12] In this section, some of the heuristic strategies employed by DB2 for query processing are discussed.

Query optimization is very important for a successful relational data base management system.

A query may contain an arbitrary number of predicates (search criteria) joined by AND or OR conjunctors. The result of the evaluation of the predicates is either TRUE (the row should be returned) or FALSE (disregard the row). The SQL predicate is very similar to the IF condition of compiler languages. Therefore, the technique used to optimize the IF condition is also applied in the optimization of SQL predicates. For example, consider the following evaluation of the criteria:

```
WHERE PREDICATE1 AND PREDICATE2.
```

if PREDICATE1 yields FALSE, the evaluation of PREDICATE2 is skipped. SQL allows the negation of the search criteria, as in the following example:

```
WHERE NOT (PREDICATE1 AND PREDICATE2)
```

These search criteria cannot be materialized until the NOT operand is evaluated. Hence, DB2 decomposes this into the following criteria:

```
WHERE (NOT PREDICATE1) OR (NOT PREDICATE2)
```

if PREDICATE1 is FALSE, the result of (NOT PREDICATE1) is TRUE and the evaluation of PREDICATE2 is skipped.

DB2 optimizes the performance of a noncorrelated *subquery* (i.e., a subquery that does not reference any column of a table in another query block) by evaluating the subquery once and saving the result of the subquery for subsequent evaluation in the main query. This is illustrated by the following example:

```
SELECT NAME, ADDRESS FROM MAILLIST WHERE NAME
       IN (SELECT CUSTNAME FROM CUSTOMERS
       WHERE AMOUNTDUE > 1000)
```

Here, the names of customers with more than $1000 due are retrieved once. The names from MAILLIST are then checked against these customer names.

Among the many ways of joining two tables, the nested-loop method and the merge-join method have been shown to be near optimal.[4] Both of these join methods have been implemented in DB2. In the nested-loop method, a row is retrieved from one table, and then all the rows in the other table with matching values in the join column are retrieved. In the merge-join method, both tables are first sorted according to the join column. Thereafter, the merge-join method is very similar to that of the nested-loop method. However, since both tables are sorted, the scan of the second table for matching values in the join column does not have to start from the top of the table for every join value. Furthermore, the second table is scanned more efficiently because it has been sorted on the join column, and it is being scanned in join-column order.

Automatic access path selection. DB2 supports two methods for retrieving rows from a table: table space scan and index scan.

A table space scan looks through the table space sequentially, whereas an index scan looks through the table space in index value order. An index can be of the clustering or nonclustering type. When a table has a clustering index, DB2 attempts to allocate storage for the rows of the table such that their physical ordering is the same as their corresponding index values.

An index may be scanned for a particular key value or for a range of key values if the index appears in a simple predicate, such as COLUMN1 < 20. An index

may also be used without any key being supplied, in which case the entire index and table space are scanned. Unlike Data Language/I (DL/I), DB2 does not read any page from the table space if it can retrieve the information from the index. For example, the following statement is used to count the number of rows in a table:

```
SELECT COUNT(*) FROM SOMETABLE
```

DB2 scans the index to count the number of index entries without reading any data pages in the table space. This method is efficient for this type of query because the index is usually much smaller than the table space.

A table space scan may be better than an index scan in some situations and worse in others. For example, a table space scan on a table space with a single table is more efficient than an index scan with no key value. On the other hand, a clustering index scan on a small table in a large table space which contains several tables is more efficient than a table space scan.

To facilitate the process of path selection, DB2 provides a utility called RUNSTATS to collect statistics on tables, table spaces, and indexes. The following statistical information is included:

- Number of rows in a table.
- Number of pages occupied by a table.
- Number of pages in a table space.
- Number of distinct key values in an index.
- Second highest key value in an index.
- Second lowest key value in an index.
- Number of index levels.
- Number of leaf pages in an index.

When making the selection of an access path, different ways of accessing each table are examined, and a cost for each access path is derived from the accumulated statistics. The cost of an access path consists of a weighted sum of the calculated I/O and processor cost. The I/O cost is a function of the estimated number of pages read for a query, and the processor cost is a function of the estimated number of calls to the DB2 component that retrieves and filters the data. The least expensive access path is chosen to retrieve rows of a single table. Finally, if the rows must be sorted, the cost of using an index to satisfy the sort sequence is compared to the least cost of retrieving the table plus the cost of sorting the rows. The less expensive method is chosen.

For a query involving joins, the order of joins significantly affects the performance of the query. Normally, DB2 examines all permutations of joining all the tables. The permutation of joining two unrelated tables is not examined unless there are no other related tables to join. (Two tables are unrelated if there are no common join columns, in which case a cartesian product of the two tables is requested.) Both the nested-loop method and the merge-join method are considered for the joining of two tables. The permutation that gives the least total cost in joining all the tables is chosen.

Bind and SQL compilation. DB2 provides the BIND and RUN subcommands to separate the processing of static SQL statements into two phases. The BIND process prepares SQL statements to be executed at RUN time and consists of the following steps:

1. Reading data sets that contain the SQL statements extracted from the application program during precompilation.
2. Saving the SQL statements in the DB2 catalog.
3. Processing each SQL statement.

An application may be run multiple times if it has been bound once.

For an SQL data manipulative statement (e.g., SELECT, INSERT), information is retrieved from the DB2 catalog for semantic and capability validation and for access path selection. An application plan describing the execution sequence of the SQL statement is then generated and saved.

For an SQL data definitional (e.g., CREATE, DROP) or data control statement (e.g., GRANT, REVOKE), information is retrieved from the DB2 catalog for synonym resolution and for the assignment of defaults, such as the default table space name. An internal form of the statement is produced and saved for execution at RUN time.

An application may be run multiple times if it has been bound once. Furthermore, the BINDing of an

application may be scheduled for off hours to reduce the demand for system resources during peak hours. An application plan may become invalid if, after the BIND process, the authorization to access an object is revoked or such objects as synonyms, tables, or indexes used by the plan are dropped. The detection of an invalid application plan at RUN time causes an automatic REBIND of the application program.

Performance studies have shown that query compilation is more efficient than interpretation.

For a dynamic SQL statement, the bind process consists of semantic and capability validation and access path selection only. This is referred to as dynamic bind.

Unlike DL/I, SQL is a compiled language. Therefore, at BIND time, an SQL statement is parsed and optimized. Code is also generated for SQL data manipulative statements. Even ad hoc queries are compiled before they are run. For instance, a query returning one hundred rows of a table does not require an interpretation of the search criteria one hundred times. Performance studies[13,14] have shown that query compilation is more efficient than interpretation.

Performance-related internal design tradeoffs

Numerous performance-related tradeoffs were made during the design and development phases of DB2, with the goal of providing the best overall product for the intended application environments. Included among the tradeoffs between performance factors were those between CPU and I/O, between virtual storage and CPU, and tradeoffs among performance and reliability, availability, and serviceability.

The allocation of system resources to DB2 requests is controlled by DB2 Resource Managers, several of which were designed with performance goals in mind. Also, one significant performance-related tradeoff concerned the use of Direct Access Storage Devices (DASD). In this section, the following aspects of DB2 resource management are discussed:

- DB2 use of MVS cross-memory services.
- The process of allocating resources to DB2 requests.
- Environmental Descriptor Manager, which keeps frequently used data base and plan descriptors in storage.
- Buffer Manager, which keeps frequently used data base pages in storage.
- Storage Manager, which allocates and frees virtual storage.
- DB2 sort, which sorts records retrieved from the data base.

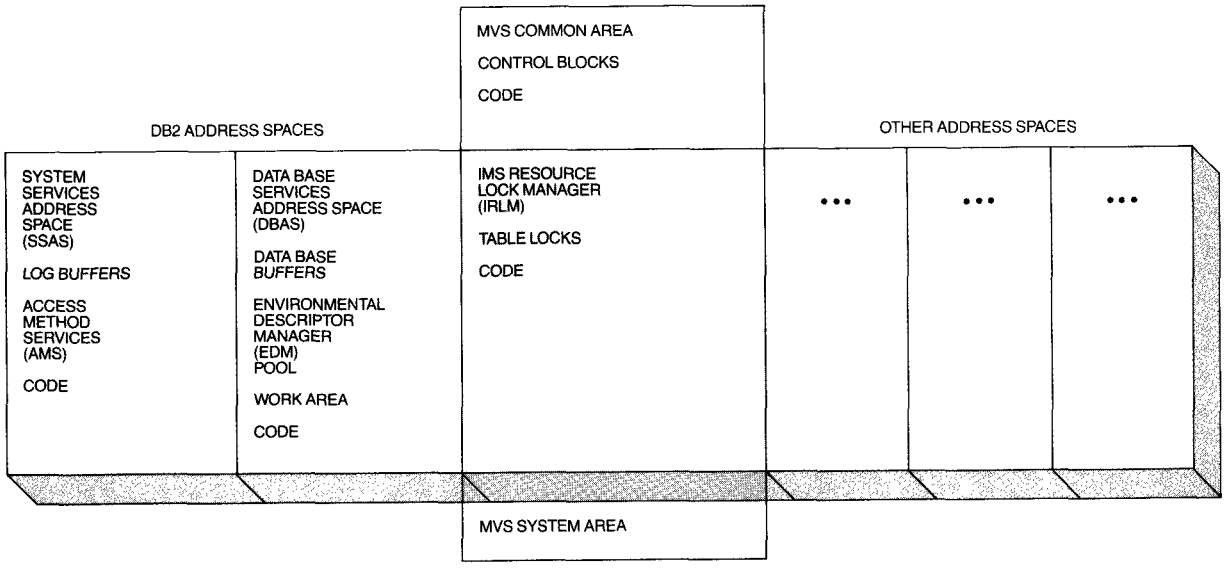
The performance-related tradeoff concerns the amount and placement of free space in DB2 table spaces. The effect of these design decisions on DB2 processor cost, I/O operations, and concurrency is discussed in the following sections. In addition, the DB2 use of virtual storage is described; greater concurrency requires increased use of virtual storage.

Use of virtual storage in multiple private address spaces

MVS cross-memory architecture supports greater virtual storage by distributing code and data into multiple private address spaces through the use of cross-memory operations. This is done instead of keeping the codes and data in one private address space or in the MVS common area. Although the hardware assist of cross-memory operations substantially reduces the processor overhead for these operations, there still exists residual overhead resulting from additional instructions executed in cross-memory mode and degradation in processor speed resulting from cross-memory operations. Therefore, it is important to minimize cross-memory overhead by placing frequently referenced code and data in the MVS common area. The distribution of code and data into many address spaces would increase cross-memory usage, with resulting increased overhead.

Figure 1 illustrates the way by which DB2 takes advantage of this MVS feature by operating out of the following two address spaces: (1) Data Base Services Address Space (DBAS), which contains code and data related to data base functions; and (2) System Services Address Space (SSAS), which contains code and data related to such service functions as logging. Some common code and control blocks are placed in the MVS common area. In most installations, the

Figure 1 DB2 use of MVS virtual storage



IMS Resource Lock Manager (IRLM) locks on DB2 data are placed in private address spaces of their own, so that outstanding DB2 locks do not take up storage in the MVS common area. A typical SQL FETCH request operates in three address spaces (application, DBAS, and IRLM), and a typical UPDATE request operates in four address spaces (application, DBAS, SSAS, and IRLM).

Resource allocation and deallocation. Resources must be allocated to a transaction before any SQL statements can be issued. The allocation process consists of creating a DB2 internal task structure, known as an *agent*, to represent the transaction, acquiring storage and initializing control blocks for the agent, reading from DASD the application plan the agent will use, opening data sets this agent might access, and obtaining data base locks. At transaction termination, these operations are reversed—the data base locks are released, open data sets are closed (unless otherwise indicated when the table space was defined), and the storage for the control blocks is freed. Information that allows DB2 to determine which data base locks to obtain and which data sets to allocate and open is stored in the application plan.

Work done at allocation time need not be repeated for every SQL statement issued by the agent. Although this normally represents a savings in processor cost and I/Os, it is not always true. For example, even though opening and closing data sets at allocation and deallocation time means that each data set is opened only once per agent, many SQL statements may use that data set. There is no savings, however, if each SQL statement uses a different data set and each SQL statement is executed only once. If some of those SQL statements are not executed at all (or, for example, they are used only in error situations), data sets are being opened and closed, but never used.

Obtaining data base locks at allocation time may also reduce concurrency. This is because data base locks acquired at allocation time are not freed until deallocation time, even if the application COMMITS and releases other data base locks. However, there is less chance of having to back out the transaction because of a deadlock with another transaction. If locks cannot be obtained, the transaction waits at the allocation stage until the locks are freed.

Separate copies of application plans (which are not re-entrant) are kept in virtual storage for each DB2 agent executing that plan. Keeping a copy of an application plan in virtual storage means that DB2 does not have to reread the plan from DASD and relocate it in virtual storage for each SQL statement executed. That also means, however, that virtual storage must be allocated to the plan even when SQL statements are not being executed. Keeping the ap-

plication plan in virtual storage is a tradeoff in which virtual storage is used to minimize processor cost and I/Os.

This use of virtual storage was an important design consideration for the Query Management Facility (QMF), because QMF users typically spend a great deal of time scrolling through output or thinking about the next query to run. They spend relatively little time actually executing SQL statements. For this reason, each time the QMF user runs a query, a copy of the QMF application plan is allocated for the QMF user. As soon as all the data in the QMF result have been retrieved, this application plan is deallocated. Since QMF initially retrieves more rows of data than can be displayed on a single screen, QMF can often release all DB2 resources before the user starts to scroll through the screens of the query output. Thus, during the time the user is composing queries and specifying QMF forms, DB2 resource usage can often be avoided.

Two DB2 components have been designed specifically to minimize processor cost and I/Os through the use of virtual storage. These are the Environmental Descriptor Manager and the Buffer Manager.

Environmental Descriptor Manager. During BIND for an application program, a DB2 plan is created and stored in an internal DB2 format called the Skeleton Cursor Table (SKCT) template in a DB2 directory table space. During creation of a data base, a data base descriptor is stored in another DB2 directory table space. The Environmental Descriptor Manager (EDM) is responsible for access to these plans and data base descriptors. A block of virtual storage, called the Environmental Descriptor Manager Pool (EDM pool), is reserved when DB2 is started. This virtual storage is used to store frequently used SKCT templates and data base descriptors. The EDM pool is also used to hold a Cursor Table (CT) needed by each agent as it executes.

The installation specifies the size of the EDM pool, which must be at least large enough to hold the CTs and data base descriptors needed by each concurrent agent. If the EDM pool is larger than this minimum size, it is used to hold SKCT templates and other data base descriptors. When an agent is allocated, the pool is searched to see whether the SKCT template and data base descriptors are already in the pool. If so, they do not need to be read from DASD. In this case, storage is allocated for a CT for this agent, and the SKCT template is copied to the CT. If the SKCT template or data base descriptors are not in the pool, it is necessary to allocate storage within the pool to hold the SKCT template or data base descriptors and read them from the DB2 directory.

Often, some other SKCT templates or data base descriptors have to be deleted to make room for this new SKCT template or data base descriptor. This is done first by randomly deleting data base descriptors that are not being used until enough storage is available for the storage request. If there is still not enough storage available, SKCT templates are deleted randomly. If no storage is available after all data base descriptors and SKCT templates have been deleted, agent allocation fails, and the appropriate subsystem (IMS, CICS, or the TSO user) is notified.

The EDM pool was originally designed to provide fast, inexpensive access to SKCT templates, and data base descriptors. Therefore, more expensive storage replacement algorithms, such as least recently used (LRU), have not been implemented. Neither have more expensive storage management algorithms (such as subdividing SKCT templates and CTs into smaller units and keeping only the active part of this information in storage) been implemented.

In MVS/370 systems, in which the EDM pool may be smaller than the optimal size, these design tradeoffs may have a significant influence on the performance of the DB2 system. If the EDM pool is so small that SKCT templates have to be read frequently from the DB2 directory, there may be an I/O bottleneck on the DASD that contains the SKCT data set. Such a bottleneck may limit the amount of work DB2 can process per hour. In addition, the SKCT templates and data base descriptors are read into the data base buffer pool in 4096-byte blocks, thereby reducing the amount of the buffer pool available for data base pages used by transactions and queries. This method differs from the IMS design, in which descriptors are read in a single block directly into the IMS descriptor pools.

Buffer Manager. In DB2, as in most data base management systems, virtual storage is reserved to hold copies of data base pages. This virtual storage, which is called the *buffer pool*, contains the following pages:

- *In-use pages*, which are currently being read or updated by transactions.
- *To-be-written pages*, which contain updates from a transaction but which have not been written to DASD. The updates may not yet have been COMMITted by the transaction.

- *Look-aside-buffering pages*, which are neither in use nor to be written. They are kept in virtual storage only because there is a possibility that they will be needed again.

DB2 allows multiple buffer pools. Three buffer pools hold 4K-byte pages (4096 bytes of data), and one buffer pool holds 32K-byte pages (32 768 bytes of data).

Assigning a separate buffer pool to 32K-byte-page table spaces allows DB2 to read and write 32K-byte blocks of data with a single I/O. These large pages are used for table spaces that contain records longer than 4K bytes.

Updated pages are normally kept in the buffer pool in preference to nonupdated pages.

Installations are free to assign the three 4K-byte buffer pools to any table spaces they wish. The only restriction is that the first buffer pool (BP0) is used for the DB2 system table spaces (e.g., the DB2 catalog, directory, and the table space DB2 uses for temporary storage).

The Buffer Manager algorithms for reassigning and writing a buffer are complex. The basic algorithm is Least Recently Used (LRU) replacement, in which the page that has been least recently used by a query or transaction is chosen to be replaced in the buffer pool. This process is known as *stealing a buffer.* The simple stealing-a-buffer algorithm has been modified to take into account other factors, such as the cost of stealing a buffer containing an updated page.

One of the factors considered in the design of the Buffer Manager algorithms was the design of the internal I/O driver used by DB2 for I/O services. The internal I/O driver writes up to thirty-two pages of a data set with a single Start I/O (SIO) instruction. Although this is a highly efficient way to schedule I/Os, it also means increased time to complete the I/O, because multiple pages must be written. Thus DB2 must make a tradeoff between frequently scheduling I/Os for a few pages and infrequently scheduling I/Os for many pages. Although DB2 agents do not usually have to wait for the completion of write I/Os, agents do have to wait for the completion of read I/Os. If an agent issues a read to a device that has just been given 32 pages to write, the agent must wait for all 32 pages to be written.

Thus the response time for a transaction can depend heavily on the DB2 decision of how many pages to write at one time.

A related consideration is the value and cost of keeping updated pages in virtual storage. Unlike pages that have been read but not updated by a transaction, if the buffer containing an updated page is stolen and used for another page, it is first necessary to write the updated page back to DASD. Therefore, if two pages are equally likely to be reused, it is better to keep the updated page in its buffer and steal the buffer containing the page that has not been updated. If, however, many pages from one table space are updated before being written, the I/O takes a long time when they are eventually written.

DB2 schedules an updated page to be written whenever too many buffers in the buffer pool are in use or waiting to be written or when too many buffers have been updated for the dataset containing that page. Thus updated pages are normally kept in the buffer pool in preference to nonupdated pages, when there are sufficient stealable buffers.

A data base page is considered to be in use if another DB2 component, called the Data Manager, has acquired the page and has not released it. The Data Manager can acquire and release the page every time it accesses a record on that page. The Data Manager, however, often accesses several records on a page, and the acquiring and releasing of the page for every access results in greater processor cost for each access. Therefore, the Data Manager normally acquires and releases each page only once while it reads all the records on the page. If, however, the time between record accesses is very long (as it may be in very complicated SQL joins), this interim may result in the page being in use for a very long period of time. In fact, other pages may be replaced and reread by the Buffer Manager while this page remains unaccessed but in use.

To alleviate this problem, when the number of stealable buffers becomes very small, the Buffer Manager warns the Data Manager that buffer space is critical. At this time, the Data Manager starts acquiring and releasing each page every time it accesses a record. The Buffer Manager also notifies the Data Manager when sufficient stealable buffers are available again. In this way, the normal LRU algorithms are modified to take advantage of the additional information available to the Data Manager.

The normal LRU algorithms are modified to use information available to the Data Manager.

Normally, when the Buffer Manager decides that it is time to start writing updated pages, DB2 service agents—known as *asynchronous write engines*—are started. Each write engine is responsible for calling the internal I/O driver to initiate writes to a single DB2 data set. When the writing completes, the pages that were written are no longer marked as "to-be-written," and their buffers are available to be stolen. In this design, DB2 agents do not normally have to wait for writing to complete, because the asynchronous write engine takes over that responsibility.

In a small buffer pool, however, it is possible that almost all pages are in use or to be written by the write engines. In this case, there are very few buffers that can be stolen. In fact, if there were many concurrent agents, it is possible that suddenly there would be no buffers available to steal when an agent requested a page. Therefore, when there are very few stealable buffers, DB2 agents start writing their updated pages synchronously. That is, when a transaction updates a page, it also waits for that page to be updated on DASD. The Buffer Manager algorithms have been designed to avoid this situation whenever possible, because this can significantly degrade response time.

Storage Manager. A Storage Manager provides two performance-critical services to other DB2 components: (1) it avoids expensive MVS GETMAINS and FREEMAINS to acquire storage from MVS, and (2) it provides a mechanism for separating storage used for different purposes into different DB2 pools. The Storage Manager also monitors storage usage by DB2 and MVS and ensures that adequate storage is available in the DB2 address spaces for MVS services. The creator of a pool can specify such characteristics of the pool as the following:

- Whether the blocks of storage in the pool are of fixed or variable length. This prevents storage fragmentation in fixed-length pools.
- Whether the storage pool is to be used by multiple DB2 agents. If the pool is not used in this way, it can be released when the agent is deallocated.
- Whether a best-fit algorithm is to be used to acquire storage in a pool that contains variable-length blocks. If a best-fit algorithm is used, the smallest (i.e., best) free block that is large enough to satisfy the request is chosen. This results in greater processor cost than the first-fit algorithm in which any (the first) free block that is sufficient to satisfy the request is chosen. In DB2, all pools except the EDM pool are first-fit pools. The best-fit algorithm in the EDM pool reduces fragmentation resulting from a wide range of storage request sizes.
- Whether a pool theshold value is to be used. In threshold pools storage blocks are divided into two groups. Storage requests below a certain threshold size are satisfied from one group of storage blocks, while requests above the threshold size are satisfied from the other. This algorithm is used in the Relational Data System (RDS) pool, in which requests tend to be either very small or quite large.

DB2 does not automatically return free storage within its pools to MVS because it is assumed that this free storage will soon be reused within that pool. Keeping it avoids the cost of an MVS FREEMAIN and subsequent GETMAIN operations. Without some mechanism to manage the use of storage in the MVS/370 environment, functions requiring storage might fail even though DB2 has sufficient free storage available but in the wrong pool. Therefore, if DB2 is executing under MVS/370, it monitors its own acquisition of virtual storage as well as non-DB2 acquisitions in the Data Base Services Address Space.

The Storage Manager is also responsible for ensuring that a sufficient amount of virtual storage can be acquired by MVS for certain non-DB2 requests, such as requests for space for control blocks for open data sets. Finally, the Storage Manager reserves space for

DB2 must-complete functions, which, if they fail (for example, because virtual storage is not available), cause DB2 to terminate. In particular, the Storage Manager reserves storage for the following:

- *MVS services.*
- *Open data sets.* The amount of storage to reserve is determined from the installation parameter that gives the maximum number of concurrently open data sets. Of course, as DB2 opens data sets, a corresponding amount of storage is released from this reserve.
- *Critical DB2 functions.* The amount of storage to reserve is determined from the installation parameter that gives the maximum number of concurrent DB2 threads.

At all times, the DB2 Storage Manager tracks the amount of storage allocated and the amount available for use as working storage. If any request for storage can cause the storage available to fall below the reserved amount, a Short On Storage (SOS) condition is set and remains in effect until the available storage level rises above the reserved amount. This causes DB2 to contract all internal storage pools (thereby returning any unused storage segments to MVS) when any request is made for additional storage. The intent of this mechanism is to maintain sufficient storage available to satisfy the requirements of all concurrently active threads.

The DB2 storage management scheme has also been designed to take precautions in case of virtual storage overcommitment. If a storage request should cause available storage to fall below the level required for completion of critical DB2 functions that must complete, that request is not granted unless it is made by such a function. Any other request is rejected, thus causing the user task issuing the request to terminate abnormally.

When DB2 is not Short On Storage, each DB2 pool is permitted to expand (i.e., GETMAIN more virtual storage from MVS) or contract (FREEMAIN virtual storage to MVS) as necessary.

DB2 sort. DB2 must sort rows when an ORDER BY, GROUP BY, UNION, or DISTINCT command is used in SQL calls and existing indexes cannot be used. DB2 also needs to sort in the merge-join access method. Initially, VS Sort was used by DB2. However, VS Sort as used by DB2 would require a minimum of 256 kilobytes of virtual storage in the DBAS for each concurrent sort. VS Sort also imposes a relatively large initial overhead, so that sorting of a small number of rows, i.e., a few hundred or less, is very expensive. The number of rows to be sorted by DB2 is expected to be much smaller than that in the normal VS Sort applications because DB2 typically sorts on selected columns of qualifying rows only.

DB2 sort uses much less virtual storage and yet it results in a very efficient small sort.

To avoid these two problems, DB2 Sort has been written to replace VS Sort for sorting performed by DB2. Sorting performed by DB2 utilities continues to use VS Sort because they tend to sort many more rows, and the sorting is done in the address space of the user invoking the utility and not in the DBAS.

DB2 Sort uses much less virtual storage (i.e., about eleven kilobytes minimum for each concurrent sort), and yet it results in a very efficient small sort at the expense of large sort performance. For large sorts with many sort work file I/Os, e.g., sorting of tens of thousands of rows or more, VS Sort is much more efficient because its device-dependent code is optimized for a given device, and it uses more extensive and sophisticated sort algorithms.

The use of virtual storage by DB2 Sort is made flexible by employing a much larger sort work area and more buffers in the Multiple Virtual Storage/Extended Architecture (MVS/XA) environment, resulting in greatly improved large sort performance compared to the MVS/370 environment.

Free space in DB2 table spaces and indexes. When data are initially loaded or subsequently reorganized in a table space, DB2 leaves room in the pages in the table space for records that may be inserted later. This space is known as *free space.*

If no records are to be inserted, having no free space results in the most compact table spaces and indexes,

thus minimizing the number of data pages, index pages, and index levels. This reduces access cost as well as the DASD requirement. If, however, a record is inserted and there is no free space in the table space and index, the record must be placed at the end of the table space. The corresponding index page must be split into two pages, one of which must be written at the end of the index space. This page split in the index leaf page also results in new records and corresponding page splits in the nonleaf pages in the index hierarchy. All this results in a disorganized table space and index, and degrades the performance of index scans, especially for scans of a clustered index.

Because the values of the free space parameters are fixed, their choice in the product design significantly impacts performance. Initially, the following free-space parameters were adopted:

- *Data.* Twenty percent free space within page and every sixth page free, for 67 percent net usage immediately after Load.
- *Index.* Ten percent free space within page and every eleventh page free, for 82 percent net usage immediately after Load.

This would have required 50 percent more I/Os in a table space scan as well as 50 percent more DASD for data, as compared with the case in which no free space had been provided.

Because emphasis was shifting from transaction to query as the primary DB2 environment during the product development cycle, free-space parameters were changed to favor predominantly read-only data at the expense of relatively frequently inserted data.

Zero free space is perfect for read-only data, but any insert—however infrequent—immediately starts a mass disorganization of index and data clustering. Any subsequent reorganization of data removes all free space created as a result of page splits and causes a subsequent insert to start another avalanche of disorganization. Thus a compromise was made that involved five percent free space within a data page and ten percent free space within an index page, but no free pages. Instead of 50 percent more I/Os and DASD, overhead due to free space is now only five percent. With this minimal free space, relatively frequently inserted data may require reorganization more frequently to maintain good performance when accessed via clustering indexes.

Data base design options

DB2 provides options that allow the user to make performance-related design tradeoffs in the physical implementation and access to the data base, which are reviewed here under the headings *Table spaces*, *Indexes*, and *Buffer pools*.

Table spaces. When allocating tables to table spaces, the designer must choose between placing a table in its own table space or grouping several tables together. In most cases it is better to have but one table in each table space. This arrangement has the advantage that the performance effects of such activities as locking, scanning, reorganizing, and recovering a table space are isolated to the single table concerned. For unrelated tables that are small but frequently used, however, or related tables that are often used together and have been appropriately loaded to take advantage of clustered access, grouping into a single table space may offer performance, virtual storage, and DASD space advantages.

Each table may be provided with any number of indexes.

Indexes. The principal decisions to be made in DB2 physical data base design involve the number and location of indexes on the DB2 tables. An index may be required to ensure uniqueness of certain columns, but the most important reason for indexes is to improve performance. When specifying an index, certain tradeoffs can be made. For example, an index can eliminate the need to scan a table to search for a particular row to retrieve or update, but this imposes the additional cost of updating the index itself. An appropriate index removes the need for DB2 to invoke a sort to satisfy an SQL GROUP BY or ORDER BY request. Each table may be provided with any number of indexes, where an index may be defined on a single column or across multiple columns.

Even though an index exists for a given column in a table, one cannot be sure that DB2 will use the index

to access that column. DB2 selects an access path and sometimes determines that it will be more efficient to access the data without using the index. If the table is very small or if a high proportion of rows in the table must be accessed, it is probably cheaper to scan the entire table. In general, however, the larger the table and the lower the level of update activity against that table, the greater the performance benefits of providing an index.

In general, the larger the table and the lower the level of update activity, the greater the benefits of an index.

For tables that are scanned in index order and for any application that accesses multiple rows within some limited range by key values, a clustering index can provide a significant performance enhancement. Even though a table may have several indexes, only one of them can be a clustering index.

Two further index options influence performance: (1) the buffer pool to be used (discussed later) and (2) the size of the logical page or subpage for index leaf pages. When an index is updated, a lock will be held on the logical page. The default logical page size is 1K bytes, but a smaller logical page size may be specified if a high level of concurrent updating is expected for the index. If the index is seldom updated or the level of sharing is very low, a larger logical page size can be used. Defining the logical page size requires the creator of the index to make a tradeoff between lock contention and processing cost. The smaller the logical page size, the lower the lock contention and the higher the CPU time requirement.

Although DB2 does not offer the option of the locking of an individual row, by their choice of logical page sizes for indexes, DB2 users may obtain any desired granularity of locking as opposed to having to select from a small number of discrete locking levels.

Buffer pools. In addition to the size of buffer pools, as discussed later in connection with DB2 storage management, a buffer pool option that can influence DB2 performance is the allocation of tables and indexes to particular pools. A table space or an index can be explicitly allocated to a particular buffer pool or allowed to default to BP0. Thus, for applications where performance is critical, a user may consider assigning the accessed table space(s) and indexes to a separate buffer pool. Alternatively, a buffer pool might be dedicated to indexes alone, to avoid contention for buffer space with data pages. One possible approach, if there is sufficient storage available, is to use BP1 and BP2 for performance-critical applications only, leaving BP0 for DB2 objects and other application requirements.

Application design options

There are a number of DB2 options in the area of application design, the most important of which are reviewed here under the headings *Static and dynamic SQL*, *Controlling data consistency*, and *Locking options*.

Static and dynamic SQL. If the installation is using QMF, some QMF users might make repeated use of stored queries. These queries will incur, at each execution, the cost of the DB2 dynamic bind process for a dynamic SQL statement. The DB2 user can identify these queries and convert them to applications containing static SQL statements, thereby eliminating the repeated dynamic bind costs.

During application program design, the user generally knows what functions the application has to perform and can complete the coding of the imbedded SQL statements. In these cases, static SQL is the appropriate choice. But if the user wishes to input additional SQL statements or create the final form of a SQL statement at execution time, this can be done through the use of dynamic SQL statements. However, the use of dynamic SQL statements in an application program imposes a performance cost on that program. In addition to executing the SQL statement, all the costs of the dynamic bind process are incurred at each invocation of that statement by the program. If the same statement were coded as a static SQL statement, these activities would take place only at program BIND time. But dynamic SQL can have some performance benefits. For example, in a large application program containing many SQL statements, it is possible that some SQL statements on rarely used execution paths can be made dynamic. This has the advantage of reducing the plan size and deferring the acquisition of locks for those statements until they

are actually executed. These benefits may offset the cost of the occasional dynamic bind requirement.

Controlling data consistency. To control data consistency in the tables accessed, users may specify one of two isolation level options at program BIND time: Cursor Stability (CS) or Repeatable Read (RR). Cursor stability ensures that a row read by one application will not be changed by another application while it is being used. Selecting cursor stability provides the greatest concurrency without loss of data consistency for most applications that access a given row only once. To guarantee data consistency for those that access the same row several times, the repeatable read option must be selected. This causes all pages accessed by the application since the last commit point to be locked. Thus concurrent use of those tables is restricted, a situation that may affect overall throughput.

DB2 uses locks to minimize interference between concurrent users.

Locking options. DB2 uses locks to minimize interference between concurrent users and to prevent them from accessing inconsistent data. The performance cost of locking is that it reduces the level of data availability to users and so increases response times. From a performance point of view, it is therefore important to limit both the size of the objects locked and the duration of locking to the minimum required. DB2 itself attempts to achieve this, but it provides additional options to the application designer.

A DB2 user normally specifies the locking granularity for a table space when the table space is created. This allows DB2 to manage the process of acquiring and releasing the appropriate locks on the basis of the desired level of data consistency specified for each transaction that accesses that table space. The locking options have the following performance implications:

- Three types of lock size are supported in the Create Tablespace statement: PAGE, TABLESPACE, or ANY. Lock size TABLESPACE locks the entire table space and offers the least concurrency. Lock size PAGE locks the table space in either an *intent-share* or *intent-exclusive* mode. These modes indicate that the pages of this table space may be locked during the life of the transaction in either a share or an exclusive mode.
- Table space locks are obtained at PREPARE time and freed at COMMIT time for dynamic SQL calls. For static SQL calls, table space locks are obtained at transaction allocation time and freed at deallocation time.
- Because multiple transactions can be concurrently updating different pages in a given table space, PAGE lock size offers the best concurrency. Lock size ANY is the default lock size and uses either PAGE or TABLESPACE lock size; actual usage depends on such parameters as isolation level (cursor stability or repeatable read), access path chosen (table space scan, index scan, or unique index access with equal predicate), and expected number of items to be locked.

The primary consideration in the selection of lock size is the virtual storage required for each outstanding lock, which is approximately 200 bytes. For example, if one transaction locks 10 000 pages in share mode with repeatable read isolation level and another transaction locks 10 000 pages in a different table space in exclusive mode with any isolation level, roughly four megabytes of virtual storage are taken up by these two transactions alone. Clearly, a tradeoff must be made here between virtual storage and concurrency.

A dynamic lock escalation facility could change lock size from page level to table-space level when excessive outstanding locks are held. Since DB2 does not currently support such a facility, the following decisions were made in the implementation of DB2 to minimize the use of virtual storage at the expense of concurrency:

- In general, the default lock size ANY uses TABLESPACE lock size if there is a possibility of more than one outstanding lock held.
- User-specified lock size PAGE is overridden in some cases and replaced by lock size TABLESPACE. This is sometimes required to enforce the repeatable read isolation level, but it still results in a tradeoff between virtual storage and concurrency.

When coding an application program, the locking level may be escalated (from PAGE to TABLE) using the LOCK TABLE statement. As a general guideline, the use of this option should be kept to a minimum because it locks not only the referenced table but also all tables in that table space. However, it is available for those situations in which an application specifically must prevent concurrent use of a table.

DB2 utility options. To support large data bases, DB2 utilities provide various options that reduce the work required or subdivide them into smaller pieces. The purpose of these options is to ensure that a required piece of work or a subset can be completed within a given batch window. Examples of such options are the following:

- Recovery may be limited to a set of pages, a data set, or a table space.
- In place of Full Image Copy, Incremental Image Copy may be used to copy only those pages updated since the last image copy. This option can be extremely advantageous when a small number of pages has been updated in a large table space.
- A partitioned table space may be used instead of a simple table space to subdivide a large table space into smaller partitions. DB2 utilities such as Load, Reorganize, and Image Copy can then operate on one partition at a time.
- If it is not necessary, DB2 utilities do not have to operate on both the data and index together. If required, for example, Runstats, Reorganize, and Recover can operate on an index only. Likewise, Load can be performed either before or after creating an index.

DB2 utilities support on-line use by providing share level (SHRLEVEL) options for Runstats and Image Copy. The specifications of the REFERENCE option permit the table space to be shared by others for read-only operations. The CHANGE option allows the table space to be updated by others but it incurs additional CPU overhead for page locking.

An accounting record is produced for every batch application program and for each instance of a user transaction or query.

Application monitoring and tuning

The DB2 Accounting Facility gathers data related to the execution of a thread and produces accounting data for each user ID on a thread basis. The accounting information for each thread is collected and written to the System Management Facilities (SMF) at thread termination for later analysis by user-written programs. An accounting record is produced for every execution of a batch application program and for each instance of a user transaction or query.

Sample accounting report. To illustrate the data available, Figure 2 shows a sample report from a typical program which a DB2 user might write to analyze accounting data. The report shows average data for multiple executions of an application plan, TRAN27. A similar report could be compiled for a single application plan instance or for a summary by user ID. In the following sections, we note some of the key data items in the report that might be used for application monitoring and tuning, although we do not exhaustively describe the fields. The data include the following major parts:

- *Summary values* shown on lines 13 through 15 indicate the elapsed time for the transaction and the CPU time directly attributable to the user's execution task or allied agent. Because of the use of cross-memory instructions, most of this is usually spent executing instructions in the DB2 address spaces. However, MVS continues to accumulate that time in the user's Address Space Control Block (ASCB). This allows the DB2 accounting function to capture a high percentage of the user's CPU consumption. A sudden increase in these values compared to previous accounting data may indicate that a change has taken place either in the DB2 system or in the application, thus triggering further investigation.

  Lines 19 to 24 in the sample report indicate how many times the transaction was terminated successfully and unsuccessfully.

- *SQL call summary* shows the counts of both manipulative and definitional SQL statements per application execution, which should be relatively stable from report to report.

Figure 2 DB2 accounting data summary

```
 1  Print time  11/07/83 18:28:07                          Report page    5
 2
 3                          DB2 ACCOUNTING DATA
 4                        USER SUB-TOTALS BY PLAN
 5
 6  REPORTING PERIOD                                    DB2 SUB-SYSTEM: DSN
 7  ------------------------------------------------------------------------
 8  Start 10/19/83          8:10:22
 9  End   10/19/83         12:00:42    Elapsed                    3:50:20
10
11  USER / PLAN STATISTICS
12  ------------------------------------------------------------------------
13  User    USER004        Count         241      Mean TCB time  2.45241s
14  Plan     TRAN27        Rate/hr     62.78      Mean SRB time  0.04080s
15                                               Mean elapsed  11.20481s
16
17  ACCOUNTING INVOCATIONS SUMMARY
18  ------------------------------------------------------------------------
19  Normal:                   Abnormal:                 Work unit in doubt:
20  --------------------      --------------------      --------------------
21  New user            0     Appl pgm ABEND      2     Appl pgm ABEND      0
22  Deallocation      235     End of memory       1     End of memory       0
23  End of task         3     Resolve indoubt     0     Resolve indoubt     0
24  Appl pgm end        0     Cancel force        0     Cancel force        0
25
26  SQL CALL SUMMARY
27  ------------------------------------------------------------------------
28    Manipulative          Control                      Definitional
29   (/transaction)      (total  counts)                (total counts)
30  ---------------- | ---------------   ------- create --- drop -- alter
31  Sel/Fch    61.70 | Lock-T         3   Table         0         0         0
32  Insert      2.20 | Grant          0   Index         0         0         0
33  Update      1.90 | Revoke         0   T-spc         0         0         0
34  Delete      0.40 | I-Bind         1   Stgrp         0         0         0
35  Describe     0.0 | Comment        0   Dbase         0         0       n/a
36  Prepare      0.0 | ---------------   Synon         0         0       n/a
37  Open        3.30   Commit     0.99   View          0         0       n/a
38  Close       2.12   Abort      0.01
39
40  BUFFER MANAGER (/transaction)  Pool   0  Pool   1
41  -----------------------------  --------  --------
42  Getpage requests                  917.00    153.00
43  Buffer Pool expansions                 0         0
44  System page updates                 5.30      1.70
45  UW page updates                      0.0       0.0
46  Synch Read I/O                     212.00     43.00
47
48  LOCKING    Suspenions      27   Deadlocks       1    Timeouts        0
```

A key value to monitor is the number of incremental binds (I-BINDS) shown on line 34. These occur if, at BIND time, the plan could not be completed, and the VALIDATE(RUN) parameter was specified. If this number is not zero, an increased response time can be expected for the transaction. All uses of incremental bind should be investigated, and the application plan should be rebound with the VALIDATE(BIND) parameter.

- *Buffer Manager summary* shows the application program's interaction with the DB2 Buffer Manager. The numbers printed on line 42 show how many times the Buffer Manager requested a page, either to read data or to update data. These requests are for a specific page in the data set and do not imply that an I/O was required. Read I/O requests are counted on line 46. These numbers should be as low as possible and should be compared with previous reports. A sudden increase might be the result of a change in the application program.

  On line 44 the system page set write counters are shown. These counters are incremented every time a row residing in a system page is updated. These ratios should be monitored and compared with previous reports.

- *Locking summary* shows how many times an SQL statement was suspended or terminated because of locking (line 48). Ideally, all values shown should be zero. However, some degree of lock contention may be inevitable for certain application mixes that process common data bases.

  These counts are highly dependent on the locking protocol selected at the table space level, on the isolation level requested at BIND time, and on the number of rows retrieved and/or updated.

Although useful information about DB2 can be obtained from the DB2 accounting record, users may wish to merge these records with other SMF records that contain application-program-related data.

Using the DB2 catalog tables

DB2 keeps in its catalog tables extensive information related to performance. This information is updated by application bind as well as by such DB2 definitional SQL calls as CREATE TABLE. Also, the RUNSTATS utility scans a target table space and its related indexes to collect statistics that are saved in the DB2 catalog tables. Some of these statistics concern the physical characteristics of such data as the number of rows, pages, and distinct key values. They are used by DB2 in choosing the least-cost access path. Other statistics may be used as performance indicators. By issuing queries against the catalog tables, it is possible to determine the need for tuning action.

Areas occupied by dropped tables cannot be reused until reorganization is done.

The following are examples of SQL statements that may be issued against the catalog to obtain this information:

```
SELECT PARTITION, TSNAME, FARINDREF, NEARINDREF, CARD, PERCACTIVE, PERCDROP FROM SYSIBM.SYSTABLEPART
```

NEARINDREF and FARINDREF in SYSTABLEPART indicate the number of rows that are not in their original page because of updates with larger variable-length rows. NEAR and FAR refer to the distance between the original and current pages. Additional I/Os are required for each row not on its original page. When the sum of NEAR and FARINDREF exceeds a predetermined threshold, such as one percent of the number of rows in the table space (i.e., the CARD value), reorganization of the table space should be considered so as to improve access cost as well as to reclaim wasted space.

PERCACTIVE reports the percentage of space occupied by rows of data from active tables. That is, PERCACTIVE is the percentage of the total space currently allocated to the table space that is occupied by active rows, as distinct from free space and space previously occupied by deleted rows and dropped tables.

PERCDROP reports the percentage of space occupied by rows of data from dropped tables. This percentage should be tracked to determine when a reorganization should be performed to reclaim wasted space.

Areas occupied by dropped tables cannot be reused until this reorganization is done.

SELECT TBNAME,NAME,CLUSTERING,CLUSTERED FROM SYSIBM.SYSINDEXES

CLUSTERING in SYSINDEXES when YES indicates that a CLUSTER is specified in CREATE INDEX, but CLUSTERED in SYSINDEXES when NO indicates that a table is not actually clustered by this index and that a table space reorganization should be considered either to establish initially or to restore the intended clustering order.

SELECT IXNAME,FAROFFPOS,NEAROFFPOS,LEAFDIST FROM SYSIBM.SYSINDEXPART

NEAROFFPOS and FAROFFPOS in SYSINDEXPART represent the number of times that the next row in index sequence is not on the same page as the prior row. FAROFFPOS is much more critical than NEAROFFPOS in an index scan. FAROFFPOS should be tracked, and when it reaches a given threshold—ten percent of CARD, for example—a table space reorganization should be considered to improve the access cost of a clustering index scan.

LEAFDIST in SYSINDEXPART represents 100 times the average number of pages between successive leaf pages during a sequential search of the index. That is, a LEAFDIST value of 100 means that the next leaf page is always physically adjacent to the current one. A value of 500 means that, on average, four pages exist between the current leaf page and the next one. The number of index I/Os is not affected by an increase in LEAFDIST, but I/O time may increase with LEAFDIST because of increased DASD seek times. Reorganization should be considered when LEAFDIST increases beyond a certain threshold such as 10 000, for example.

The isolation level and plan size of each application plan defined in the system can be obtained via the following:

Select NAME,ISOLATION,PLSIZE from SYSIBM.SYSPLAN

Repeatable read isolation levels of some plans may explain unusually great lock contention. Frequently used transactions having large plans could be the cause of I/O contention on the DB2 directory data base.

The amount of DASD storage allocated to a storage group and to each table space and index within a storage group can be found. This requires first running the STOSPACE utility and then issuing a query to retrieve NAME and SPACE from SYSSTOGROUP, SYSTABLESPACE, and SYSINDEXES.

To determine whether a particular application plan makes use of a given index, the following query might be used:

SELECT BNAME,BTYPE,DNAME,DTYPE FROM SYSIBM.SYSUSAGE

For a given application plan (DNAME), the indexes used may be checked (BNAME). If a particular index (which the user expects to be used by this application) does not appear in BNAME, the index is not used by DB2. The reason might be the particular form of SQL statement used, in which case the user can rephrase the statement in an attempt to secure use of the index. Alternatively, the reason may be that the RUNSTATS utility has not been run against the table space in question. This forces DB2 to use default assumptions about the physical characteristics of tables and indexes, which may be quite different from the real data.

Data base buffer pool assignments must be carefully tracked in the MVS/370 environment because of virtual storage constraints in the DB2 data base services address space. Selecting NAME and BPOOL from SYSDATABASE, SYSTABLESPACE and SYSINDEXES shows the allocation of DB2 objects to buffer pools. The use of small pools other than BP0 can result in excessive write I/Os with a small number of pages written per I/O (less than five, for example). Other consequences of small pools may be a low buffer hit ratio and high CPU overhead due to the incidence of buffer critical conditions.

CLOSERULE in SYSTABLESPACE and SYSINDEXES indicates whether a data set is to be closed when not in use. If frequently used table spaces or indexes have CLOSERULE=YES, this may explain a large amount of VSAM I/O to MVS catalogs and the VSAM volume data set. On the other hand, a large number of table spaces and indexes with CLOSERULE=NO may contribute to virtual storage constraints in the DB2 data base services address space, particularly in the MVS/370 environment.

LOCKRULE in SYSTABLESPACE and PGSIZE in SYSINDEXES may be examined when investigating the effects of locking granularity on concurrency. Having LOCKRULE TABLESPACE or ANY may be responsible

for lock contentions, because table space lock granularity is normally used. PGSIZE, which represents the logical leaf page size, should be small for any index that is subject to frequent insertions or deletions. This minimizes locking contention on leaf pages.

Larger buffer pools can be used to minimize the number of I/O operations.

DB2 system tuning

DB2 storage management. In an MVS/370 environment, the DB2 user must allocate virtual storage in the DB2 data base services address space, where space is required for certain pools and system areas. The storage available in this address space must be distributed among the following areas:

- *DB2 code and MVS system areas.*
- *VSAM control blocks.* Each concurrently open DB2 data set requires VSAM control blocks in the private address space. Thus an area must be available for all DB2 catalog and directory table spaces. Also required are user table spaces for which CLOSERULE=NO is specified and any other user table spaces expected to remain in common use.
- *Environmental Descriptor Manager (EDM) pool.* All DB2 data bases in use require space in the EDM pool for data base descriptors, and each concurrently active plan must reside in the EDM pool. For efficient use of the EDM pool, it should be large enough to contain the Skeleton Cursor Templates (SKCT) for frequently used plans. The reason for this is to minimize I/O operations required to load those SKCTs. In addition, some space must be allowed for fragmentation.
- *The buffer pools.* Data base buffer pools are areas of virtual storage which DB2 uses during the execution of application plans to temporarily store pages of table spaces and indexes. The actual number of buffers in a buffer pool is dynamically assigned by DB2, but the number varies within a range defined by the user. Given sufficient virtual and real storage resources, larger buffer pools can be used to minimize the number of I/O operations required to access the DB2 data bases. At least one Buffer Pool (BP0) must be provided that must be large enough to provide data base read and write buffers for all concurrently active threads. In a constrained environment, it is probably much more efficient to use all the available space for a single buffer pool, rather than to attempt to partition that space into two or more smaller pools.
- *DB2 working storage.* DB2 requires additional space for various buffers, which are allocated in a number of internal DB2 pools. For dynamic SQL, the dynamic bind process uses a variable amount of working storage. This storage increases with the complexity of the SQL statement being bound, the number of tables referenced in that statement, and the number of columns in those tables.

  After completion of dynamic bind (and for static SQL statements) there are further demands upon working storage during the execution phase. Execution of each SQL statement requires an increment of storage. Also, the DB2 sort—if invoked—uses a small base plus a variable number of 4K buffers, up to 54K bytes in total.

In an environment with QMF and dynamic SQL only, the EDM pool must be large enough to contain the data base descriptors and multiple copies of the QMF plan—a relatively small plan of less than 30K bytes. In this environment, most of the available storage can be allocated for use as working storage, where it is required for the dynamic bind process.

Conversely, in a dedicated transaction environment, very little working storage is required, but the application plans are normally much larger. Thus a different distribution is required to avoid storage-related failures. Clearly, in a mixed environment, with both queries and transactions, the question of allocating limited storage becomes even more difficult.

To allow the user to manage these various requirements, DB2 provides a storage management scheme and a number of user options. The DB2 Storage Manager monitors the allocation of storage to concurrently executing threads. If the demand for storage cannot be met, intermittent failures of user tasks

may result. If such virtual-storage-related failures occur, the installation should reduce the demand for virtual storage. This may be achieved by the following actions:

- Decreasing the maximum number of concurrent threads.
- Reducing the size of the buffer pool(s).
- Reducing the size of the EDM pool. Such reductions must be done carefully because the space available for application plans must be sufficient for the number of concurrent threads.

Thread queuing. One way to reduce the demand for storage is to limit the number of concurrent threads that can be active. The installation can use IMS or CICS facilities to control the number of concurrent DB2 agents in an IMS-only or CICS-only environment, but there are no such mechanisms in TSO. To allow the option of limiting the number of concurrent threads in all environments, DB2 provides the following installation parameters:

- The maximum number of concurrent DB2 connections (IDENTIFIES) from background jobs and started tasks.
- The maximum number of concurrent DB2 connections (IDENTIFIES) from TSO foreground.
- The maximum number of concurrent threads for DB2—including IMS, CICS, TSO (foreground and background), and utilities.

When the maximum number of concurrent threads is reached, DB2 queues up later requests to create a thread. The queue is serviced on a first-come first-served basis.

Having a limit on the number of TSO IDENTIFIES permits the installation to control the population size of users logged on to DB2. The primary purpose for this is to permit the installation to maintain acceptable service levels and limit the degree of queuing for DB2 threads in a storage-constrained environment.

No restrictions are provided by DB2 to limit the number of IDENTIFIES from IMS or CICS. IMS and CICS perform one or more IDENTIFIES to establish communication paths to DB2, but these do not consume significant resources by themselves. When an SQL statement is issued on behalf of a transaction, a thread is created (or for CICS, an existing thread may be used). It is the creation of a thread that signals a potential requirement for storage resources. Thus the mechanism provided controls the total number of concurrent DB2 threads for all environments.

The statistics facility. DB2 records statistical data to SMF or GTF at each checkpoint. Statistics for the System Services Address Space (SSAS) and for the Data Base Services Address Space (DBAS) are recorded in separate records.

Sample statistics reports. To illustrate the statistical data available, Figures 3 and 4 show sample reports from a typical program which a user might write to analyze statistical data. The reports show total data for multiple DB2 checkpoints. In the following sections we note some of the key data items that might be used for application monitoring and tuning. We do not attempt an exhaustive description of each field.

The sample report in Figure 3 shows data collected in the DB2 System Services Address Space (SSAS). The data were assembled during a period of approximately fifty-three minutes. During this period, 9378 transactions and queries were executed, as evidenced by the count of threads created and summarized in the reporting period summary. Although this report shows total values, it is also useful to report the other counts on a per-thread basis. The SSAS data include the following major parts.

Record counts. These are the numbers of SMF records written by the DB2 Accounting and Statistics Facility. SSAS and DBAS counts (line 13) are occurrences of two statistics records during the reporting period. The ACCT count is the number of SMF accounting records during the same period. The error codes indicate counts of records not written because of SMF buffer overrun (BUF), records not accepted by SMF (RNA), and SMF not active (ACT).

CPU times. The TCB and SRB are components of CPU time in seconds consumed in the System Services Address Space and the Data Base Services Address Space during the reporting period (lines 13 and 14). Most of the processing time of DB2 users is accounted for in their own address spaces. Only those tasks that execute under the control of DB2 itself—such as logging or buffer writing—are reflected here.

Subsystem Services Component (SSSC). This section contains counts of connects to DB2. It also contains counters for create and terminate threads, and for successful prepare-to-commits, commits, aborts, and synchronizations.

Figure 3 Data collected in DB2 System Services Address Space

```
 1   Print time  11/07/83 18:28:07                        Report Page    2
 2
 3              DB2 STATISTICAL DATA FOR SYSTEM SERVICES
 4                      SUMMARY PAGE - TOTAL DATA
 5  REPORTING PERIOD                                  DB2 SUB-SYSTEM:  DSN
 6  ------------------------------------------------------------------------
 7  From:   09:00:43        Date:          10/19/83       Threads:      9378
 8  To:     09:53:22        Elapsed time    0:52:39       Thrd/sec      2.97
 9
10  RECORD COUNTS              Write error codes                CPU TIMES
11  ----------------------     Dest  BUF  RNA  ACT  ----------------------
12        SSAS  DSAS  ACCT     SMF     0    0    0         Jobstep  SRBtime
13  GOOD    12    12  9371             0    0    0  SSAS     5.008   48.730
14  BAD      0     0     0             0    0    0  DSAS    13.660   45.678
15
16  SSSC
17  ------------------------------------------------------------------------
18  Identify      25   COMMIT          0   Queued          0   Exit       25
19  CREATE      9378   Abort          24                       EOT         1
20  Signon         0   In-doubt        0   Resolve         0   EOM         0
21  Term-all    9372   Prepare         0   SYNCHs       9368   SSIC       26
22
23  AGENT SERVICES                      STORAGE MANAGER
24  ------------------------------      ----------------------------------
25  Alloc failures:                        SOS
26    Res-invalid             0              Contraction          178
27    Res-unavail             0              Critical              23
28    Alloc-deadlock          0              Abend                  2
29
30  LOG MANAGER
31  Calls:          Writes:                Reads from:     Archive Log :
32  ------------------------------------------------------------------------
33  Wait         0  CIs created     1476   Buff       244  Offload        0
34  Noforce  67015  Actv writes     4287   Actv         0  Alloc-R        0
35  Force     4896                          Arch         0  Alloc-W        0
36  BSDS       641  Buffer waits       0                    R-delay        0
37
38  COMMAND DATA
39  ------------------------------------------------------------------------
40    DISPLAY           START              STOP              MISC
41  Database      0   Database      0   Database      0   Rec BSDS       0
42  Thread        1   Trace         0   Trace         0   Rec INDO       0
43  Utility       0   DB2           0   DB2           1   Unr CMDS       0
44                                                        Ter UTIL       0
```

Figure 4 Data collected in DB2 Data Base Services Address Space

```
  Print time  11/07/83 18:28:07                          Report Page    2
1            DB2 STATISTICAL DATA FOR DATA BASE SERVICES
2                      SUMMARY PAGE - TOTAL DATA
3   REPORTING PERIOD                                    DB2 SUB-SYSTEM:  DSN
4  From:   09:00:43       Date          10/19/83       Threads          9378
5  To:     09:53:22       Elapsed time  00:52:39       Threads/sec      2.97
6  SQL
7  Manipulative:      Control:          Definitional:
8  ----------------   ---------------   ------- create --- drop -- alter
9  Sel/Fch   102716   Lock-T        0   Table        37        0        0
10 Insert     10093   Grant         0   Index         0        0        0
11 Update      9779   Revoke        0   T-spc         0        0        0
12 Delete      2114   I-Bind        0   Stgrp         0        0        0
13 Descrb       439   Comment       0   Dbase         0        0      n/a
14 Prpare       476                     Synon         0        0      n/a
15 Open       14410                     View          0        0      n/a
16 Close       9531
17 BUFFER MANAGER                  Pool   0
18 -------------------------      --------
19 Current active buffers                113
20 Getpage requests     (GET)         629143
21 Read I/O operations  (RIO)         166591
22 Buffer Pool expansions                  0
23 Expanded to limit                       0
24 Storage unavailable                     0
25 System page updates  (SWS)         280894
26 UW page updates                         0
27 System pages written (PWS)          27819
28 UW pages written                        0
29 Write I/O operations (WIO)           4010
30 Reads with paging                       0
31 Writes with paging                      0
32 Datasets opened                        73
33 Read I/O per thread                 17.76
34 Getpages per Read I/O                3.78
35 Sys. pg. updates/pages written      10.10
36 Sys. pages written / WIO             6.94
37 SERVICE CONTROLLER  Datasets open   Allocation:       Authorization:
38 ------------------  -------------   ---------------   ---------------
39                     Current   132   Attempts   9377   Attempts  10106
40 PLANS bound      0  Maximum   147   Success    9377   Success   10106
41
42 Auto Bind:        Bind:             Rebind:           Free:
43 ---------------   ---------------   ---------------   ---------------
44 Attempts      0   Add           0   Commands      0   Commands      0
45 Success       0   Replace       0   Attempts      0   Attempts      0
46 Inv.Res       0   Test          0   Plans         0   Plans         0
47
48 LOCKING:      Suspensions     495   Deadlocks     1   Timeouts      0
```

Agent services. These data include counts of suspends, execution unit switches to another SRB and TCB, and information on resource allocation. Most of the data in this section relate to DB2 internals and cannot be used for tuning purposes. The allocation failure counts (lines 26 through 28), however, are monitored and any nonzero values investigated.

Storage Manager. The three Short On Storage (SOS) counters on lines 26 through 28 record the behavior of the DB2 virtual storage management algorithm in the Data Base Services Address Space. The SOS contractions count should be monitored together with the value specified in the DB2 initialization parameters as the maximum number of concurrent threads. As the number of threads increases, the SOS contractions and SOS critical counts will probably increase. It is safe to increase the number of threads as long as the SOS-abend count remains zero.

Log Manager. This part contains information on read and write operations to the recovery log data sets. An important count to monitor is the number of reads from the archive log (line 35), which is normally zero.

The sample report in Figure 4 shows data collected in the DB2 Data Base Services Address Space. These particular data were assembled during the same period as the first report. During this period, multiple application programs and queries were executed. A total of 9378 threads were created. Although this report shows total values, it would also be useful to report these counts on a per-thread basis. The DBAS data include the following major parts.

SQL call. Shown here are the counts of SQL statements. The user should monitor the number of control statements, particularly the number of LOCK TABLE statements, as well as definitional statements, because these statements can affect concurrency. In the sample report in Figure 4, no LOCK TABLE statements are shown as issued in the observed period. The sample report covers executions of both transactions and queries from QMF. Thus we can expect counters for some of the definitional SQL statements. Line 9 indicates that a total of 37 CREATE TABLE statements were issued. Further analysis will probably show that these are the result of SAVE DATA commands in QMF.

Buffer Manager summary. The number of read and write I/OS on a system basis are included in this category. This section shows information on the Buffer Manager. In contrast to the data collected by the Accounting Facility, the Statistics Facility does provide information on the number of write I/OS initiated by the Buffer Manager during the observed period. Line 21 shows the number of reads, and line 29 indicates the number of writes issued. The user should monitor these values as one indicator of the total workload being processed. Line 32 shows the number of times data sets were opened during the period. If this number is consistently high, it may indicate that CLOSERULE=YES has been specified for some frequently used table spaces.

Service controller. These counts show activity in opening data sets, allocation and authorization of plans and counts of binds, rebinds, and automatic binds. If the number of automatic binds is not zero, and automatic binds were not expected, the user should find out what caused plans to become invalid. It may be that an index was dropped either in error or without realizing how many plans would be affected.

Under the heading Datasets Open is shown the number of currently open VSAM data sets (line 39) and the maximum number of concurrently open VSAM data sets during execution of DB2. This number should be compared with the maximum number of open VSAM data sets for which storage has been reserved in the DB2 initialization parameters. If the observed maximum is consistently lower than the value specified, the user may decrease the reserved storage area in order to free virtual storage.

Locking summary. These data are identical to those described earlier in the sample accounting report, except that here they are summarized across all transactions and queries.

Concluding remarks

In this paper we have discussed performance-related strategies for query processing. This has included a discussion of the Structured Query Language (SQL) query optimization, automatic access path selection, bind, and SQL compilation. We have also presented performance-related tradeoffs in the internal design of DB2 components, including those for resource allocation, the Environmental Descriptor Manager, the Buffer Manager, the Storage Manager, DB2 sort, table spaces, indexes, and buffer pools. In addition to DB2 component design options for improving performance, there are also such application design options for improved performance as static and dy-

namic SQL, the control of data consistency, locking options, and DB2 utility options. Techniques for monitoring and tuning are presented for further observation and improvement of operational performance. We have also discussed the collection and analysis of DB2 accounting data and the use of DB2 catalog tables for monitoring and improving performance. Regarding DB2 system tuning, we have presented the collection and analysis of data on storage management, thread queuing, and statistics.

Cited references

1. M. M. Astrahan and D. D. Chamberlin, "Implementation of a structured English query language," *Communications of the ACM* **18,** No. 10, 580–588 (October 1975).
2. H. M. Weiss, "The ORACLE data base management system," *Mini-Micro Systems,* 111–114 (August 1980).
3. M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson, "System R: A relational approach to data base management," *ACM Transactions on Database Systems* **1,** No. 2, 97–137 (June 1976).
4. M. W. Blasgen and K. P. Eswaran, *On the Evaluation of Queries in a Relational Database System,* Research Report RJ-1745, IBM Research Division, 5600 Cottle Road, San Jose, CA 95193 (April 1976).
5. P. Buneman, R. E. Frankel, and R. Nikhil, "An implementation technique for database query languages," *ACM Transactions on Database Systems* **7,** No. 2, 164–186 (June 1982).
6. W. Kim, "On optimizing an SQL-like nested query," *ACM Transactions on Database Systems* **7,** No. 3, 443–469 (September 1982).
7. E. Wong and K. Youssefi, "Decomposition—A strategy for query processing," *ACM Transactions on Database Systems* **1,** No. 3, 223–241 (September 1976).
8. Y. E. Lien, "Design and implementation of a relational database on a minicomputer," *Proceedings of the ACM Annual Conference,* ACM, New York, 1977.
9. A. Makinouchi, M. Tezuka, H. Kitakami, and S. Adachi, "The optimization strategy for query evaluation in RDB/V1," *Proceedings—Very Large Data Bases,* 7th International Conference on Very Large Data Bases, Cannes, France (September 9–11, 1981), pp. 518–529. IEEE Catalog No. 81CH1701-2; available from the IEEE Service Center, Piscataway, NJ 08854.
10. P. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," *Proceedings of '79 SIGMOD Conference,* 1979. Available from ACM Headquarters, 11 W. 42nd St., New York, NY 10036.
11. M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The design and implementation of INGRES," *ACM Transactions on Database Systems* **1,** No. 3, 189–222 (September 1976).
12. J. A. Weeldreyer and O. D. Friesen, "Multics relational data store: An implementation of a relational data base manager," *Proceedings of the 11th Hawaii International Conference on System Science,* 1978, pp. 52–66.
13. M. Stonebraker, J. Woodfill, J. Ranstrom, M. Murphy, M. Meyer, and E. Allman, "Performance enhancements to a relational database system," *ACM Transactions on Database Systems* **8,** No. 2, 189–222 (June 1983).
14. D. D. Chamberlin, M. M. Astrahan, R. A. Lorie, J. W. Mehl, T. G. Price, M. Schkolnick, P. G. Selinger, D. R. Slutz, B. W. Wade, and R. A. Yost, *Support for Repetitive Transactions and Ad-hoc Query in System R,* Research Report RJ-2551, IBM Research Division, 5600 Cottle Road, San Jose, CA 95193 (May 1979).

Reprint Order No. G321-5218.

Josephine M. Cheng *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, CA 95150.* Ms. Cheng joined IBM at the Santa Teresa Laboratory in 1977. Since 1978, she has worked on the Advanced Data Base Project on data base management physical services. In 1979, she worked on the data base management logical services. She is currently working on IBM Database 2 advanced functions development. She received the B.S. degree in mathematics and computer sciences and the M.S. in computer sciences from the University of California, Los Angeles, in 1975 and 1977, respectively.

Christopher R. Loosley *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Mr. Loosley is an advisory programmer on the Advanced Data Base Project of the General Products Division. He joined IBM United Kingdom in 1970, working as a systems engineer and specializing in DB/DC and operating system performance. He later transferred to the United Kingdom Field System Centre, where he established a program of performance monitoring for all UK IMS/VS customers. Since 1978, he has worked on relational data base performance at the Santa Teresa Laboratory. Mr. Loosley received his bachelor's degree in pure mathematics and statistics from the University College of Wales, Aberystwyth.

Akira Shibamiya *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Mr. Shibamiya joined IBM at Kingston, New York, in 1965 to work on the development of the General Purpose System Simulator (GPSS). He later worked on the Computer System Simulator (CSS). Since transferring to San Jose in 1967, he has been involved with performance evaluation of data management and data base products as well as hardware devices. Mr. Shibamiya received a B.S. in engineering science from Stanford University in 1964 and an M.S. in applied mathematics from Harvard University in 1965.

Patricia S. Worthington *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Ms. Worthington joined IBM in 1976 at the Santa Teresa Laboratory, where she worked on the programming language APL. She began working on performance analysis of QMF and DB2 in 1979. Ms. Worthington received a B.S. in mathematics and an M.S. in computer science from Stanford University in 1976.