

DB2 Data Sharing as the High Performance Server for E-Business

By Susan Lawson, YL&A

Introduction

Let's take a look at how DB2 data sharing is being used as the server behind some of the largest volume e-business sites. Many high volume transaction sites are powered by DB2 data sharing in a parallel sysplex environment due to the processing and capacity requirements necessary to push the limits on transaction rates for web applications.

However, we need to be aware of all of the tuning opportunities in a data sharing environment so that we can get the best performance throughput possible and not solely rely on option to add additional processors when performance criteria changes or performance begins to suffer. In other words, we want to be careful not to throw hardware at a performance problem in this environment because there are limits. The real key to getting more transactions through a high volume system is to have not only the processing power and capacity to do so, but also having this environment tuned to handle large workloads. In this paper we are going to take a detailed look of some of the performance tuning issues in a data sharing environment.

I spoke with a few customers who are driving over a thousand transactions a second through a data sharing sysplex and the most common areas for tuning this type of load is what this paper will focus on.

SQL and Applications

Rumour has it that we do not have to make any application changes when we move to data sharing – but let's be real about this. We must revisit the applications that are moving to this environment because from a performance standpoint there are good applications and those that can kill a sysplex. We must be smart about how we design and deploy our web applications in a data sharing environment if we are going to push the transaction volume to the maximum.

Physical Design for Performance

The web is open 24 hours a day 7 days a week all over the world. When we open our business to the world via the Internet we also open ourselves to new availability issues. We have to find a way to 'hide' outages, as we know 24 X 7 is truly impossible to guarantee. Data sharing is very helpful for doing this, as one of the key benefits to this environment is that we can do just that – hide outages (planned and unplanned). But the key here is that we must design for it – just moving to a data sharing environment does not guarantee availability. We will take a look at a few of these issues.

Tablespace Partitioning

The use of tablespace partitioning can yield several benefits in a data sharing environment. For instance, you could partition a tablespace and through affinity routing control access to the partitions by member to avoid overhead of lock propagation and group bufferpool dependency

for partitions that otherwise would have R/W interest from several members. You can also use partitions to help with utility processing, such as a LOAD or REORG. Utilities such as LOAD and REORG can take advantage of partitioning. For instance, if you split a table into 10 partitions you could run 5 concurrent LOADs on two members, however keep in mind that any NPIs defined on the table will become GBP dependent during this process – not a good idea, especially if you have large NPIs (some are in the 200+ gig range now days). If these are large NPIs this could be a problem and it would be best to drop the NPIs before the LOAD and re-create them afterwards.

Selective Partition Locking

Selective Partition Locking (SPL) gives us the ability to tell DB2 only to acquire locks at a partition level and not to escalate locks to the tablespace level. This will allow for less propagation of locks to the coupling facility and better concurrency for application accessing data in various partitions. This can be especially useful if usage against the partitions is spread over multiple members using affinity routing. Performance trace class 6 (IFCID 0020) will show if you are truly using SPL on your tablespace partitions or you can use the `–DISPLAY DATABASE(LOCKS)` command to display information about SPL as well.

However, selective partition locking will only help in a non-data sharing environment if an agent actually escalates. If you are not having escalation problems then you may not want consider using SPL. Without SPL, the parent intent lock is taken on the last partition, regardless of which ones you access. These locks are almost always intent locks and therefore almost never cause a problem. With SPL, the parent intent lock is taken on whichever part you access. If escalation occurs then with SPL the lock escalation occurs just for the part (or parts) that you hit too many locks on. Without SPL since the parent lock is only on the last one we escalate the entire pageset and access is prevented to all partitions.

Be careful with this because SPL can increase the amount XES contention if you don't have true partition independence because SPL locks all parts accessed (instead of just the last one). Therefore if an agent is accessing multiple parts it will hold more parent locks than without SPL and if other agents are doing the same the chance of XES contention is greater. That is one of the reasons not to use SPL unless you are sure that partitions are only accessed by one application or one member.

If you have affinities per partition, then SPL can buy you independence if one of the applications escalates since the escalation happens at the part level with SPL instead of the tablespace level without SPL. If using SPL you can avoid GBP dependency on the additional tablespace partitions and will lessen coupling facility overhead.

Even if you are spreading your work across multiple partitions and even using SPL you are still inserting into the one NPI and it will be group bufferpool dependent. There is not a great deal we can do about the NPI and GBP dependency. Page P-locks are taken on the index leaf pages and while a page p-lock is not expensive but the process of P-Lock negotiation of it is when two processes must negotiate locks for data that is on the same index page. You help to reduce the negotiation by not having access to the same data.

Multiple Page Sizes

There are two new page sizes of 8K and 16K are available in DB2 Version 6. Appropriate use of this can have a performance impact on data sharing, since an 8K page size only requires 1 page lock instead of the 2 that would have been required if using 4K pages. This reduces the overhead to the coupling facility since locking would effectively be cut in half. These larger page sizes also help for those wide rows required to support of certain types of warehouse applications, specifically those very wide fact tables used in some enterprise warehouses. There are still only 4k and 32k page sizes used for the work files however, which should not be a concern.

Eliminating Space Map Contention

Space map contention is another area that proved to be a performance problem as more and larger installations of data sharing were moved into production. Hot spots are often caused due to the current space management techniques used for optimum space utilization, which caused high update activity in the associated space map pages. This often resulted in a significant increase in p-lock negotiations; due to the attempt to cluster data by using the clustering index or when not present the use of the default index used for clustering. In order to provide relief for these situations, there is a new pageset type called MEMBER CLUSTER (PQ02897), which will allow for inserts to choose the insert location to avoid lock and latch contention. This is done by allowing each member in the data sharing group to identify and use a separate space map page so that insert space management can avoid acquiring page P-locks unconditionally. This option is defined by using a new MEMBER CLUSTER keyword on the CREATE TABLESPACE statement. As with many performance tuning features, there is a downside. Inserts will not cluster the data but instead DB2 chooses an insert location that minimizes lock and latch contention. Space maps have been changed to cover only 199 data pages instead of the normal 10,000 page coverage for tablespaces using the MEMBER CLUSTER option. Each member will then get a different space map page so they are inserting into different areas of the tablespace.

Space Map Page Tracking

When DB2 maintains information in the space map pages in a page set, to help optimize the performance of incremental image copies, it may be prone to causing 'hot spots'. This often occurs frequently in data sharing environments when multiple processes from more than one member are performing updates to the same space map pages. This usually causes logical contention on the space map page as well as additional page P-locking. DB2 Version 6 provides another new options for dealing with contention on space map pages. We can now specify whether or not to have DB2 track the data page changes in the space map, which identifies which pages have changed. This will generally cause incremental copies to run slower since they would now be required to scan the pageset in order to identify which pages changed. However it can provide faster application performance due to less contention on the space map pages. This new feature is called TRACKMOD YES/NO and is defined on the CREATE/ALTER TABLESPACE statement

Group Bufferpool Tuning

Depending on how your data is accessed by the application will drive how your group bufferpools are implemented. If you have not tuned your group bufferpools (via object separation according to usage and appropriate parameter tuning) your performance will suffer as you incur additional overhead for additional unnecessary processing. In a data sharing environment one key performance measurement will be measured by how well you control the caching and writing of data.

Sizing

Sizing is very important to control how well the group bufferpool is utilized. Sizing group bufferpools is not like sizing normal virtual bufferpools. The group bufferpools are defined as structures in the coupling facility. They are given an initial size when they are defined in the CFRM policy and for performance and availability reasons will need to be created in a coupling facility separate from the coupling facility that holds the Lock and SCA structures. It must also have enough room for fail-over in that other coupling facility in the event of a failure. However, this issue changes somewhat with the implementation of group bufferpool duplexing.

There are some standard rules of thumb for sizing but most are very generic at best. For the best sizing of your group bufferpools you are going to have to have a good understand of the amount of sharing that will be occurring against the objects that are in the group bufferpool. In other words, you are going to have to worry about object separation in virtual bufferpools even more so when implementing group bufferpools, otherwise your initial sizing estimates will be rather difficult.

Generic Guidelines and Rules of Thumb

There have been some published guidelines for sizing group bufferpools, however they are all so generic that they can only be considered a starting point. You have to consider so many aspects of objects in the group bufferpools that it makes any generic 'one-size-fits-all' formula almost useless. We have to consider such things as:

- 1) How many members are in the data sharing group currently that will be sharing the data in the group bufferpool, and how many are going to be added in the future?
- 2) Is there use of affinity processing in the group? If some subsystems are not going to use the data in the group bufferpool, but they still have a virtual pool that is backed by a group bufferpool, why include their virtual pool size in the sizing formula?
- 3) Were the subsystems existing and merged into the group, or are they new, clean installs? If merged, it is doubtful all virtual bufferpools are using the same separation strategies.
- 4) What about GBPCACHE NONE and SYSTEM?
- 5) Is the group bufferpool going to be GPBCACHE(NO)?
- 6) Are the virtual bufferpools sized correctly?
- 7) Size of the pages in the virtual pools (4K,8K,6K,32K)

Listed below are the general rules of thumb published for sizing group bufferpools. As you can see there are a lot of assumptions made here, so be careful if you choose to follow these rule of thumb formulas and be sure to monitor the group bufferpools and adjust this sizes accordingly.

- For objects defined GBPCACHE=CHANGED we can take the total allocation of all members' virtual pools, hiper pools and then multiple by 10% (for light sharing with little updating), 20%(for medium sharing with moderate updating) or 40% (for high amount of sharing and lots of updating).
- For objects defined with GBPCACHE=ALL we can take the total allocation of all members virtual pools ONLY and then multiply by 50%(for few data sets), 75% (for half the data sets) or 100% (when almost everything is shared).

Keep in mind that over sizing local bufferpools will not be of any benefit in a data sharing environment and if you are using GBPCACHE ALL, hiperpools will not be used.

GBP Sizing for Cached Changed/Non-Cached/LOB Data

Recently there have been some more specific formulas published for specific types of data(source: IBM Data Sharing Planning and Administration – SC26-9007-00). This take into account weather or not the data is cached, and there is also one for LOB usage. For sizing a group bufferpool where all data is cached the rules of thumb mentioned before can provide some guidelines.

Caching Changed Data

For group bufferpools caching changed data:

First calculate the estimated number of data entries by using a variable to represent the estimated degree of data sharing. Where U estimated degree of data sharing:

- 1 - high degree of data sharing high update activity
- 0.7 –moderate sharing and moderate update activity
- 0.5 – low amount of sharing low update activity

$\text{Data Entries} = \text{Degree of Data Sharing}(U) * \text{Pages written to disk per second for all members}(D) * \text{Page Residency Time in GBP in Seconds}(R)$

This will gives us an estimate of the number of data entries need to stored cache pages for a period of time suitable for the necessary rereferencing of the pages. Next, take this number times the page size (P) of the pages in the group bufferpool (4, 8, 16, 32) to determine the size of the actual data entry and divide this by 1024 (1K) to get the size in MB for the actual amount required for the data entries.

Next, we will need to determine the number directory entries required. Add up the number of data pages required for all of the hiperpools(HP) for all members plus the total number of data pages for a virtual bufferpools(VP) across all members. Take this result times the estimate degree of data sharing determined earlier (U). You add this result to the data entries. This provides us with an estimated number of pages that will need to be registered in the directory. But we still need to determine that size, so take this result times 1.1(which is additional storage required for the coupling facility control structures) and then times 0.2 (size of directory entry in

KB). Divide all of this by 1024 (1K) to get the size in MB for the size needed for the directory entries.

Now to give a size to the group bufferpool and determine the directory/data entry ratio. Add the amounts determined for the data entries and directory entries together, result is total MB required. This an estimated size for your group bufferpool. You can then take the directory entries and divide them by the data entries to get the estimated ratio.

Summary calculation:

- $U * D * R = \text{Data Entries}$
- $\text{Data Entries} * P/1024 = \text{Data Entries(MB)}$
- $\text{Data Entries} + (U * (\text{VP pgs} + \text{HP pgs})) = \text{Directory Entries}$
- $1.1 * \text{Directory Entries} * 0.2/1024 = \text{Directory Entries(MB)}$
- $\text{Data Entries(MB)} + \text{Directory Entries(MB)} = \text{Group Bufferpool(MB)}$
- $\text{Directory Entries/Data Entries} = \text{Ratio}$

Caching No Data

For group bufferpools not caching data:

First calculate the estimated number of directory entries by using a variable to represent the estimated degree of data sharing. Where U estimated degree of data sharing:

- 1 - high degree of data sharing high update activity
- 0.7 –moderate sharing and moderate update activity
- 0.5 – low amount of sharing low update activity

$\text{Directory Entries} = \text{Degree of Data Sharing}(U) * (\text{Total Number of Pages in all Virtual Pools(VP)} + \text{Total Number of Pages in all Hiperpools(HP)})$

We take the degree of data sharing and multiple it by the total number of pages in the virtual and hiperpools for all members. This gives us the total number of pages that will need to be registered in the directory. Remember, even if we do not cache data it still must be registered for cross-invalidation purposes. This will gives us an estimate of the number of directory entries need to be registered. But we still need to determine that size, so take this result times 1.1(which is additional storage required for the coupling facility control structures) and then times 0.2 (size of directory entry in KB). Divide all of this by 1024 (1K) to get the size in MB for the size needed for the directory entries. Remember the ratio is not needed here because it will be ignored.

Summary calculation:

- $U * (\text{VP pages} + \text{HP pages}) = \text{Data Entries}$

- $1.1 * \text{Directory Entries} * 0.2/1024 = \text{Directory Entries(MB)}$
- $\text{Directory Entries(MB)} = \text{Group Bufferpool(MB)}$

Caching LOB Space Maps

If plan to have a group bufferpool for LOB objects and it is defined as GBPCACHE(SYSTEM), to only cache the space map pages, there is an another formula to consider.

First calculate the estimated number of data entries by using a variable to represent the estimated degree of data sharing. Where U estimated degree of data sharing:

- 1 - high degree of data sharing high update activity
- 0.7 – moderate sharing and moderate update activity
- 0.5 – low amount of sharing low update activity

$\text{Data Entries} = (\text{Degree of Data Sharing(U)} * \text{Pages written to disk per second for all members(D)} / 10) * \text{Page Residency Time in GBP in Seconds(R)}$

When determining the pages written to disk per seconds you want to get a discount count of pages per member. You can get this count from the PAGES WRITTEN field in the Statistics report. After you determine this you will need to divide it by 10 for an estimate of the LOB system pages written per LOB data page and then take this times the degree of data sharing(U), and then multiplied by the residency time(R). This result will give you an estimate of the number of data entries need to store space map pages for a period of time suitable for any necessary rereferencing. Next, take this number times the page size (P) of the pages in the group bufferpool (4, 8, 16, 32) to determine the size of the actual data entry and divide this by 1024 (1K) to get the size in MB for the actual amount required for the data entries.

Next, in order to determine the number directory entries required, add up the number of data pages required for all of the hiperpools(HP) for all members plus the total number of data pages for a virtual bufferpools(VP) across all members. Take this result times the estimate degree of data sharing determined earlier (U). You add this result to the data entries. This provides us with an estimated number of pages that will need to be registered in the directory. But we still need to determine that size, so take this result times 1.1(which is additional storage required for the coupling facility control structures) and then times 0.2 (size of directory entry in KB). Divide all of this by 1024 (1K) to get the size in MB for the size needed for the directory entries.

Now to give a size to the group bufferpool and determine the directory/data entry ratio, which will be a little bit different for LOBs. Add the amounts determined for the data entries and directory entries together, result is total MB required. This an estimated size for your group bufferpool. You can then take the directory entries and divide them by the data entries to get the estimated ratio. However, for the ratio LOB space map pages, you can put your numbers in to determine the ratio, but you would only use that is it was less than 255. Else use 255 as the ratio.

255 would be the maximum because this is the largest numerical value that can be stored in a byte (which is probably going to have to change in the future to support the correct ratio for LOBs using GBPCACHE(SYSTEM)).

Summary calculation:

- $(U * D/10) * R = \text{Data Entries}$
- $\text{Data Entries} * P/1024 = \text{Data Entries(MB)}$
- $(U * (VP \text{ pgs} + HP \text{ pgs})) + \text{Data Entries} = \text{Directory Entries}$
- $1.1 * \text{Directory Entries} * 0.2/1024 = \text{Directory Entries(MB)}$
- $\text{Data Entries(MB)} + \text{Directory Entries(MB)} = \text{Group Bufferpool(MB)}$
- $\text{MIN}(\text{Directory Entries/Data Entries}, 255) = \text{Ratio}$

These formulas provide a much more accurate way of calculating the group bufferpool size and ratio. Still keep in mind that these do still add up all pages in all the members virtual and hiper pools. If you are using affinity processing you may not want to include all of the members pools in the calculation. Also, do not forget that when new members join the group you may have to resize.

So as you see, to be able to start to size a group bufferpool appropriately we will need to start by placing objects in separate bufferpools first based on weather they are shared or not, and then further separate them by weather you need to be caching all the pages for these objects, or caching just the changed pages, or caching no pages. Also, be sure that the bufferpool usage is the same on every subsystem in the data sharing group.

Bottom line here is that there really is no one formula for sizing these. Adherence to generic formulas, means generic performance. Choose your formula wisely, monitor and be sure to leave room in your coupling facility structure definitions for growth.

GBP Tuning

First of all, if you are not implementing good local(virtual) bufferpool standards for your local bufferpools, then good luck tuning your GBPs because the same rules apply! You need to have a clear separation of bufferpools by the type of data and the usage of the data because tuning the size and the parameters of the GBPs will depend on these factors, same as sizing and tuning virtual bufferpools do. It is recommended that virtual bufferpools on all subsystems in the data sharing group have the same characteristics (size and thresholds).

Ratio

The ratio is a setting on the group bufferpool that establishes the number of directory entries to the number of data entries in the GBP. If we do not have enough directory entries (entry for each page read on any DB2 member – only one page registered regardless of number of members with interest), then when a new page needs to be registered a directory will be reclaimed in order to register a new page. This will then cause process requiring that page to have to go to disk to reread and register the page. Depending on the number of time this occurs it can add up to significant overhead. Use the `-DISPLAY GROUPBUFFERPOOL` command to determine how

many times this occurs. IFCID 0255 can also be used to determine the number occurrences of a buffer refresh caused by cross-invalidation of a data page in the GBP to help you get an idea of the amount of read/updates occurring among the members.

Castout and Checkpoint

Since there is no connection between the coupling facility and disk, DB2 must have a way to move changed pages out to disk. It does this through a process called castout. The castout process(performed by castout engines) moves the changed pages from the group bufferpool through a private area in the DBM1 address space (NOT a virtual bufferpool) and from there they are written to disk.

The castout process is triggered when the number of changed pages exceeds the CLASST threshold, or the number of changed pages exceeds GBPOOLT threshold, or a psuedo/physical close performed on a dataset by the last updating member. Think of the CLASST threshold the same as the VDWQT threshold, on local bufferpools, and the GBPOOLT as the DWQT threshold.

CLASST

The CLASST is monitored by the castout owner for a particular pageset that is assigned to a particular changed class queue. Pages are assigned to class queues and are generally kept together by tablespace and indexspace. There are 1024 changed class queues used and they are managed in LRU order. When the number of changed pages in a class reaches the CLASST threshold (this is the percentage of changed pages in the group bufferpool) the pages are castout to disk. The default for CLASST is 10% and you want to drive your castout processes by this threshold as much as possible to keep the writes constant. When the threshold is hit, the castout owner will begin to castout the pages. This threshold is there to keep a particular pageset from monopolizing the group bufferpool.

GBPOOLT

The GBPOOLT threshold is monitored by the structure owner for the GBP and when this threshold is reached the pages are castout until a reverse threshold of 10% is reached. The default for the GBPOOLT is 50%. The first may not be bad in the majority of cases, but a default of 50% is probably not suitable unless the objects in that bufferpool have a lot of rereference. If you allow the GBPOOLT threshold to control all writes you may run into situations where you could run out of write engines. You can use IFCID 0262 to get summary stats for each time the GBPOOLT is reached. Use this to monitor how efficiently the threshold is handling work. Use IFCID 0263 to get summary stats for the castouts done by the page set and partition castout owners.

GBPCHECKPOINT

The group bufferpool checkpoint is the time in which all changed pages are castout to disk and an LRSN(Log Record Sequence Number) is recorded in the log. When a checkpoint is reached it is the responsibility of the castout owners to begin to cast pages out to disk. The default for this is 8 minutes but will need to be adjusted, per group bufferpool, depending on the usage and rereference of the data in the group bufferpool. Remember you will have to adjust this in

conjunction with the CLASST and GBPOOLT parameters who are also driving the castout process. You can use IFCID 0261 to obtain additional information about GBP checkpoints.

If checkpoints are not advancing the LRSN fast enough you would want to decrease the interval. You can determine how fast the checkpoint is advancing by issuing the `-DIS GROUPBUFFERPOOL` command. This will show the LRSN of the last checkpoint taken. The best set interval is one that achieves balance between the performance impact of frequent group bufferpool checkpoints and the impact to recovery. The lower the checkpoint interval the higher the resource consumption due to more frequent checkpoints. The higher the checkpoint interval the slower the recovery from a group bufferpool failure due the extended period of time between checkpoints.

Castout thresholds and group bufferpool checkpoint interval are two areas for tuning with our group bufferpools that are similar to tuning out virtual bufferpools. Tuning these thresholds will depend on the referencing rates of the data by the application.

Forcing Castout Ownership

Here is a quick thought on controlling castout performance. If you are very high performance situations and doing a good deal affinity processing (not using WLM Workload Manager), you may be able to drive which DB2 will become responsible for the castout process. Castout owners can be driven thus ‘forcing’ less active member to become castout owner. You may want to do this to get Better castout performance by driving it though a subsystem that has less I/O activity occurring on that subsystem. Remember, the castout process drive the changed pages through the DBM1 address space before it gets out to disk, so you may not want castout to be occurring on a subsystem where the DBM1 address space is already constrained. It is best to try to control by forcing structure owner. You would want to force ownership to a member by having the member use(first allocate) the group bufferpools and then you would use this member to drive I/O to disk. This is not an easy task to accomplish and guarantee success at all times, so be careful when attempting to accomplish this.

GBPCACHE Option

GBPCACHE(ALL) can be used to cache all pages in the group bufferpool even if they are not changed. The only reason you would ever want to do this is when you have some very ‘hot’ pages that would benefit from remaining in the GBP. Also, if you are using 3390 disk cache there will less overhead for reading a page from disk than from the GBP. For example, reading 32 pages from a GBP is more CPU overhead than the CPU overhead to schedule a single read I/O request to read 32 contiguous pages from disk, therefore this would out perform GBPCACHE(ALL) and would not require as large of a GBP. GBPCACHE(CHANGED) is the default is suitable for the majority of situations. This option will only cached the changed pages in the GBP, however clean pages will still be registered in the GBP for cross invalidation purposes (you will have directory entries, but no data entries).

With DB2 Version 6 there are now a few new options for GBP caching. GBPCACHE (NONE) will allow us to only use the GBP for cross invalidation purpose, with no data pages being cached and all changes written directly to disk. This will have minimal usage but will be very useful in a high insert, low re-reference environment where it is not necessary to keep the updated pages resident in the GBP. The other new option of GBPCACHE (SYSTEM) allows for only the space map pages to be group bufferpool dependent and all other pages are written

directly to disk. This option is only applicable to LOBs (Large Binary Objects – introduced in V6) because we do not want these extremely large objects to be using the coupling facility. Both of these options will help with performance in the coupling facility but of course the trade-off is the overhead associated with the immediate writes and reads to/from disk vs. the group bufferpool. However, GBPCACHE(NONE) is not recommended for availability purposes. A better option for availability would be to use group bufferpool duplexing. The dynamic inter-system interest tracking has been recently enhanced to help lessen the need for this option to be used in the case of heavy INSERT/low re-reference.

You can also change GBPCACHE option through Alter at group bufferpool level. The GBPCACHE(YES) option on the group bufferpool means that the bufferpool will be used as usual and the value specified in the pageset/partition level GBPCACHE option will be used for caching. If the group bufferpool is defined as GBPCACHE(NO) then all changed pages will be written to disk, not to the GBP. Damage assessment is avoided and no data will be needed during a recovery. This is less disruptive to change this than it is to change the page set level attribute. It will take precedence over GBPCACHE option on page set. You could use to plan for using different types of processing at different times of day, there may be times when caching is not necessary. It is recommended that you put objects that do not require caching into a group bufferpool defined with GBPCACHE(NO) and not mix and match pagesets with different caching options defined (say some pageset with GBPCACHE(CHANGED) and some with GBPCACHE(NONE) because this will make appropriate sizing very difficult and inaccurate. If a group bufferpool is defined with GBPCACHE(NO) the ratio is ignored because there is no storage of data.

So the question to be answered is ‘When to cache?’. There is no need to cache an updated page is rarely referenced, for example a batch job sequentially updating a large table. We can see a performance benefit for the application in these cases. By not caching cost of transferring data to GBP and casting out from GBP is saved which in turn saves cost of synchronous I/O to disk during a commit. Additional overhead reduction can also be achieved by then reducing amount of synchronous disk I/O at commit by lowering the DWQT threshold on the VBP so DB2 would write more page asynchronously prior to the commit. To help you determine if you should be caching at all you can look at group bufferpool statistics see if you would benefit from GBPCACHE(NO). If the ratio of READS, DATA RETURNED/PAGES WRITTEN is <1% then you may benefit from using GBPCACHE(NO) or maybe GBPCACHE(NONE) on GBP level.

Group Bufferpool Dependency

Group bufferpool dependency of a dataset occurs if a dataset is open for read access by one or more DB2 member and open by at least one member for read/write access. It can also be present if there are changed pages in the group bufferpool that have not been cast out. The reason we need to be concerned with group bufferpool dependency is because it has impacts on data sharing overhead because the amount of activity in the group bufferpool will be driven by the amount of datasets that are group bufferpool dependent and also this dependency can drive additional lock propagation to the coupling facility.

In order to determine if P-Locks are being held and/or if a dataset is group bufferpool dependent you have two options. You can issue the –DISPLAY BUFFERPOOL command and it will show if a pageset is GBP dependent (Version 6 only) or you can issue a –DISPLAY DATABASE LOCKS command to view if P-locks are being held.

PCLOSEN/PCLOSET

When a dataset moves from read/only to read/write, or read/write to read/only, it is moving in and out of the group bufferpool (effects whether or not the dataset is GBP dependent). This movement is performed during:

- Pseudo Open - occurs when a page set is first physically updated, after it was already opened in a read only state
- Pseudo Close – occurs when a page set has not been updated for a period of time(PCLOSEN/PCLOSET) Interest goes from read/write to read only

This process is used to narrow the ranges of log records necessary for recovery of a pageset and is partially controlled by the PCLOSEN and PCLOSET DSNZPARMS. You do not want these parameters so low that they are continually driving datasets in and out of the group bufferpool constantly. It is recommended that you set the PCLOSEN ZPARM(number of checkpoint between write activity before a dataset is closed) very high a basically disable it, thus allowing you to control dataset closure through the PCLOSET ZPARM(amount of time between write activities before the dataset is closed). You can set this at 15 –20 minutes and then monitor the number of dataset closures to observe the number of closes and opens. This can cause excessive overhead. Improvements were made in Version 6 to help with a reduction in elapsed time for open/close activity for datasets (approximately 20% according to initial testing by IBM – SG24-5351-00). This came about with DFSMS/MVS 1.5 enhanced catalog sharing which will allow for MVS catalog information to be stored in the coupling facility so that it can be shared among the members in the sysplex.

Duplexing

The removal of unplanned outages is always the ultimate goal, both in our designs of systems and within the DBMS component as well. A group buffer pool recovery can take significant amount of time since DB2 has to recover data from the logs. In order to eliminate this single point of failure in data sharing, group buffer pools can now be duplexed, implementing the strategy of a hot backup. This feature allows the same buffer pool to reside in two different coupling facilities. Any changes made to the primary group buffer pool will be reflected in the backup, and the backup can take over as the primary if the primary fails without causing an outage. The backup group buffer pool only receives changed pages that are written to the primary. Thus read operations are not affected by duplexing. When a page is updated, it is written asynchronously to the backup and synchronously to the primary. This way the writes are overlapped for better performance. Also when castouts occur, they are only written to disk from the primary group buffer pool. After the castout is complete, the pages are simply deleted from the backup.

Locking and Concurrency

DB2 protects the consistency of data among the members in the group is protected via concurrency and coherency controls. In order to provide concurrency controls among the DB2 members in the data sharing group, a new locking structure is used. Since tuning locking is

extremely critical to performance in a data sharing environment it is important to understand how it works.

Explicit Hierarchical Locking

Data Sharing locking is not like locking before. With data sharing we introduce Explicit Hierarchical Locking (EHL). If you were not aware, prior to data sharing you were using Implicit Hierarchical Locking. The only difference is that with EHL a token is kept that identifies parent/child relationships. A parent lock is a tablespace/partition lock and a child lock is a page/row lock. The benefit of using EHL is the fact that only the most restrictive lock moves to the coupling facility reducing the number calls to coupling facility to control concurrency, which can create a great deal of overhead. Lock avoidance still works with EHL and type 2 indexes work best. This does not lessen the fact that the use of uncommitted read (UR) should still be considered wherever possible.

With EHL only the most restrictive parent lock is propagated until the time in which it is necessary for the child to be propagated(recorded in the coupling facility), thus lessening the amount of lock activity.

Here is an example of how EHL works: If parent lock is a tablespace in IX(Intent Exclusive) mode, then a child lock in S(Share) mode on a page would not have to be propagated to the coupling facility lock structure because the lock on the parent is more restrictive.

Basically we lock only what is necessary, and negotiate locks in CF if there is conflict. Child locks will get propagated only if the parent locks are in conflict.

Lock Management

Locking gets a little more complicated in data sharing – and more critical in terms of performance tuning. There are more lock types to worry about. With thousands of transactions a second coming in via heavily hit web applications we cannot afford to be waiting on locks, taking excessive lock or getting caught in deadlock situations. We need to take a look at the unique locking issues to the data sharing environment as well as revisit some best practices for minimizing locking problems.

Data sharing locks include all the traditional locks we are familiar with but it also introducing some new locking mechanisms. These are modified locks, retained locks, P-locks (Subsystem or physical locks) and L-Locks (Transaction or logical locks).

P-Locks

Page Set P-Locks are used to track intersystem interest between DB2 members and determines when a page set becomes GBP dependent. These locks will have different modes depending on level of R/W interest on the pageset among the DB2 members. A P-Lock cannot be negotiated if it is retained. They are released when page set or partitioned dataset is closed. There are few P-Locks taken, and they are usually held for long periods of time.

Page P-Locks are used to ensure physical consistency of a page when it is being modified – these work at subpage level and are used in the same manner as latches in a non-data sharing environment. P-Locks are also used when changes are being made to a GBP dependent space map page.

PageSet P-Lock negotiation takes place when P-Locks are noted as being incompatible. The two member with the incompatible lock will negotiate the lock so that both can still use the object. Since the P-Lock is used for coherency, not concurrency, this negotiation is not sacrificing any data integrity.

The reason for P-lock negotiation is to lessen the amount of locks that are propagated to the coupling facility. The most restrictive P-lock is taken first and then if necessary, negotiated so another process can have access to the pageset. Pageset P-locks are used to track interest in a pageset and know when it is necessary to being propagation of child locks because of the level of interest in the DB2 members for the pageset. Basically think of P-locks and negotiation can be thought of as ‘I need to know what you are doing, here is what I am doing, lets find a way to work together or do we have to take turns’. It can also be though of as an indicator.

If you have a good deal of sharing going on among the members you want to be careful because you can run out of engines available for P-Lock negotiation, thus causing waits in the applications. Although this is rather rare and with 500 available engines should not be a huge issue.

L-Locks

L-Locks, or Logical Locks, occur in both data sharing (can be local or global) and non-data sharing subsystems. These locks are transaction or program owned. They are non-negotiable locks and work like the normal locks in a single subsystem environment to serialize access to objects. L-Locks are controlled by IRLM of member and are held from update to commit. There are two types:

- Parent L-Lock - tablespace or partition level (page set)
 - Almost always propagated to find if conflict exists with another member
- Child L-Lock - table, data page or row
 - Based on parent L-Lock conflict. If no conflict, then not propagated

Modify Locks

Modified and retained locks are two more types of locks introduced in data sharing. Modified locks are used to identify a lock on a resource that is being shared (updated). An active X type(X, IX, SIX) P-lock or L-lock. This lock is kept in Modified Resource List in the Lock Structure of the coupling facility and is kept regardless of group bufferpool dependency of the object. These locks are used to create retained locks if the DB2 member holding the modify lock were to fail.

Retained Locks

Retained locks are modify locks that get converted to a retained locks if a member of the group fails. You can kind of think of a retained lock as the ‘captain going down with the ship’. They are necessary to preserve the data integrity in the event of a failure. This lock will be held when a DB2 subsystem fails and the locks belongs to the failing member and must be resolved before access to the locked object will be allowed by other members.

This can be a performance/availability bottleneck of proper procedures are not in place for recovering a failed DB2 member. These locks are held at the GLM level and owned by the LLM,

not a transaction. This means that only that DB2 member that had the lock can resolve it, so the subsystem MUST come up to resolve the lock. So, regardless of where a transaction may resume (I.e. another subsystem) the locks are still retained and the data is still not accessible by any process (readers using uncommitted read can still view the data). The DB2 can be restarted on the same MVS or another one in the same group – it does not matter, as long as it comes up. This is why ARM (Automatic Restart Manager) is so important. Each local IRLM keeps a local copy of retained locks for fast reference. So retained locks can survive a CF failure.

Yes, it is true that late in Version 5(via an APAR) we were given the ability to purge retained locks. If you are at all concerned with data integrity, DO NOT USE THIS! I really can only see this being used carefully in a test/development environment – NEVER in production.

Lock Avoidance and Data Sharing

Lock avoidance will still work with data sharing and is very beneficial, but it is only going to work if you design your application correctly. We know that the CLSN(Commit Log Sequence Number) only gets set when the write claim count goes to zero for a page set. So if you have an application that is not committing and the write claim count never goes to zero, then the CLSN does not get set, and as a result the PGLOGRBA on the page is never less than the CLSN and the PUNC bit is not checked and lock avoidance is not used. Therefore we are sending more locks to the coupling facility than necessary. But you may ask why this is any more of any issue that it is in a single subsystem environment. Well think about this.....

If you have several members in the same data sharing group manipulating the same data you could run into trouble if someone decides not to play nice. Suppose you have a highly tuned application that commits frequently as it should. But, little did you know that there is a renegade application in the group that has decided it does not feel like committing. In a data sharing environment the CLSN is global – GCLSN(Global Commit Log Sequence Number) . There is one for the entire group and the oldest unit of work wins! That means if an application is not committing, therefore not setting the GCLSN for a particular pageset, then that application could be directly affecting your performance because you will never get lock avoidance and will be driving unnecessary lock to the coupling facility. What to do??? Have strict standards for commit strategies, especially in a non-affinity data sharing environment.

Locking Contention

There are three types of locking contention that can occur in a data sharing environment:

- Global Lock Contention (IRLM/Real)
 - This contention occurs when there is real contention against two resources.
- False Lock Contention
 - This contention occurs when two lock hashes to the same entry in the lock table but the actual locks are not in contention.
- XES Contention

- This occurs because XES only interprets locks as X or S, therefore some locks that are in contention are actually compatible because they are really intent locks

There are tolerable levels of each that we must be aware of because when contention becomes excessive, transactions begin to suffer due to waits.

For global lock contention you can calculate the amount of global lock contention by using a Statistics Report to add up the number of synchronous XES requests (Lock Request+Change Requests+Unlock Requests) divided by the total amount of contention (IRLM Global Contention)+(XES Global Contention)+(False Contention). This total is then divided by the total number for request to XES times 100). You can also view this in the GLOBAL CONTENTION RATE field. The total should be under 2% and if it is not then the amount of locking needs to be reduced, especially the number of P-locks propagated to the coupling facility. Lock avoidance through application tuning can help in this situation. The goal is to keep total number of combined lock, change and unlock request from exceeding 98%.

With false contention a lock requester could be suspended until the determination of whether or not the contention is false is made. As it increases it can cause transactions to terminate and response time to increase. You can also calculate the false contention in the Statistics Report by taking the number of unlock requests and dividing it by the total number of request and dividing that number by 100. You can also view this number in the FALSE CONTENTION RATE field. The total needs to be kept under 50% to the calculated total global contention. If it is not there are a few options such as increasing the lock table – the more entries allowed in the lock table, this decreases the chance of two lock requests hashing to the same entry, decrease the granularity of the locks taken (i.e. table/tablespace locks), or decreasing the number of users connected to the lock structure, which is usually not an feasible option.

A statistics report will also show the XES contention, as will an RMF(Resource Management Facility) report. In order to reduce this type of contention you need a reduction in the amount of tablespace locking. The best help for this type of contention is to bind your packages with RELEASE(DEALLOCATE) and to drive thread reuse. By doing this, the intent locks on the tablespace will be held through a commit and will not get continually propagated to the coupling facility through XES who is going to interpret them as S and X, and allowing for the opportunity of XES contention to occur.

Row-Level Locking

There are legitimate reasons for the implementation of row-level locking but row-level locking in a data sharing environment will do nothing but add overhead to your transactions and burn CPU. Good news is – this problem can avoided.

Row level locking does have a greater negative effect on the coupling facility than page level locking. While it may appear that you should be taking the same 'number' of locks, in a data sharing environment this is a bit different. The locks most people think of are L-locks, the same as those that we use in a non-data sharing environment, but where the problems arise in data sharing, and coupling facility overhead, is with excessive P-locks(physical locks used to control page coherency, not transaction concurrency).

Here is a quick scenario....if you are locking row (rid 111) on page 7 in member DB2A, a P-lock is taken on the page. Then member DB2B wants a row lock on rid 112 (which is on the same page). DB2B will also want to get a p-lock on the page. This will cause the members to now perform p-lock negotiation, which is expensive and cause additional requests to be driven through the coupling facility. So if you are doing a lot of row level locking the overhead associated with taking P-locks on the PAGE and then having to negotiate these P-locks so that both members can have interest in the page where the rows resides will kill any benefit you may require by using row level locking. The coupling facility works at a page level, not a row level. So, by using MAXROWS=1 you are now working at the page level and you can get a lock on two different 'rows' from two different members without having to go through P-lock negotiation and also you will still get the concurrency benefits you are looking for.

Also, the CPU burned to implement row level locking can quickly drive up the need to buy more hardware to support your e-business applications.

The IRLM uses the same resources in memory and the same CPU usage for row locking as for page locking. It cost just as much to lock a row as it does to lock a page. If a program is updating several rows in the same page the application could suffer if you change from page locking to row locking. The more rows per page, the higher the cost. For example in the following table

LOCKSIZE	ELAPSED CLASS 2	CPU CLASS 2	LOCK REQUEST	MAX LOCKS HELD
PAGE	13.37	1.98	2	25
TABLE	12.89	2.08	660	607
ROW	21.68	3.27	23983	23826

you can see how much CPU overhead is involved in row level locking vs. page level locking. In this example a single dynamic SQL update statement updating 28326 rows (on a 50 MIP processor) is measured in terms of the overhead incurred if you were using table locking vs. page locking vs. row locking. Just by looking at the chart you can see that it takes more elapsed time and CPU time to execute the same SQL statement when using row level locking. Also by dividing the difference in CPU by the difference in lock request you can see the CPU cost for locking.

$$(3.27 - 2.08) / (23983 - 660) = 51$$

This shows the CPU averaging about 51 microseconds for the total lock operation (lock, change, unlock) or about 17 seconds for each individual operation. This can accumulate if you are updating several rows in the program.

Row level locking can also potentially *increase* your chances for contention and deadlocks. This can happen if you have two applications that are performing updates on the same rows of a page and these updates are not occurring in the same sequence. If you were using page level locks the

second application would have to wait to access the page (it may time out waiting, but if the commit scope is properly set this may not happen). If you are using row level locking the applications will be accessing the same page at the same time and may deadlock while attempting to access the same set of rows. In this example you have two programs (PRG1 and PRG2) accessing the data in two different sequences (PRG1 is more sequential and PRG2 is more random). This example shows what could happen if you are using row level locking. PRG1 wants to make updates to rows 1,5 and 7 – in this order, and then it will commit. PRG2 wants to make updates to rows 7,5 and 3 – in this order, and then commit. If these programs start about the same time a PRG1 gets its X lock on row 1 and 5, and then PRG2 gets its X lock on row 7, these programs will then deadlock as PRG1 tries to get a X lock on row 7 and PRG2 tries to get an X lock on row 5.

If you are using page level locking in this scenario. PRG1 will get the X lock on the page, makes its updates and then commits. PRG2 will wait until PRG1 has committed, get the X lock on the page and then makes its updates. PRG2 may or may not timeout, depending on the commit scope and the length of time specified for the system timeout period.

Bottom line here...do not implement row-level locking in a data sharing environment. Use MAXROWS=1 if this type of concurrency is need. So be careful when packaged applications are implemented as some are 'grown-up' from other platforms where this was preferred and the DDL may be putting something in your system that could be detrimental to your web transaction performance.

Coupling Facility

Data sharing overhead is directly related to the interaction with structures that reside in the coupling facility which can be separate piece of hardware, or an ICF (Internal Coupling Facility) or a separate LPAR. It is worth the time and effort to be sure that the coupling facility is configured correctly and optimally for best performance.

The coupling facility is usually the one of the top items in this environment that significantly undersized in terms of storage. It is recommended that you have at least 1 GB on each coupling facility to start. Attempting to support a data sharing group with anything less can be very difficult, if not impossible.

You will need enough storage to hold all of the necessary structures and enough room (empty) to hold the structures from the other coupling facility in the group in order to be able to rebuild all the necessary structures from a failing coupling facility, or a coupling facility that has been taken offline for maintenance.

We must also not forget to consider the speed and number of links to the coupling facility. Performance and availability can both be affected by having an inappropriate number of links from the coupling facility to the processors running on the operating system. It is recommended that you have two links from each coupling facility to each MVS. One link would give you a single path to buffers, lock structures and for communications, not to mention a single point of failure.

You want to keep your links around 40% busy and not overload them or performance will degrade. Each coupling facility link has two subchannels and two link buffers. If the links are overloaded then MVS will wait and retry operations until the link is cleared, this results in excess

overhead for operations using the coupling facility and can degrade application response time and overall performance. There are two types of links: Mutli-mode fiber(50/125 micron) links supporting distances up to 1km, and single-mode fiber(9/125 or 10/125 micron) links supporting distances up to 3km.

Just having enough coupling facilities and enough coupling facility links is not enough. Having enough coupling facility engines is just as critical. A shortage of coupling facility processing cycles will back up work in the complex. One physical machine can have more than one logical coupling facility, but each logical coupling facility needs to have dedicated processors. There should be no sharing of processors due to the fact that sharing processors degrades coupling facility service time to members. There is a simple rule to follow: DO NOT assign a single processor to more than one logical coupling facility. Logical coupling facilities cannot be higher in number than processors. If you have only 1 processors then you should only have 1 logical coupling facility. However you can of course have a greater number of processors than logical coupling facilities. The speed of the processors will vary by the model number of the coupling facility. For example: a 9674-C04 takes about 30 microseconds to handle a lock requests.

Coupling facility utilization should be monitored and should not be allowed to be above 40%. Keep in mind that the overhead for lock requests is not linear...once performance goes – its gone! There is no curve of ever decreasing performance, you just die.

We must also monitor the usage of the resources in the coupling facility. We can do this via the RMF (Resource Management Facility) reports. Some items to monitor are resource usage, link contention, structure activity, and service times.

Structure activity can be measured for all of the structures in the coupling facility by using the RMF report Coupling Facility Structure Activity report. You can monitor global and false contention, and directory reclaims for the group bufferpool structures. Resource usage can be observed in an RMF report under ‘Coupling Facility Usage Summary’ and look for the AVERAGE CF UTLITIZATION (%BUSY). It is important that for performance and availability reasons that this remain low. If you are constantly 30% (or greater) busy, you many want to consider upgrading your coupling facility. Remember that once a coupling facility is over 50% busy you will see a serious degradation in the performance of request to the coupling facility and generally this degradation is not a gradual decrease, it is an immediate and harsh. Coupling facility links (or channels) can get busy when a lot of request are being issued. You will need to monitor the links in particular for requests that are being delayed and also to see if contention on the links is getting too high. We need to monitor the Subchannel Activity section from an RMF report. Try to keep the number of TOTAL DELAYED REQUESTS below 10% for best performance. In order to help alleviate this type of channel contention, if it is present, you may want to consider reducing the amount of request over the links, in other words the amount of sharing between members on the same objects. This could be done by redistributing the workload possibly by using some affinity processing. You could also consider upgrading the link. Also, if you notice that the BUSY COUNT for the PTH is high you may want to evaluate the dedication of paths to the MVS subsystem because there is too much contention. You can keep an eye on service times for each of you structures through the Structure Activity section of the RMF report. The field SERV TIME(MIC) shows the average service time for the synchronous requests. For lock structure requests you want to keep the time below 150 microseconds(Will depend on your level of coupling facility hardware). For a group bufferpool

structure you would want to keep it under 250 microseconds. Since the requests for the SCA are so few don't sweat the request time here.

Geographically Dispersed Parallel Sysplex

While data sharing adds another dimensions to the world of large system processing, there is basically one piece missing and that is the ability to integrate multiple sites, especially for the purposes of disaster recovery. The GDPS (Geographically Dispersed Parallel Sysplex) was introduced by IBM as a way of managing systems across multiple sites for maximum availability, primarily for disaster recovery. GDPS is a very automated environment with enhanced system management functions, including disk configuration management and peer-to-peer device dynamic switching through extended use of remote copy software. The GDPS technology builds upon the existing Parallel Sysplex technology, remote copy technology, and exploitation of DB2 for OS/390. Planned and unplanned outages become less painful and less risky since GDPS provides data redundancy and site redundancy. With GDPS there is another level of redundancy over single site redundancy with 2 copies of data at 2 sites, providing both single-site and multi-site environment support for greater application availability.

Continuous operation and support for all enterprise wide computing is also a critical requirement. Most organizations cannot afford to lose more than a couple days of processing after a site disaster (some of our clients would be out of business if not back online within 24 hours). Depending on the plans for business segment processing after a disaster, full recovery may never be possible. The financial impact to customers varies by line of business and has been known to approach \$6 million per hour.

GDPS Implementation

There are costs associated with implementing a GDPS. The parallel sysplex environment will have to be replicated at both sites. The hardware in both environments must include at least one coupling facility, a sysplex timer, processor(s), and disk with remote mirroring. There are many prerequisites for establishing a site including operation system, disk subsystem support, communications, system automation software, and disk subsystem with FREEZE/RUN support. The data required for restart needs to be disk resident and mirrored. With all of the requirements listed, it is easy to see that the implementation of a GDPS is not an easy, or inexpensive, task.

The GDPS is a reality today. Results from a customer who has successfully implemented GDPS technology for unplanned site reconfigurations has seen a reduction in their recovery window from 12 hours to 22 minutes. Other customer's results will vary upon their respective workloads. One way to estimate the reduction in recovery time is to determine the restart time after a system failure. This is how fast restart will be and this can even be reduced by 10-15 minutes if stand-by, IPLed minimal OS/390 images are ready. Customers testing GDPS have also experienced the elimination of data loss during this reconfiguration with minimal operation intervention.

Current distance between GDPS sites has distance limitations. We cannot state them here as they constantly get increased with new technology, but they should well exceed the older 40 km limit that was due to the limited distances using synchronous protocols. For unlimited distances there was discussion about a possibility of providing a similar function using async protocols based on XRC technology. Eventually there will probably be support for all disk vendors and better support for continuous availability during reconfigurations through advanced technology.

GDPS Impacts On DB2 Data Sharing

With the limitation of all coupling facility access currently being synchronous, the distance between coupling facilities is limited. There is a planned conversion strategy to move the majority of messaging to the coupling facility from synchronous to asynchronous. Converting coupling facility synch accesses to async will reduce the performance degradation and allow for longer distances between Parallel Sysplex (GDPS) sites as well as improved data sharing performance.

DB2 Data Sharing supports some of the largest businesses, and their most mission critical systems. Disaster recovery planning for data sharing not only requires complex planning, it is difficult to stage a backup site recovery. This situation has given rise to many and varied strategies to allow for a potential loss of systems, but all of them require much effort and talent, not to mention down time. GDPS will allow the issue of disaster recovery difficulty to essentially vanish, providing the level of safe security that most firms are seeking today.

Conclusions

There are many organizations today that are using multiple member data sharing environments to support their high volume web applications. The key to pushing the high volumes through the system is not only the flexibility in terms of adding hardware, but also the ability tuning existing systems to get more out of them. Data sharing gives us a lot of power and flexibility to support these new environments.

Susan Lawson is a Principal Consultant with YL&A. She is an internationally recognized consultant, teacher and lecturer specializing in database performance, VLDBs and data warehouses. She was formerly an IBM Data Sharing advocate for the Santa Teresa Laboratory where she provided technical expertise for DB2 Data Sharing customers. She has co-authored two books on DB2, "DB2 Answers", published by Osborne-McGraw-Hill and "DB2 High Performance Design and Tuning", published by Prentice-Hall and can be reached at Susan_Lawson@YLAssoc.com.