IDUG Content Blog (p/bl/et/blogid=278)

DB2 12 In-Memory Feature: Fast Index Traversal

Akiko Hoshikawa

DB2 12 In-Memory Feature: Fast Index Traversal

Akiko Hoshikawa & Nina Bronnikova, IBM, Silicon Valley Lab

DB2 12 is classified as an in-memory data base by Gartner. It is not just a simple in-memory data base but a very smart one. In this paper, we explain one of the most innovative in memory optimizations in DB2 12.

Why In-Memory Optimization in Index?

The fastest way to access a record in a table is to use direct index lookups. Many Online Transaction Processing (OLTP) applications make extensive use of direct index lookups, where the application provides a fully qualified key that uniquely identifies a single row in the target table. The access pattern for such key-based lookups is often random, as these keys are usually based on customer-id, sales-

record, or card number. As the volume of data grows, the size and levels of the index grow, as does the cost of traversing the index tree structure. Index traversal here means accessing the index from top down, from the root to non-leaf pages, and then to the leaf page as illustrated in Figure 1 below.

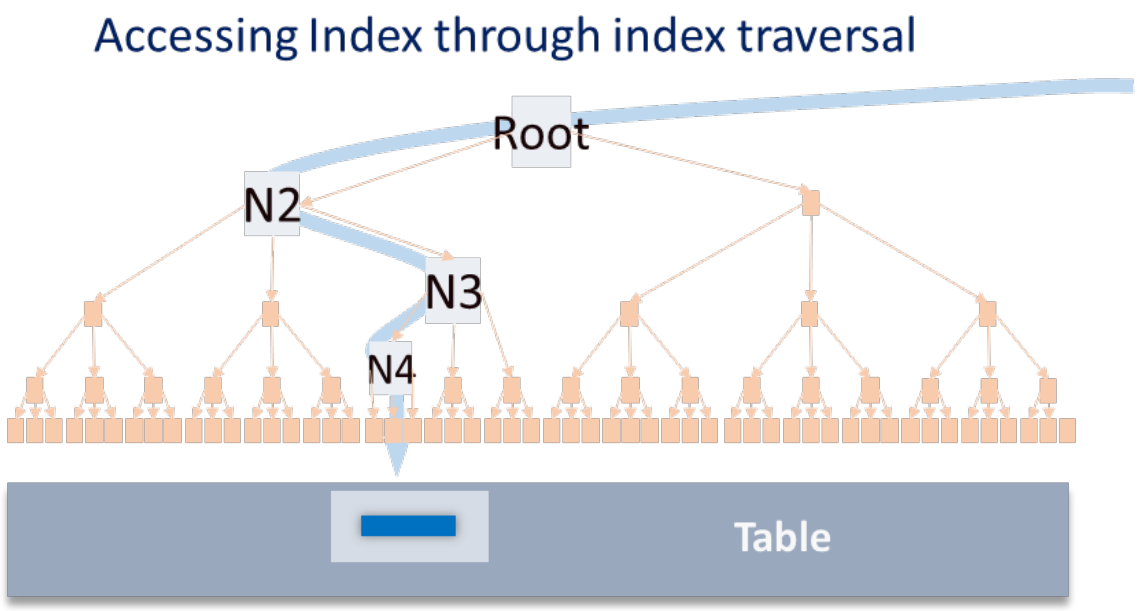

Figure 1. Index traversal access

DB2 10 for z/OS introduced hash access capability to store tables into hash-based table spaces and eliminated the need for indexes. This works well if the access to the table is always random. However, the tables are often accessed not only randomly via unique keys, but also sequentially through batch accesses. The performance impact on sequential access with hash-based table spaces can negate the performance benefit for random access.

Meanwhile, DB2 indexes continue to grow in size, as do the number of index levels.

DB2 12 for z/OS takes a new approach by storing index information in a memory structure to reduce the index traversal cost for primary key lookups. This memory structure, internally called Fast Traversal Blocks (FTBs), holds the root and non-leaf pages of the indexes. The Fast Traversal Block structure is specifically designed to be processor cache friendly for traversing index keys and the index tree traversal operations can be bypassed as illustrated in Figure 2. Because of this in memory structure, we often refer to this feature as Fast Index Traversal.

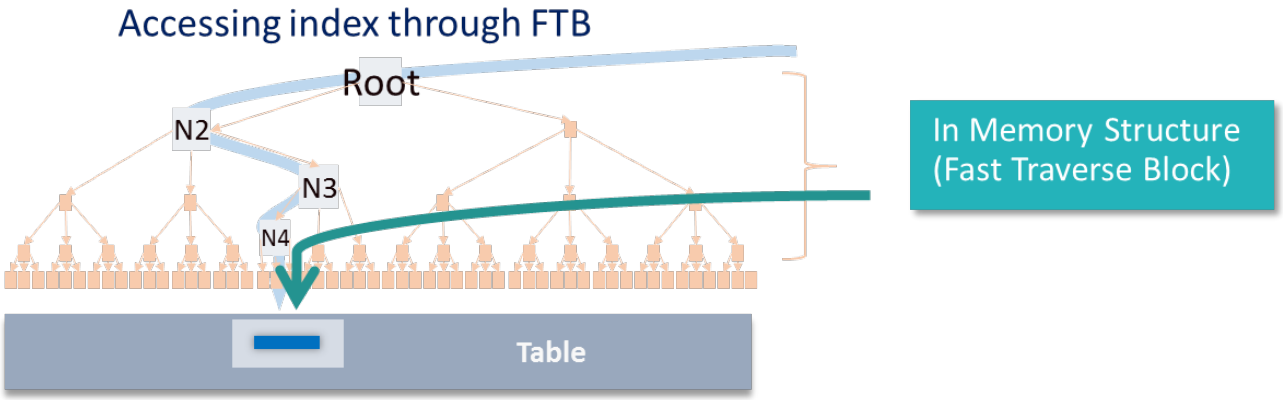

Figure 2. index traversal in DB2 12

How FTB improves performance

The Fast Traversal Block improves performance in several different ways.

- Traversal within a single page is faster due to the simple layout of FTB pages.
- Traversal of one FTB page triggers at most one L2 cache miss because the page size is always equal the cache line size.
- One traversal through all FTB levels requires getting a single mutex whereas general purpose index structure must latch every page it traverses. The mutex is implemented as a single compare and set machine instruction (CAS).

System z and memory

As DB2 exploits memory optimization, system z continues to make the progress in expanding the memory available to system z processors.

The maximum amount of memory that can be installed in an IBM z Systems server increased significantly with the advent of the z13 in 2015, as shown in Figure 3 below. Today, a fully-configured z13 server can have up to 10TB of memory installed, with each logical partition (LPAR) able to allocate up to 4TB of memory. We expect further growth in the future processors from IBM z Systems.

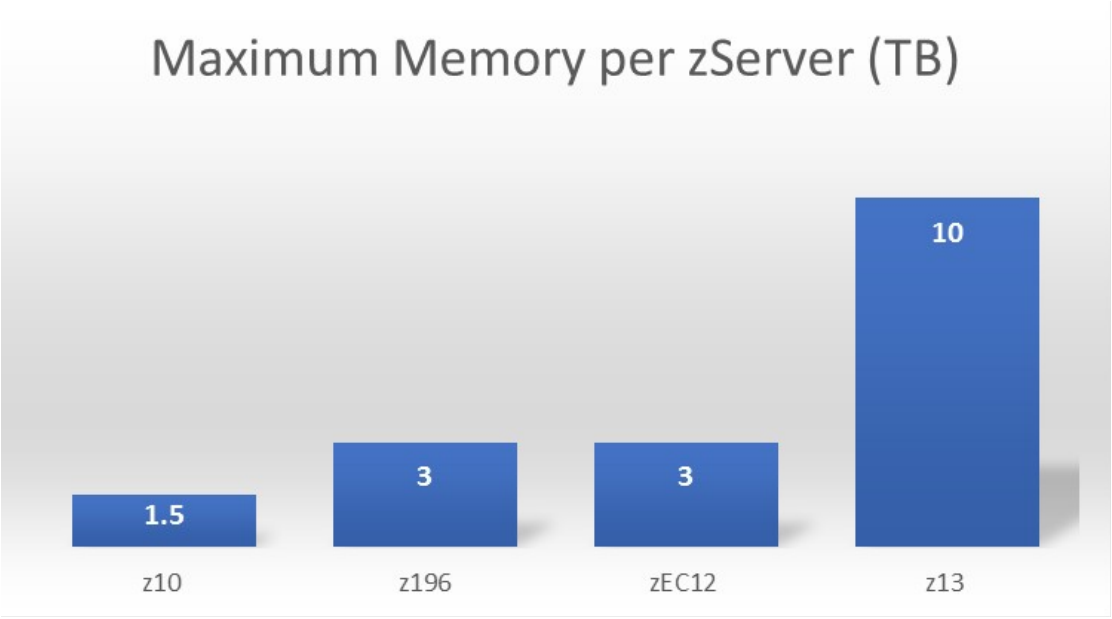

Figure 3. System z Maximum Memory per server

At the same time, the cost per GB for mainframe real storage has dropped sharply in recent iterations of IBM's z Systems servers, particularly with the latest IBM z13.

Because of these changes, today's typical z Systems server configurations have significantly more memory than before. With easier and cheaper access to large amounts of memory, Index in memory optimization (Fast index traverse) in DB2 12 makes a great synergy.

How the Index In-Memory Optimization (Fast Index Traversal) works

Not every index is a good candidate for the in-memory optimization. The candidates are unique indexes which are accessed through direct index access, as opposed to sequential index access. This could have posed a challenge to the end-user to identify the right candidates. Instead of burdening the end-user, DB2 12 provides the autonomic selection for the candidate indexes for fast index traversal.

Candidate index selection

DB2 monitors the usage of indexes and automatically builds a Fast Traverse Block for an index with many random traversal accesses. It avoids building FTBs or effectively removing FTBs if the indexes are primarily used for sequential access, or if the indexes frequently go through index structure modifications. This autonomic selection of FTB candidates relieves the burden of data base administrators and utilizes the feature only when it provides the best outcome based on runtime decision.

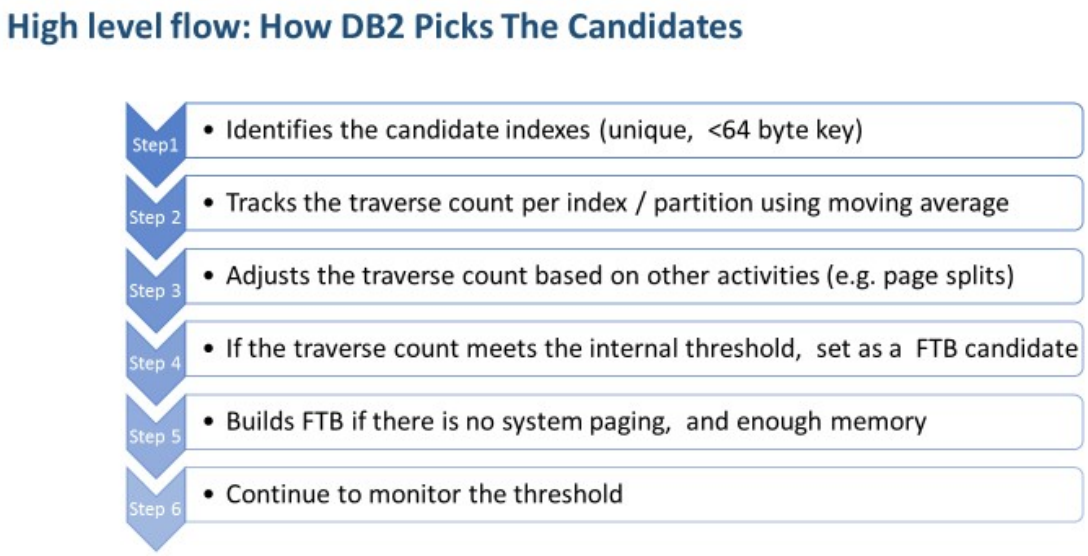

Figure 4. How FTB candidates are picked

The monitoring is done through system tasks against the indexes at each subsystem or member.

Candidate selection pursues three goals - build FTB for frequently traversed indexes, avoid FTB for indexes that take page splits often, build FTB for better suited indexes where we cannot build FTB for all eligible indexes due to memory constraints.

The rating of indexes is implemented such as to minimize the odds of spiky behavior (thrashing) as well as keeping unused FTB for far too long.

User control via INDEX_MEMORY_CONTROL subsystem parameter

A new system parameter INDEX_MEMORY_CONTROL is introduced to control FTB usage at the subsystem level so that the user can disable the feature completely, or enable it but define the upper limit of memory usage for FTBs. The default value is AUTO, which enables index in-memory optimization and allows storage of up to 20% of the sum of allocated buffer pools to be used for FTBs.

- INDEX_MEMORY_CONTOL= 'AUTO', then DB2 will use a minimum of 10MB or 20% of the currently allocated buffer pool storage, whichever is larger.
- INDEX_MEMORY_CONTOL= 'DISABLE', DB2 will disable the fast index traversal feature and no FTBs will be created.
- INDEX_MEMORY_CONTOL equals an integer between 10 and 200000 (10MB~200GB), then DB2 will allocate up to the size of the specified value for fast index traversal usage.

User Control via SYSINDEXCONTROL table

A new DB2 catalog table, SYSIBM.SYSINDEXCONTROL is added for users to influence the behavior at the individual index level. Users can force or disable the fast traversal feature for specific indexes, days of the week and time windows. For example, it's possible to let the system decide whether to create and use the fast traversal feature during the day, disable it at night for all week days and force it during weekends for a specific index or even index partition.

Qualified indexes

To qualify for fast index traversal, the index must be a unique index with a key size of no more than 64 bytes. Include indexes are good candidates for fast index traversal.

Function level required

By popular demand, DB2 12 moves away from conversion mode and new function mode. DB2 12 continues to support co-existing configuration with DB2 11 under DB2 12 function level 100(V12R1M100). The Fast Index Traversal is supported under DB2 12 function level 100 and enabled as a default through system parameter. In data sharing environment, the feature is only enabled for the indexes which are not GBP-dependent in function level 100. This is because the feature introduced the new type of lock, FTB p-lock to communicate the index structure modification across the data sharing members and this lock is not recognized by DB2 11 members. Once the activation of function level 500 or above is done, the feature is available for GBP-dependent indexes.

Rebind of packages are not necessary to trigger the feature.

Performance Improvement and Monitoring

Performance improvement with index in-memory optimization depends on the access pattern, depth of index level (determined by key size and numbers of index entries), and commit frequency.

Index level and performance improvement

When we executed a loop of singleton select statements through batch jobs with infrequent commits, we observed up to a 20% reduction in CPU time. Figure 1-5 demonstrates the measured DB2 CPU time savings using indexes with different numbers of index levels. For example, when utilizing FTBs on a 5-level index, the number of index getpages per statement was reduced from 5 getpages to 1.

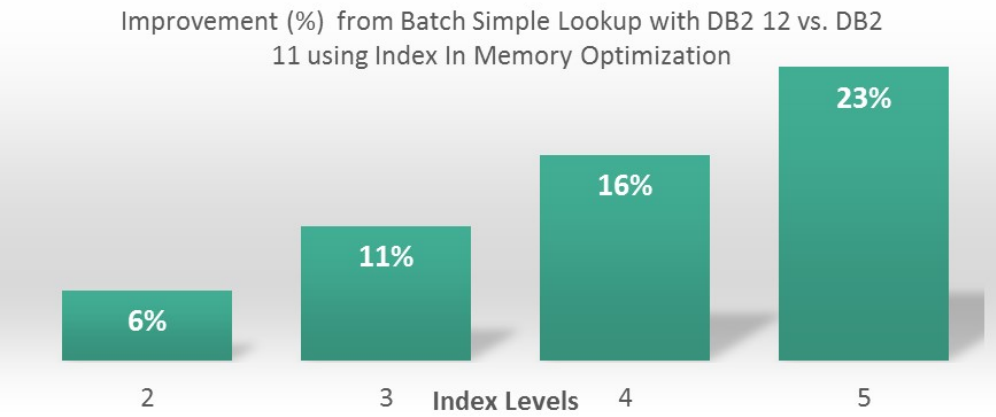

Figure 5. Index in memory optimization in batch select with different index level

Online transactions

The performance improvements with index in-memory optimization in Online Transactional Processing workloads where commits are issued more frequently demonstrate a more realistic usage. As shown in Figure 1-6, the index in-memory optimization results in a few additional percent in overall DB2 12 CPU improvement. IBM measurement indicates that Index Fast Traversal provides additional 0-4% CPU reduction in DB2 online transaction workloads.

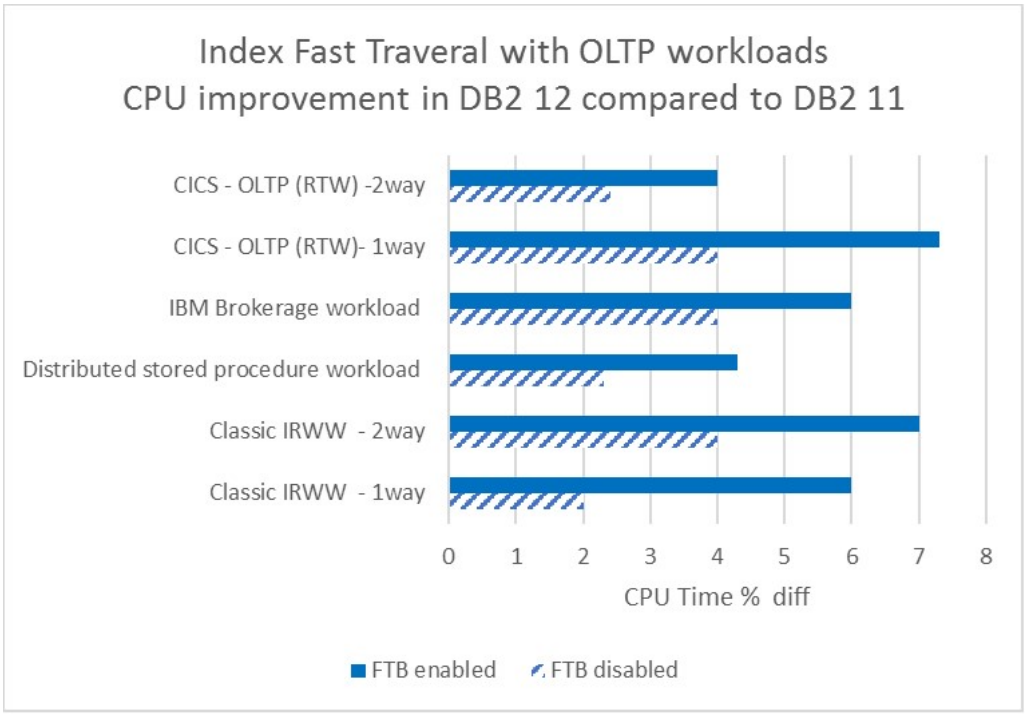

Figure 6. DB2 12 improvement over DB2 11 with and without using index in-memory optimization

The measurement data below in Figure 7 describes more details in our IBM Relational Warehouse Workload. The transactions in this workload are mostly very short running with primary key lookups for select, insert, update or delete. As shown, the number of getpages per transaction has greatly reduced

from the buffer pools assigned for indexes when we enabled Fast Index Traversal feature. Note there were no access path changes in this workload between DB2 11 and 12. The significant getpage reduction from indexes leads to the additional CPU reduction.

The upcoming Redbook, DB2 12 for z/OS Performance Topics will contain more information on performance evaluation.

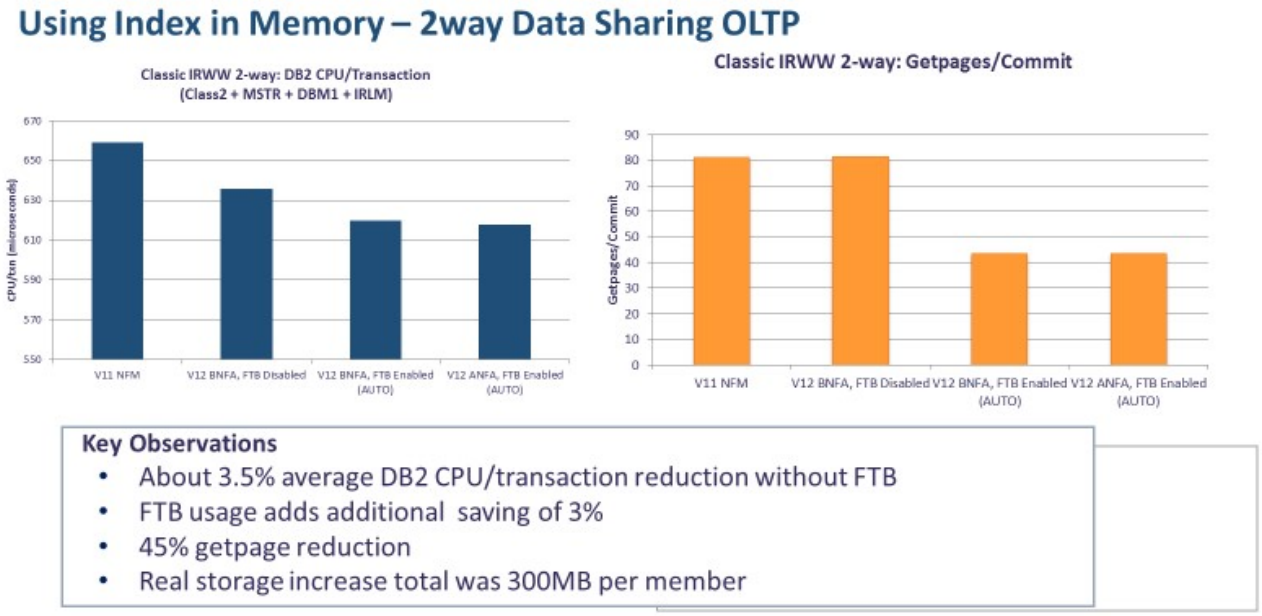

Figure 7. IBM Relational Warehouse Workload with Fast Index Traversal

Monitoring Fast Index Traversal

A few IFCIDs and a DB2 display command are provided to monitor FTB behavior.

Instrumentations

Two new IFCID trace records are introduced to track fast index traversal usage. IFCID 389 is part of statistics trace class 8. It records all indexes that use fast index traversal in the system. IFCID 477 is part of performance trace class 4 and records the allocation and deallocation activities of FTBs for fast index traversal.

New fields were added to IFCID 2 (statistics record):

- Current threshold for FTB creation
- Number of FTB candidates
- Total size allocated for all FTBs
- Number of FTBs currently allocated
- Number of objects that meet the criteria for FTB creation.

Console messages

A new DSNI070I console message is also introduced. This message contains information about fast index traversal memory usage, the number of indexes using fast index traversal and the number of candidate indexes. This message is currently only issued if the fast index traversal usage changes. The format of DSNI070I is:

```
DSNI070I subsystem-id FAST INDEX TRAVERSAL STATUS: storage-size MB, USED BY: objects-count
```

For example,

```
DSNI070I -DB2A FAST INDEX TRAVERSAL STATUS: 2 MB,
          - USED BY: 1 OBJECTS, CANDIDATE OBJECTS: 1
```

DISPLAY command

DB2 12 also introduces a new -DISPLAY STATS command to take an instantaneous snapshot of all the index objects that are currently using the fast index traversal feature. This command displays the number of index levels and the amount of storage used by the index page set/partition in the fast index traversal storage pool. A typical command is:

```
 -DIS STATS(IMU) LIMIT(*) or
 -DISPLAY STATS(INDEXMEMORYUSAGE)
```

For example,

```
DBID PSID DBNAME   CREATOR    INDEXNAME      LEVEL PART  SIZE(KB)
---- ---- -------- ---------- -------------- ----- ----- --------
0258 0201 TPCEA100 USRT031    TEWIIX1        0004  00001 00003705
0258 0179 TPCEA100 USRT031    TEHSIX1        0004  00001 00001846
0258 0063 TPCEA100 USRT031    TETHIX1        0003  00970 00000932
0258 0077 TPCEA100 USRT031    TEHHIX1        0003  00777 00000645
```

Consideration in Data Sharing environment

In data sharing environment, when the index structure modification such as index leaf page split or leaf page consolidation, is needed for indexes with FTBs, DB2 uses FTB p-locks and IRLM notify to communicate to other members. With heavy insert environment, this can become noticeable overhead. In IBM measurements, we observed a few percent of overhead if FTBs are created on the indexes with heavy index splits. While DB2 will discourage to create FTBs on the indexes with heavy index splits or consolidation, there can be a learning period. In such cases, users should be aware of the possible overhead and consider using SYSIBM.SYSINDEXCONTROL table to disable the creation of FTBs for the unique indexes which are heavily updated by multiple members.

ESP customers experience

The customers who had participated in DB2 12 ESP program had evaluated Fast Index Traversal feature. Most customers saw additional CPU reduction in their online transactions when the feature was enabled. The reduction varies in the range of 2-3% up to 10%. This matches with IBM measurements.

A few customers found it is challenging to create FTB structures at will for their evaluations. In the most cases, the reasons were identified as "not enough traverse counters" to trigger FTB creation when the index access was sequential or skip sequential. Once the access pattern from the applications was modified, they were able to enable the feature for the targeted indexes. Some other cases were due to system paging as FTB creation will be blocked if DB2 detects the paging on the LPAR.

During ESP period, customers and IBM internal testing exposed the following problems with the features. The corresponding APARs are all resolved and available. At the time of this article, there are no major pending problems associated with Fast Index Traversal feature.

Important APARs exposed during ESP by customers and IBM

- PI72018 ABEND 04E RC00C90206 DSNIDIFS WHEN DELETE RECORDS
- PI73564 FTB IS BUILT INSPITE OF IT IS DISABLED IN SYSINDEXCONTROL TABLE
- PI69873 0C4-00000038,LOC=DSNIDM .DSNKFTOB+029FC, compile error
- PI69156 Performance impact of double p-locking
- PI71639 ABND=04E-00C90101 WHEN INSERTING DATA

Summary

The index in-memory optimization - fast index traversal, targets performance improvement of a simple online index look up by utilizing a new in-memory index structure called Fast Traverse Blocks.

While DB2 adds various instrumentations and display command, the key design point of the feature is transparency from the users. It is enabled by default and designed to take advantage of larger memory automatically by monitoring index access pattern.

The feature provides CPU reduction by avoiding get page and release page operations from buffer pool for index non-leaf pages. The CPU reduction varies depending on the workloads, such as index access pattern, level of indexes.

Released: July 26, 2017 10:38 PM | **Updated**: July 26, 2017 10:38 PM

Keywords: DB2 for z/OS (p/bl/kw/kt=1&kw=DB2%20for%20z%2FOS) | Index (p/bl/kw/kt=2&kw=Index) | New Release DB2 (p/bl/kw/kt=2&kw=New%20Release%20DB2)

Recent Stories

Db2 for z/OS – New Encryption Capabilities to Protect Your Data (p/bl/ar/blogaid=910)

Updated 6:04AM CDT, Wed Sep 11th, 2019

(p/bl/ar/blogaid=910) With many high-profile data security breaches, data encryption has become a focus item up to the CIO and CEO levels. Any loss or exposure of sensitive data might cause irreparable damage to the busine...

Transparent Data Encryption (p/bl/ar/blogaid=902)

Updated 6:04AM CDT, Wed Sep 4th, 2019

(p/bl/ar/blogaid=902) There have been several high visibility data exposures in the news over the last year or so. As a Db2 DBA, one of your responsibilities is the protection of the data your business needs to run. Your a...

August Content Recap (p/bl/ar/blogaid=907)

Updated 10:58AM CDT, Tue Sep 3rd, 2019

(p/bl/ar/blogaid=907) Recap of IDUG content for the month of August 2019.

International Db2 Users Group
2601 Iron Gate Drive, #101
Wilmington, NC 28412 (https://maps.google.com/?q=2601+Iron+Gate+Drive,+%23101+Wilmington,+NC+28412&entry=gmail&source=g)
Phone: (910) 660-8649 (tel:+19106608649) / Fax: (910) 523-5504 (tel:+19105235504)

(https://www.facebook.com/IDUGDB2) (https://twitter.com/IDUGDB2)
(https://www.linkedin.com/grp/home?gid=46747) (https://www.youtube.com/user/IDUGDB2)