DB2 12 for z/OS: Technical Overview and Highlights

by John Campbell and Gareth Jones

Introduction

Cloud, Analytics, and Mobile are changing the landscape for enterprise customers. These technology trends are partly driven by an explosion of data and partly by business needs to gain "deeper business insight" from data to improve business efficiency and effectiveness. The enterprise's data server is at the heart of this revolution, and it is important that the data servers are agile, secure, and provide seamless integration to these newer technologies. This paper will provide an overview of how DB2 12 addresses these business and technological needs.

Why You Should Read This Paper

This paper is targeted at IT Architects, such as Enterprise, System, Software, and Data Architects, who work with business leaders and subject matter experts. It will help architects ensure that business and IT are aligned and also ensure that they design and deliver an architecture that supports the most efficient and secure IT environment meeting an enterprise's business needs.

Highlights

This IBM® DB2® for z/OS® white paper provides a high-level overview of some of the key changes introduced in DB2 12 for z/OS, including the following topics:

- Performance for Traditional Workloads
- Performance Enablers for Modern Applications
- Application Enablement
- RAS—Reliability, Availability, Scalability, plus Security
- Migration and Prerequisites

Before looking at those areas in detail, it's important to put them in context in terms of the goals that the DB2 for z/OS development team set themselves, in four broad themes:

- Application enablement
- DBA function
- OLTP performance
- Query performance

Application Enablement

DB2 development set themselves the target of addressing a number of key customer requirements to expand the use of the existing features of DB2, as well as delivering mobile, hybrid cloud, and DevOps enablement. Two more objectives in the application enablement area were to provide enhance IDAA functionality to support an expanded number of use cases, and to provide incremental improvements in the SQL and SQL/PL areas to make DB2 ready for the next wave of applications.

DBA Function

Even with the existing capability to grow partitioned table spaces to 128 TB, some customers have been constrained by table and partition scalability limits, and addressing this issue has become one of the goals of this release. To complement this, another objective is to simplify large table management. Further goals are to remove the biggest inhibitors to 24 × 7 continuous availability and to provide incremental security and compliance improvements.

OLTP Performance

OLTP performance is still one of the biggest requirements for our customers, so the goal in this area is to build on the improvements in DB2 10 and DB2 11 to deliver even more performance improvements. For DB2 12, the goals are to reduce CPU consumption in the range of 5–10% by exploiting in-memory features, to double the throughput when inserting into a non-clustered table, to remove system scaling bottlenecks associated with high *n*-way systems, and to provide incremental improvements related to serviceability and availability.

Query Performance

Query performance for OLAP, BI, and other more complex workloads also remains a focus for our customers, and we have targeted four improvements in this area, to build on the work done in DB2 11: 20–30% CPU reduction for complex query workloads, improved efficiency delivered by reducing other resource consumption, a particular target of 80% for `UNION ALL` performance improvement; and simplified access path management, especially for dynamic SQL.

Quick Hits

Let's have a look at some of the highlights of this release before moving on to discuss the specific changes in detail.

Scale and Speed for the Next Era of Mobile Applications

DB2 development has measured over 1 million inserts per second, and we believe we can scale higher.

DB2 can also support up to 256 trillion rows in a single table, with agile partition technology.

In-memory Database

In-memory database is a major theme for this release. DB2 development has measured up to 23% CPU reduction for index lookup with advanced in-memory techniques.

Next-generation Application Support

In terms of next-generation application support, DB2 can now handle up to 360 million transactions per hour through a RESTful Web API into DB2.

Deliver Analytical Insights Faster

Response time is not just a requirement for OLTP workloads, but also for analytical workloads, where DB2 can deliver up to a two-times speed-up for query workloads, and for targeted queries, up to a 100-times speed-up.

Performance for Traditional Workloads

In this section, we'll look at changes in DB2 12 for z/OS that improve performance for traditional workloads.

In-memory Computing

DB2 12 places a strong emphasis on *in-memory computing*, combining large real memory size together with memory-optimized data structures to drive performance improvements. Unlike in prior releases, where some of the performance improvements were available without necessarily making use of more real memory available to DB2, many of the DB2 12 enhancements require customers to provision more real memory for DB2 to exploit before they can realise the significant performance gains. All the enhancements in this section require additional real memory. As a general comment on real memory provisioning, customers should plan to avoid all paging and make sure that they have sufficient free real memory to run safely.

In-memory Contiguous Buffer Pools

One of the features that falls into this category is the in-memory contiguous buffer pool. The objective of this buffer pool option is to cache entire table spaces or index spaces in

the buffer pool. The larger the object, the larger the buffer pool, and the larger the buffer pool, the more real memory is required.

The in-memory contiguous buffer pool improves performance and reduces CPU consumption by providing direct page access in memory—DB2 development has measured up to an 8% CPU reduction for OLTP workloads using this feature. Direct page access greatly reduces the CPU overhead of Get Page and Release Page operations and is achieved by laying out objects contiguously in page order in the buffer pool, and then accessing the page directly in memory.

This is not the first step taken by DB2 to provide support for in-memory buffer pools. DB2 10 introduced the `PGSTEAL(NONE)` buffer pool attribute for objects such as table spaces and indexes that can fit in their entirety into a buffer pool. The pages for each object are prefetched into the buffer pool when that object is first accessed, saving subsequent I/O, saving CPU, and providing elapsed time benefits. In DB2 10 and 11, prefetch is disabled, saving CPU. However, DB2 10 and 11 buffer pools defined with `PGSTEAL(NONE)` still maintained hash chains and LRU chains, with the CPU cost still visible for workloads running with large size buffer pools and `getpage`-intensive processing.

DB2 12 changes `PGSTEAL(NONE)` functionality to avoid the LRU and hash chain management overheads. And to make this feature more resilient, DB2 12 introduces an overflow area that is automatically managed by DB2. An overflow area is reserved by DB2 from the buffer pool allocation, and represents 10% of the buffer pool size. It is only used if the objects assigned to the buffer pool do not fit into 90% of the buffer pool size. It is allocated when the buffer pool is allocated, but is only backed by real storage when it is used. Any pages in the overflow area are automatically managed by DB2 on an LRU basis. While no page stealing occurs within the main buffer pool area, it is possible in the overflow area.

In-memory Index Optimization

As customer tables grow larger and larger, index sizes inevitably grow, too. The larger the index in terms of the number of levels, the greater the cost of random index access becomes.

Prior to DB2 12, DB2 made use of index lookaside, to try to avoid the full cost of index probing. However, this typically only benefited skip sequential access through the index. Random SQL only benefited from index lookaside occasionally, often requiring a Get Page for each level of the index.

To make index lookups faster and cheaper, DB2 12 introduces the Index Fast Traverse Block (FTB) in-memory area to provide fast index lookups for random index access for unique indexes with a key size of 64 bytes or less. It is a memory-optimized structure, and contains the non-leaf pages of the index. The FTB area resides outside of the buffer pool, in a memory area managed by DB2, and requires additional real memory. Unique indexes with include columns are also supported provided the total length is 64 bytes or less.

A new zparm, INDEX_MEMORY_CONTROL, is used to control the size of this memory area, with a minimum size of either 500 MB or 20% of total allocated buffer pool storage, whichever is larger, and a maximum size of 200 GB. Customers who wish to control the introduction of the FTB area at a system-wide level can use the zparm to disable fast index traversal, but there is also a new catalog table, SYSIBM.SYSINDEXCONTROL, which is a mechanism for controlling fast index traversal at the index level, by time window.

DB2 automatically determines over time which unique indexes would benefit from the FTB area. Fast index traversal is not attractive for indexes that suffer from frequent leaf page splits, due to inserts and updates, for example, but when the index is used predominantly for read access via key lookups, a significant performance improvement is expected. SELECT, INSERT, UPDATE, and DELETE can all benefit from FTB area because they can avoid traversing the index b-tree. The savings can be substantial, depending on the number of levels in the index b-tree.

A new zIIP-eligible Index Memory Optimization Daemon monitors index usage, and allocates FTB storage to indexes that will benefit. Using the FTB, DB2 can do very fast traversals through the non-leaf page index levels without having to do page-oriented access.

Customers can monitor FTB area usage using the new DISPLAY STATS command, which shows which indexes are using the FTB area. Also for use by customers and by IBM service are two new IFCIDs, 389 and 477, which allow the tracking of FTB area usage at a detailed level.

Figure 1 shows, based on measurements taken by DB2 development, that simple random index lookup is faster and cheaper in DB2 12. It also shows that the greater the number of index levels, the greater the expected CPU savings, varying from 6% for a two-level index to 23% for a five-level index. The expected CPU savings will increase with the number of index levels. Index-only access will show higher percentage CPU savings.

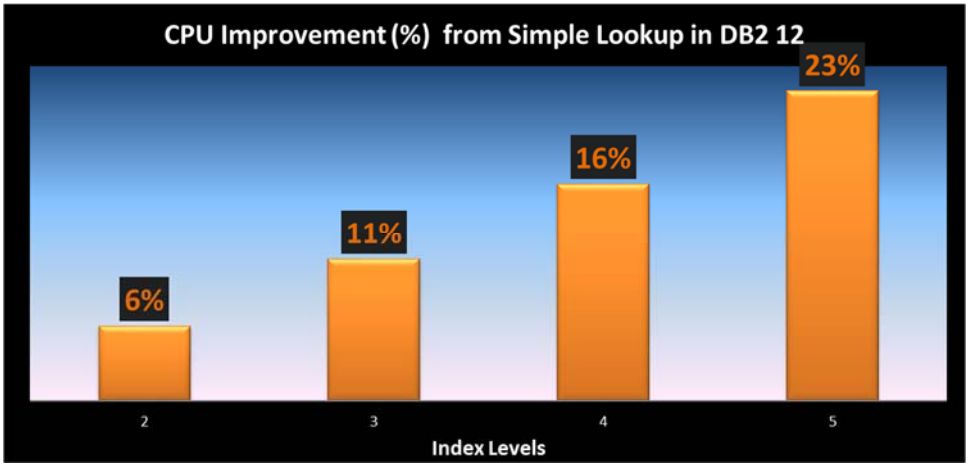

Figure 1: CPU improvement using fast index traversal

INSERT Performance

`INSERT` workloads are very common across the DB2 for z/OS customer base and are often performance critical. A typical use case is the journal or audit table, with a high rate of concurrent insert, where rows are inserted at the end of the table space of partition, and where data clustering is not required—it's not always necessary for the rows to be inserted in the order of the clustering index. Some customers try to balance the need for fast insert processing with the need to be able to query the data rows in key order by processing the inserted rows again later to populate other tables with the rows now in clustering key order.

However, performance has historically been a challenge for some customers who need the capability to insert very large volumes of data rows into the database at very high speed. This performance issue was addressed in prior releases of DB2 by forcing the insert of new rows at the end of the current partition for table spaces/tables with the `MEMBER CLUSTER` and `APPEND` attributes, without regard to data row clustering. Prior to DB2 12, it was often the case that the space search algorithm used for the tablespace or partition that could be a bottleneck for insert performance, and it is that bottleneck that is addressed in DB2 12.

However, some customers need even more improvement in insert throughput performance, and DB2 12 takes a significant step forward with a new fast `INSERT` algorithm, which streamlines the free space search operation. This feature is targeted specifically at UTS table spaces with the `MEMBER CLUSTER` attribute. The use cases that will benefit from this change are broadened, as DB2 does not consider the `APPEND` table attribute when determining which tables qualify for the new algorithm. The old insert algorithm is known as *Algorithm 1*, and the new, faster insert algorithm is known as *Algorithm 2*.

However, this feature is not available until new function has been enabled (or *activated*) in DB2 12 (now called ANFA, or After New Function Activation). This is because the new fast insert algorithm is dependent on new log records that are introduced with ANFA.

To summarize the requirements for the new fast insert algorithm:

- New function has been activated—ANFA.
- The UTS table space type is used.
- The table space is defined with Member Cluster—MC.

To allow customers to control usage of the new algorithm, it may be turned off via new `zparm DEFAULT_INSERT_ALGORITHM`, or at the table space level via the DDL attribute `INSERT ALGORITHM`. The default `zparm` setting when new function has been activated is to use the new algorithm.

Customers who want to exploit this new feature should plan for additional real memory and larger size buffer pools, for each qualifying table space partition and for each DB2 member—that is, there is an additional real memory requirement per partition per member.

Figure 2 below shows faster insert rates into a group buffer pool–dependent partition by range (PBR) universal table space (UTS) defined with `MEMBER CLUSTER` and `APPEND` in two-way data sharing in DB2 12 AFNA. This is a special use case, as the table has no indexes, but it does demonstrate the benefits of faster insert into a table space without any of the side effects introduced by indexes. The workload consists of a number of short, fast insert processes running in parallel, inserting a small number of rows each. There are three things to note with this chart:

- The number of inserts per second increases from under 800,000 per second to over 1,000,000 per second—a 25% improvement.
- The class 2 elapsed time per transaction with Algorithm 1 at 0.012 seconds per transaction is reduced dramatically to 0.002 seconds per transaction.
- The class 2 CPU time per transaction is reduced by about 20%.

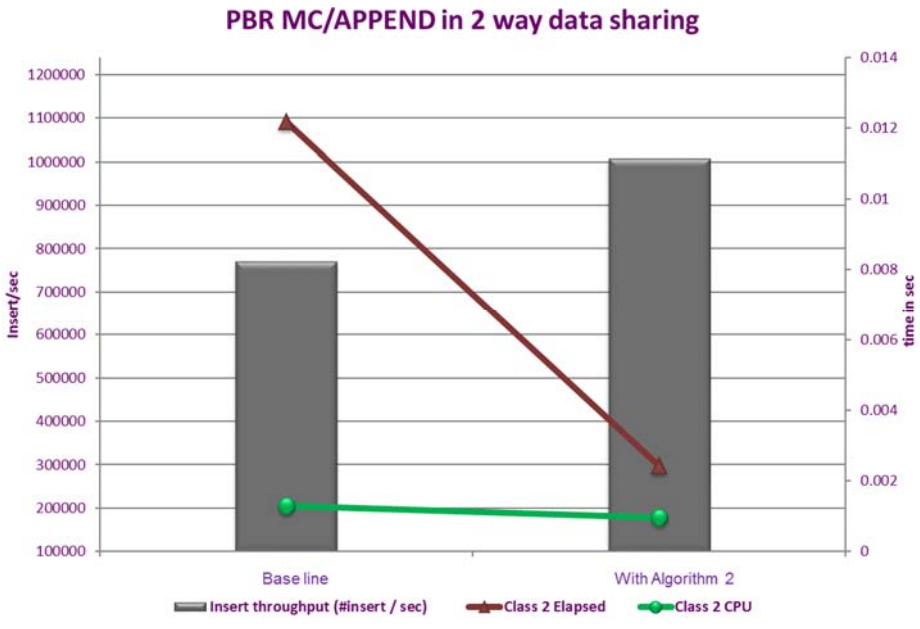

Figure 2: Response time and CPU time improvements with new fast insert algorithm

A more common scenario is outlined in Figure 3, which is based on an application journal. Just as in the previous example, the experiments were performed in a two-way data sharing environment, with group buffer pool dependency. This is one of the use cases making up the high-insert performance workload test suite run by DB2 development on a continuous basis at the IBM Silicon Valley Laboratory (SVL).

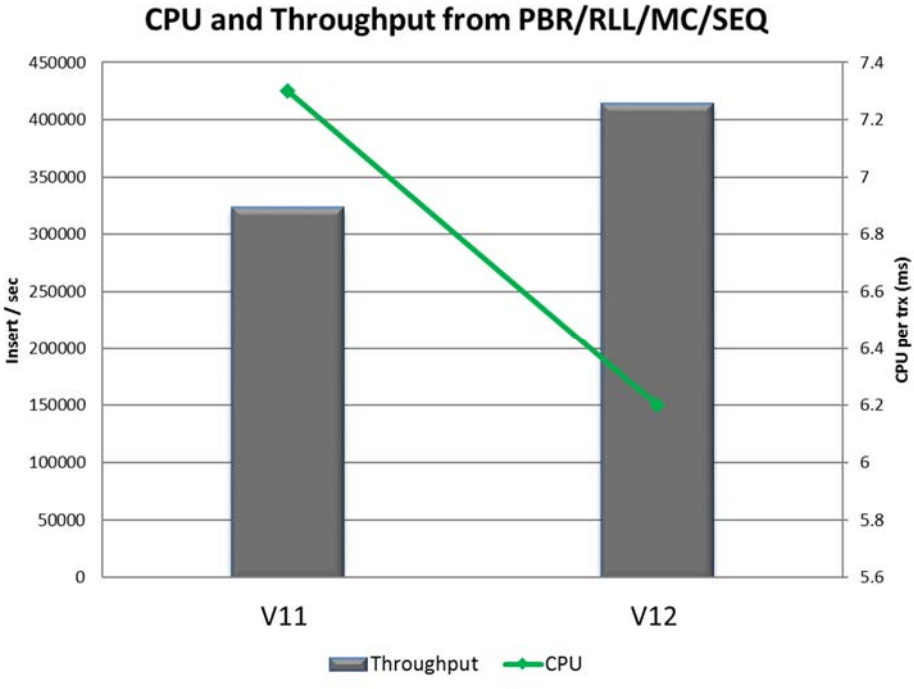

Figure 3: Improved throughput and CPU with the fast insert algorithm

There are three tables involved in this workload, one table with one index, the second with two indexes, and the third with three indexes. The journal table is defined as PBR UTS with row-level locking, and with Member Cluster. The index is based on a sequential key, and the rows arrive at the table in an order based on that sequential key.

The chart compares insert throughput (inserts per second) and CPU per transaction in milliseconds on DB2 11 with insert Algorithm 1, and on DB2 12 with insert Algorithm 2. The CPU per transaction drops significantly, from around 7.3 seconds per transaction to 6.2, approximately a 15% reduction. The insert throughput climbs dramatically, from just over 300,000 per second to just over 400,000 per second—around a 30% increase.

Bear in mind that the benefits you will see will vary, and workloads that are constrained by lock/latch contention on the space map pages and on the data pages in the table space or table space partition are more likely to benefit from the new insert algorithm.

Access Path/Access Plan Stability

Many customers rely on DB2's plan stability management features to stabilize access paths, to revert back to a previous access path if they face a performance regression after a BIND REPLACE or REBIND operation, or both. Plan stability was delivered for static SQL in DB2 9, evolved in DB2 10 and DB2 11, and receives additional usability enhancements in DB2 12. The main points we are going to explore in detail are:

- Dynamic SQL plan stability—the headline enhancement in this area
- Incremental enhancements to improve static plan stability usability
- Preserving the local dynamic statement cache (DSC) at rollback
    - At some customer sites, application programmers are using ROLLBACK instead of commit at the end of a read-only unit of work.
    - This causes the statement to be purged from the local DSC and drives an expensive full prepare.
        - A second side-effect is to pollute the DB2 accounting and statistics metrics.
    - At rollback, the access path is now preserved in the dynamic statement cache, rather than being invalidated, providing customers with a welcome performance enhancement by replacing the cost of a short prepare with prepare avoidance.
- Integration of RUNSTATS and the optimizer, which will provide automatic update of statistics profiles
- Improved support for statistics profiles, with automatic update driven by CREATE/ALTER INDEX DDL, and usage of statistics profiles by inline statistics collection
- Simplified creation of all the tables required by EXPLAIN, via a new stored procedure, ADMIN_EXPLAIN_MAINT.

Dynamic SQL Plan Stability

For many customers, unstable performance of repeating dynamic SQL statements has been a problem. Dynamic SQL is prepared—that is, the access path is recalculated—at every Global Dynamic Statement Cache (DSC) miss via an expensive full prepare. If statements are regularly purged from the Global DSC, they can be reoptimized several times a day. Environmental changes can result in access path changes when a dynamic SQL is re-prepared, possibly leading to performance regression, and this can be difficult to manage. This includes changes introduced by:

- RUNSTATS updating the DB2 catalog to reflect changes in data volumes and column value distribution
- Software maintenance affecting optimizer choices
- Migration to a new release of DB2
- zparm (system parameter) changes
- Database schema changes

Prior to DB2 12, static SQL had several advantages. The access path is established and locked in at BIND time, and, with a few exceptions, is not recalculated until there is an explicit BIND REPLACE or REBIND operation. The static plan management features introduced in DB2 9 and DB2 10 provide advanced plan management functions, including the ability to fall back to a prior access path.

DB2 12 makes static SQL advantages available to repeating dynamic SQL workloads. DB2 12 does this by providing the infrastructure to save, reuse, monitor, and manage runtime structures for dynamic SQL. It stores dynamic SQL statements and their associated Global DSC structures in the DB2 catalog, so that they can be reloaded into the Global DSC from the catalog on a Global DSC miss to reuse the runtime structures on subsequent executions. This provides access path stability, consistency, and predictability, including across members of a data sharing group. The capability to maintain access path stability is unaffected by events such as DB2 stop and restart, RUNSTATS, DB2 maintenance, and release migration.

DB2 12 provides a number of mechanisms to control and manage dynamic SQL plan stability.

New zparm CACHEDYN_STABILIZATION controls whether dynamic SQL statements or statement groups can be captured for stabilization and saved in the DB2 catalog, and whether they can be loaded into the Global DSC from the catalog when there is a miss in the Global DSC.

New commands can be used to perform a one-time capture of qualifying statements—or logical statement groups—in the Global DSC, or to keep monitoring the Global DSC for statements that qualify. That is, capture can be performed with or without monitoring. Statements can qualify based on a statement ID, a current SQLID, the number of statement executions, or a combination of these. In addition, customers can set a global variable to drive stabilization of dynamic SQL for specific applications. It is important to understand that, as well as SQL statement text and AUTHID, APPLCOMPAT is also part of the matching criteria.

It is possible to remove statements from stabilization control with new options on the FREE command, that is, remove them from the DB2 catalog and from the Global DSC, if present. It is possible to free an individual stabilized statement, or a stabilized statement group. When this is done, not only is the statement or statement group removed from the DB2 catalog, but it is also invalidated in the DSC.

Related to this, the LASTUSED statistic for stabilized dynamic SQL can be used to identify stale statements, which can then be freed from the DB2 catalog.

There is full EXPLAIN support, enabling customers to explain the saved current copy and a saved invalid copy, so that they can understand the access path chosen and optionally take action to favor a particular access path.

Full invalidation support is retained—database schema changes (SQL DDL) and authorization changes (SQL DCL) can invalidate stabilized dynamic SQL statements, in the same way that such changes can invalidate static SQL packages.

DB2 12 provides instrumentation so that customers can know which statements are using which stabilized runtime structures. The statement hash or query hash—a value derived from the SQL statement text itself—is externalized so that customers can use the hash as a stable identifier to monitor statements over time, even if they move in and out of the global statement cache and/or the catalog as stabilized dynamic SQL, including across recycles of DB2. In addition to statement tracking and explain support, customers can also obtain statistics about the cache and catalog hit ratios.

There are some key limitations in this initial implementation of dynamic SQL plan stability. Concentrate statements with literals and bitemporal are not supported for stabilization, and plan management features such as REBIND SWITCH or APREUSE/APCOMPARE are not yet available.

Static Plan Stability: Usability

The usability of static plan stability is improved in three areas: FREE PACKAGE and REBIND PACKAGE.

DB2 9 introduced plan management to retain copies of the original, previous, and current versions of a package. Prior to DB2 12, only two options were available when freeing packages: to free all copies of a package, or to free the inactive copies (original and previous). In DB2 12, FREE PACKAGE is enhanced to provide the capability to selectively FREE either the PREVIOUS or ORIGINAL individually. This allows customers to easily remove a stale ORIGINAL copy without removing an up-to-date PREVIOUS copy. It is also now possible to choose to FREE only invalid copies after schema changes or release migration has invalidated old copies.

Customers can now also free invalid package copies when an application using the current copy of the packages is running because the exclusive package lock held by the running application no longer extends to the invalid copies.

DB2 10 introduced the capability to influence the optimizer to reuse the previous access path at REBIND time using the APREUSE option. This has proved to be very successful, with customers requesting further enhancements. DB2 11 provided customers with more flexibility by delivering the APREUSE(WARN) option.

DB2 12 takes this further by providing a new option for REBIND PACKAGE, called APREUSESOURCE. This allows customers to explicitly specify whether the current, previous, or original copy should be used as the access path source for APREUSE.

As well as enhancing usability, this also addresses the problem where an invalid package copy accidentally becomes current. This could occur where, for example, in the following scenario:

- A DROP and CREATE of an index causes the current package copy to become invalid.
- When the package is automatically rebound by DB2, it picks up a bad access path.
- The previous and original versions are also invalid, but at least one of them contains the desired access path.
- REBIND SWITCH to one of these results in an invalid package becoming current, with the risk of automatic rebind before REBIND with APREUSE can be executed.

REBIND with APREUSESOURCE avoids the two-step process to restore a previous good access path and additionally avoids the risk of an invalid package becoming the current package.

Finally, statement literal concentration is now a BIND or REBIND package parameter.

RUNSTATS Enhancements for SQL Performance

DB2 12 makes a number of improvement to RUNSTATS for improved access path selection.

It enhances the CLUSTERRATIO formula to handle the impact that unclustered inserts have on the clustering index, and the impact of the space search algorithm on inserts into table spaces with larger segment size. This ensures that CLUSTERRATIO better reflects dynamic prefetch behavior.

Prior to DB2 12, RUNSTATS (and other utilities such as REORG and LOAD collecting inline statistics) invalidated dynamic SQL statements in the global dynamic statement cache that were dependent on the object being operated on. DB2 12 adds an INVALIDATECACHE option to RUNSTATS, with a default of NO, to prevent RUNSTATS from invalidating the cache unless the parameter is set to YES. Note that this is a change in the default behavior of RUNSTATS from previous releases.

RUNSTATS UPDATE(NONE) REPORT(NO) will continue to invalidate the cache to ensure that existing customer jobs are not impacted. For other utilities, statement invalidation will only occur if the object was in a restricted state before the utility began—such as rebuild pending or reorg pending states for example. Other utilities (e.g. LOAD, REORG) do not provide an option to control dynamic statement cache invalidation.

Statistics Profiles are now supported by inline stats, ensuring that they can be used whenever RUNSTATS can be collected via the REORG and LOAD REPLACE utilities. Additionally, index changes via SQL DDL will update the profile to ensure that dropped indexes are removed, or new indexes are added.

DB2 12 provides an automated value for COUNT for FREQVAL, for the collection of statistics on skewed column values. Knowing what number to specify for FREQVAL

COUNT *n* has always been data dependent—and thus each situation may require a different value. In DB2 12, specifying FREQVAL without the COUNT *n* keywords will result in DB2 automatically determining the appropriate number of frequently occurring column values to collect. The objective is to allow DB2 to collect all the skewed values, up to the top 100 most skewed values, or until no skew can be detected for the remaining values.

Determining which statistics to collect has often been a challenge for customers, especially when large numbers of objects have to be managed. To address this, DB2 11 provided optimizer externalization of missing statistics into catalog and explain tables. DB2 12 takes this a stage further so that the optimizer automatically updates the Statistics Profile with RUNSTATS recommendations. Statistics collection, specifying USE PROFILE, whether via a standalone RUNSTATS utility or via REORG and LOAD REPLACE inline statistics, will automatically include all statistics recommended by the optimizer.

This new feature is known as enhanced statistics profile management. As shown in Figure 4, when calculating the access path for a given query, DB2 will update the profile in the catalog to ensure that missing statistics are collected when collecting statistics via a statistics profile. The flows in red are the new flows in DB2 12.

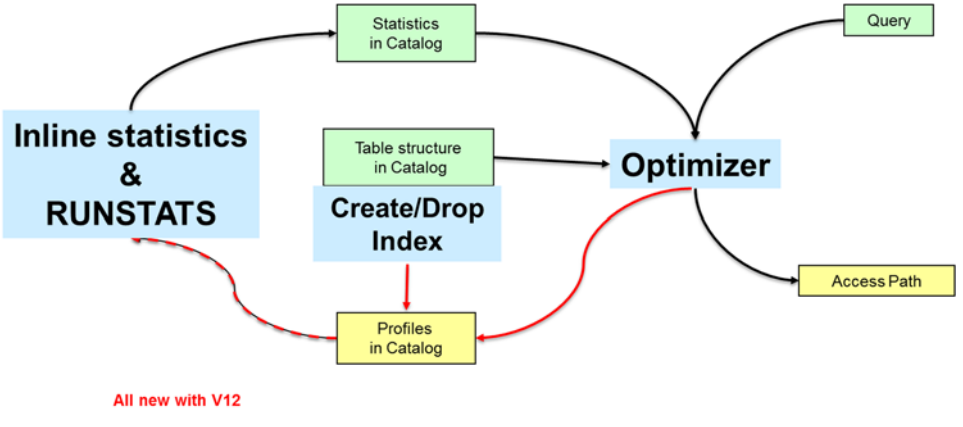

Figure 4: Automatic update of statistics profiles by the DB2 optimizer

First, this diagram shows that the optimizer will update the profile to ensure that missing or conflicting statistics will be corrected by RUNSTATS. This is controlled by new zparm STATFDBK_PROFILE.

Second, CREATE and DROP INDEX will also update the profile.

And finally, this diagram shows that statistics profiles are supported for inline statistics as well as standalone RUNSTATS.

DSNACCOX will recommend RUNSTATS whenever a profile has been updated. Specify USE PROFILE on RUNSTATS to collect current statistics recommendations.

Automatic Creation of All Tables Required by EXPLAIN

DB2 12 provides the new ADMIN_EXPLAIN_MAINT stored procedure. For a specified schema, the procedure will update the structure of all existing explain tables to the new format, create any missing explain tables, or create new set of explain tables. It can also drop or drop and recreate explain tables.

More Granular Global Commit LSN and Read LSN

DB2 use the Commit LSN (log sequence number) for lock avoidance checking, in order to avoid taking unnecessary locks and to reduce CPU consumption. Previously, in a data sharing environment, there was a single Commit LSN value across all group buffer dependent objects. The Global Commit LSN is the Unit of Recovery ID (URID) of the oldest uncommitted Unit of Recovery (UR) in the data sharing group, and is used to do lock avoidance checking for all group buffer pool-dependent objects in the data sharing group. This means that a single long-running program that does not commit, or that only commits infrequently, holds back the Commit LSN (CLSN), degrades lock avoidance, and causes poor space reuse when inserting LOBs.

To mitigate the impact of long-running URs, DB2 12 maintains the Global CLSN of the 500 worst objects, and whether an object outside those 500 objects is being accessed.

DB2 also provides a more granular current Read LSN value, which provides better space reuse from DELETEs when inserting LOBs.

Avoid Scheduling Unnecessary Prefetch

This feature is designed to avoid the CPU overhead of unnecessary scheduling of dynamic prefetch engines. The problem it addresses is that, prior to DB2 12, when all the pages being accessed on behalf of the application are already in buffer pool memory, DB2 still schedules a prefetch engine, even though there is no need to do so. This wastes CPU, unnecessarily ties up a prefetch engine, and can cause "out of prefetch engine" conditions, affecting other applications running on that DB2 member or subsystem.

DB2 development has tried to solve this problem in previous releases of DB2, but it has finally been solved in DB2 12, by avoiding scheduling the prefetch engine until the first buffer miss (that is, the page is not found in the buffer pool). While the benefits of this feature are clearly workload dependent, a CPU reduction of up to 6.8% for OLTP workloads and up to 4.5% for query workloads has been observed.

Miscellaneous Performance Enhancements for Traditional Workloads

DB2 12 adds a new buffer pool advisory mode that simulates the effects of larger size local buffer pools, in line with the emphasis on larger real memory exploitation. This provides a new set of instrumentation, simulating what would happen in terms of reducing I/O. These metrics can be seen in the DB2 statistics trace and in the output of the -DISPLAY BUFFERPOOL command. There is a small CPU overhead of using buffer pool simulation—about 1–2% per buffer pool—and about a 1% real memory overhead,