DB2 and Memory Exploitation
Part 2

Fabio Massimo Ottaviani – EPV Technologies

May 2016

4 Global Dynamic Statement Cache

In recent years the use of dynamic SQL has grown dramatically in the DB2 world.

One of the most important characteristics of dynamic SQL is that for a dynamic SQL statement to run, DB2 has to determine an access path and build the skeleton control structure (SKDS). This is done through the PREPARE statement.

The PREPARE process is generally long and consumes CPU so it can be an important issue when the amount of dynamic SQL is very high.

A dynamic statement has to be prepared the first time it runs, there's no way to avoid that, but the prepared statement can be re-used in all subsequent executions if it can be saved in somewhere. The Global Dynamic Statement Cache (GDSC) was introduced in DB2 V5 for this purpose.
The GDSC has to be enabled by setting ZPARM CACHEDYN to YES while the size of the cache is specified with ZPARM EDMSTMTC.

The size of GDSC, limited for years because of 31bit virtual storage constraints, can be now very large but at many sites it is still set too small to allow a full exploitation of its potential.

When a statement has to be prepared DB2 will search for the statement in the GDSC; the following situations may occur:

- the same statement is found; it will be copied to local thread storage; this is called Short Prepare; as you can imagine it requires a lot less time and effort;
- the same statement is not found; the complete prepare process has to be run; this is called Full Prepare.

GLOBAL DYN CACHE ACTIVITY SWITCH

BY HOUR Fri, 29 Apr 2016

PREPARE ACTIVITY	8	9	10	11	12	13	14	15	16	17
SHORT PREPARE	349.382	1.249.503	733.653	797.095	758.776	517.438	433.905	723.482	587.085	303.178
FULL PREPARE	55.172	87.289	86.587	98.801	116.330	136.607	47.351	92.561	171.734	81.245
TOTAL PREPARE	404.554	1.336.792	820.240	895.896	875.106	654.045	481.256	816.043	758.819	384.423

Figure 5

In the example in Figure 5 you can appreciate the benefits provided by the GDSC. As you can see a lot of Full Prepare processing has been avoided. The Short Prepare number is generally much higher than that for the Full Prepare.

All the metrics used in Figure 5 are provided by the DB2 statistics trace and collected in IFCID 2 (SMF 100). They are[1]:

FIELD	DESCRIPTION	SOURCE
SHORT PREPARE	Total number of short PREPARE.	QISDSG - QISEDSI
FULL PREPARE	Total number of full PREPARE (see Note 1).	QISEDSI
TOTAL PREPARE	Total number of PREPARE.	QISEDSG

Figure 6

The hit ratio can be estimated by dividing the number of Short Prepare by the total number of Prepare.

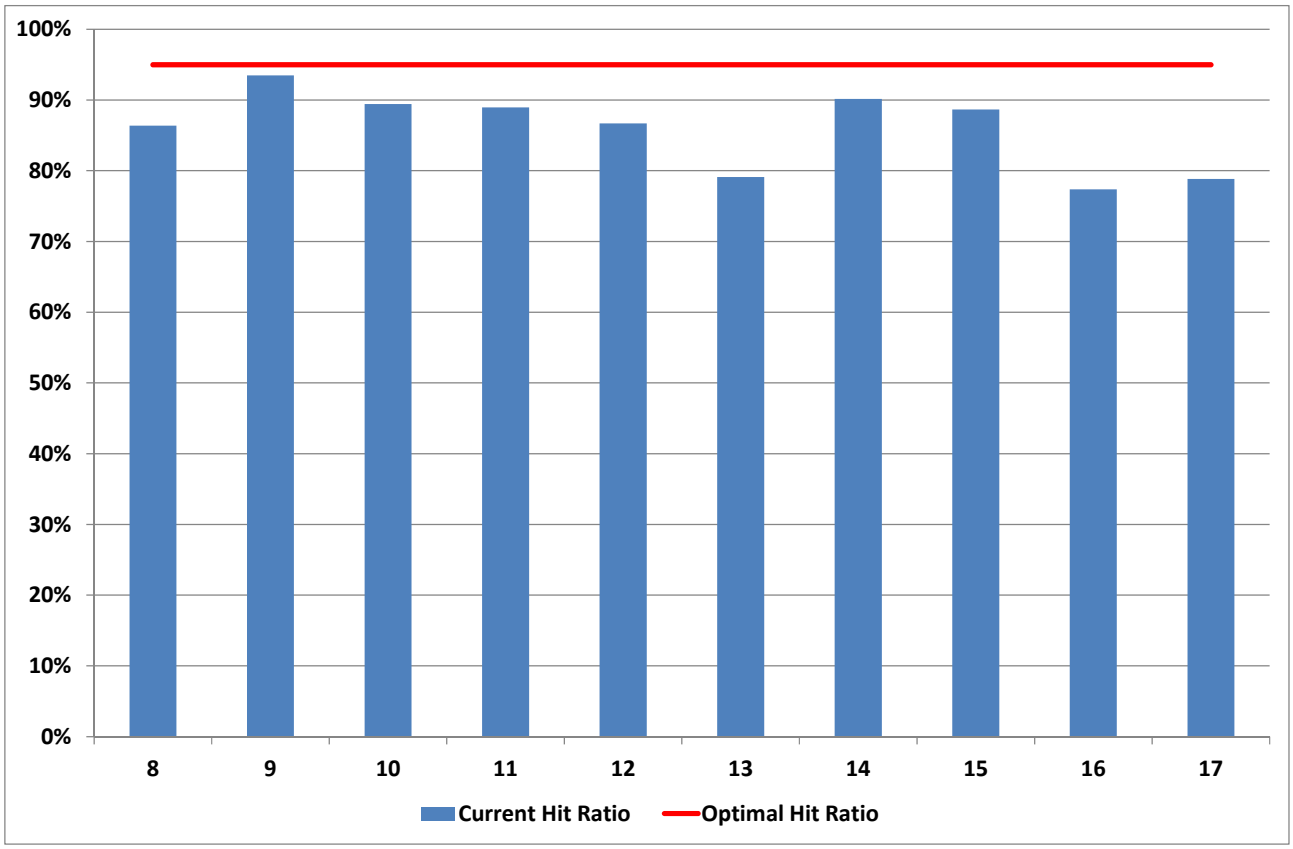

Figure 7

The GDSC hit ratio is reported in Figure 7. The red line, set at 95%, represents a commonly used best practice for the GDSC hit ratio minimum value. As you can see it is never reached.

[1] From the EPV for DB2 Help System

In this case a possible reason is that the GDSC is too small. It has a size of only 40.000 pages and it is fully used, so the suggestion would be to double it (it will require only 160 MB of storage) and analyse the effects on the hit ratio.

However in some situations the main issue is the statement format not the size of the GDSC.
For a statement in the GDSC to be re-used it should be exactly the same to the smallest detail (including literal values, spaces, order, authorization id, bind options)[2].

Generally the use of literals is the most likely reason why a statement in the GDSC can't be re-used. The best solution should be using parameter markers instead of literals but it will require modifying the application.
A possible alternative is the CSWL (Concentrate Statement With Literals) PREPARE parameter introduced with DB2 V10. When using CSWL the statement is stored into the cache and each literal is replaced with &[3].

For example:

SELECT COL1, COL2, COL3 from Table WHERE COL1 = 'A'

is stored in cache as:

SELECT COL1, COL2, COL3 from Table WHERE COL1 = &

If the following subsequent statement has to be prepared:

SELECT COL1, COL2, COL3 from Table WHERE COL1 = 'B'

DB2 will find a statement in the dynamic statement cache containing the replaced character (&) and will re-use it.

CSWL doesn't need more storage conversely it can reduce the storage wasted by statements which differ only in the literals.

5 Buffer Pools

Buffer pools are a key element in DB2 application performance.
Their goal is to allow the application to get data and index pages from storage without the need of I/O operations.
Tuning buffer pools is a very complex topic and it will not be completely discussed here.
In this paper we will only talk about the possibility to improve application performance by increasing the buffer pool size and exploiting buffer pool long term page fixing.

When an application issues a getpage request, the page is searched for in the buffer pool correspondent to the tablespace or indexspace where the page resides. If the page is not found (buffer pool miss) the application is suspended and a synchronous I/O is needed to load that page into the buffer pool.

[2] See DB2 V11 for z/OS Managing Performance.
[3] This process doesn't occur if the statement is coded with the "?" parameter marker.

In SMF 101 IFCID 3 the following metrics are provided:

- QWACAWTI; total wait time for synchronous read;
- QWACARNE; total wait events for synchronous read

QWACAWTI tells you what the is impact of a buffer pool miss on application performance.
By dividing QWACAWTI by QWACARNE you can evaluate the average response time for synchronous read operations in order to understand if there are I/O performance issues.

Another important characteristic of synchronous read operations is that the CPU consumed to execute the I/O is charged to the application and it is standard CPU time, not zIIP time as it happens for asynchronous read operations charged to the DBM1 address space.
A recent IBM study estimated 35 CPU microseconds (on a 2827-712) per DB2 synchronous read. Based on that you can estimate about 40 MIPS used every 1.000 read per second.[4]

In the next figure we show an example of very high read rate to a couple of buffer pools on a two member data sharing group.[5]

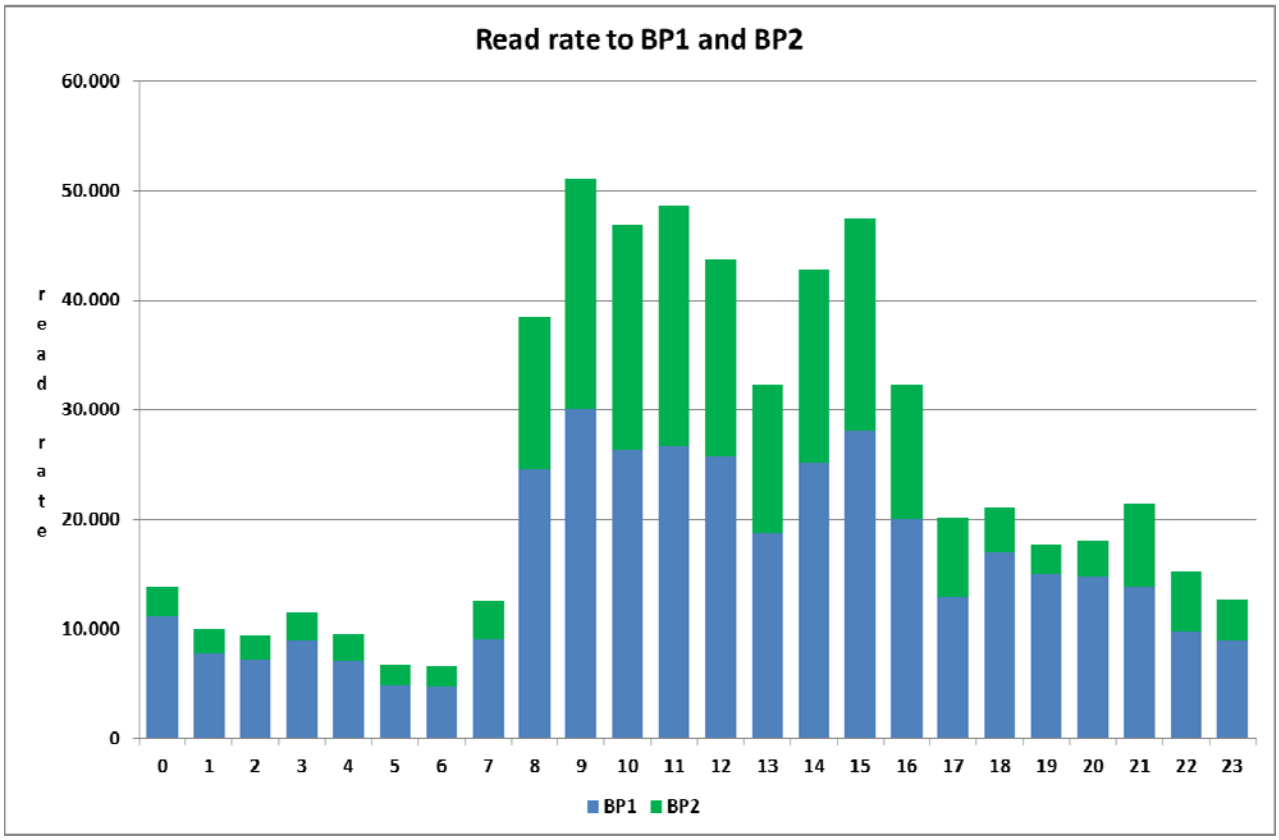

Figure 8

The size of BP1 and BP2 was about 200.000 page each on both members.
After showing this graph to the DB2 system manager he decided to double the size of both buffer pools in both members "investing" about 3,2 GB of storage.

[4] "System z: Advantages of Configuring More Memory for DB2 Buffer Pools" V2.0 - February 2015
[5] The metric used in the graph is QBSTRIO provided in SMF 100 IFCID 2.

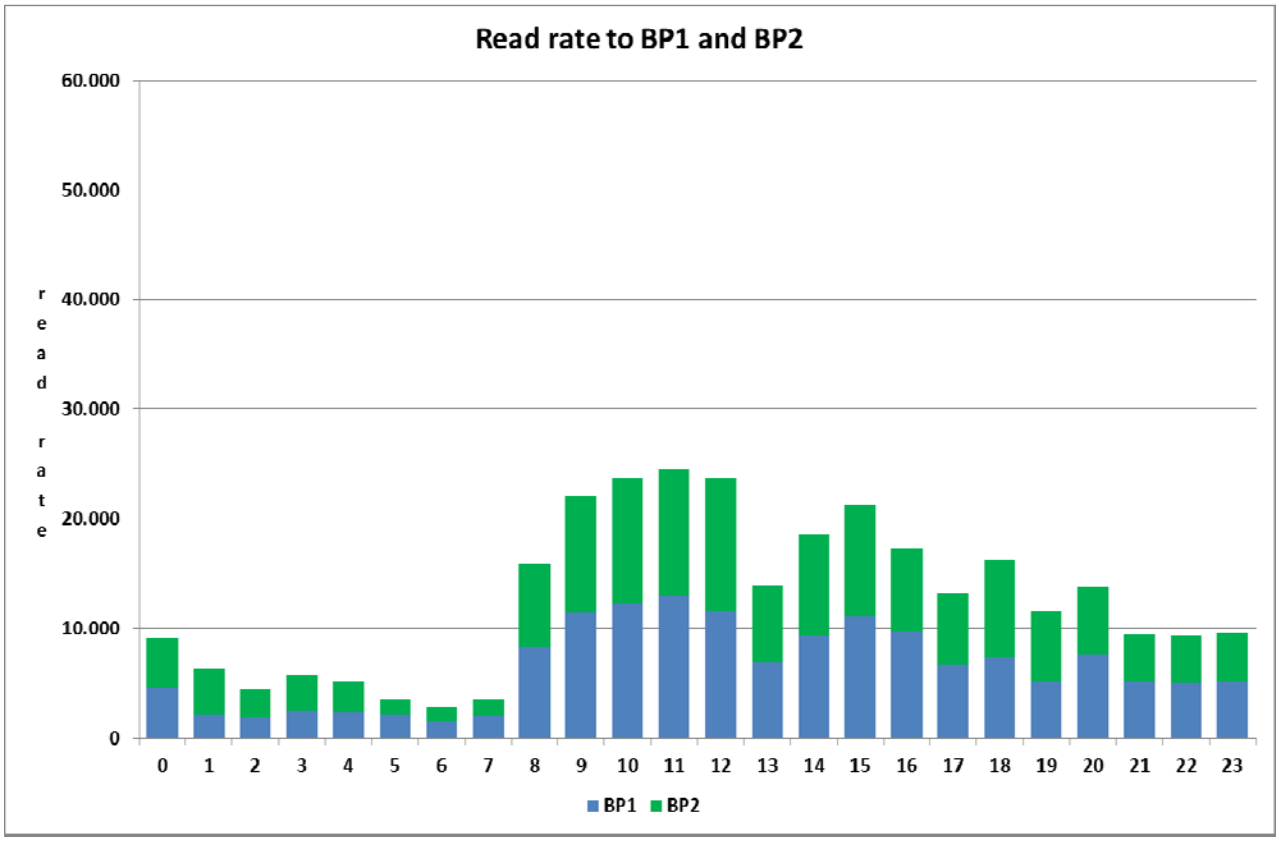

Figure 9

The result is reported in Figure 9.
At least 20.000 read per second have been saved. About 50% were synchronous so the estimated saving is about 400 MIPS.

It's not guaranteed that by increasing the buffer pool size you will be able to get a significant reduction of read operations. This is sometimes due to specific workload characteristics.[6]

In this situation you can try to minimize the impact of read operations on performance by exploiting the PGFIX buffer pool parameter. If you set PGFIX=YES the buffer pool is fixed in real storage so the page fix/unfix operations are not needed anymore when performing a read to the buffer pools with significant benefits in terms of CPU usage.

If 1MB pages or 2GB pages are available in the system LFAREA (Large Frame Area), DB2 will automatically use them for the buffer pools where PGFIX=YES is specified. This will provide additional performance benefits and CPU savings due to quicker virtual address translation operations.

[6] In DB2 V11 you can use simulated buffer pools to measure how many read I/O operations can be avoided if the buffer pool size is increased.

6 Conclusions

In modern z/OS systems memory is normally abundant and often only partially used.
Many available DB2 functionalities allow for exploitation of this available memory with big benefits for application performance and reduced CPU consumption.