DB2 for z/OS Best Practices
Log Activity

John J. Campbell
Distinguished Engineer
DB2 for z/OS Development
db2zinfo@us.ibm.com

Transcript of webcast

Slide 1 (00:00)

Hello, this is John Campbell here, a Distinguished Engineer in DB2 development. Welcome to another lecture in a series of DB2 for z/OS best practices. The subject of today's web lecture is log activity and then some other miscellaneous topics will be covered.

Slide 2 (00:18)

On slide 2 is a disclaimer about some of the trademarks used in this web lecture. Can we please now turn to slide 3.

Slide 3 (00:26)

So on slide 3, the first topic I'd like to talk about here is writing to the active log. And the first hanging question there is: When does DB2 write to the active log? And really there is three major categories. First of all UR-related events, unit of

recovery related events. And this can happen at the end of commit phase 1, at the begin of commit phase 2 when we have a two phase commit. And in a single phase commit, where we have phase 1 and phase 2 combined into a single phase. When we have a single phase commit, there is only one log write for one phase commit.

Now let's talk about database writes. DB2 is designed to use a write ahead logging protocol. What that means is, all undo-redo log records with change or page being updated must be hardened onto DASD before the page is written to DASD or the coupling facility. Another example of this write ahead logging protocol, is when we have page P-lock negotiation and index leaf page split in data sharing. Basically, before the negotiation is complete, and the requester can actually have the P-lock you must actually do forced writes to the log. Basically to expose the update to the other members or to the other member. And there are two forced physical log writes per index leaf page split. And when this happens, we force the log up to what is called the PGLOGRBA. The RBA or LRSN of the last update to the page.

And the third example of write activity to DB2 active log is when we take DB2 system checkpoints which occur based on time or based on number of log records since the last system checkpoint. We also take system checkpoints as well when we do an active log switch. Now let's turn to slide 4.

Slide 4 (02:16)

So continuing the theme of active log write here, when does the actual writes take place? Well first of all, we have what is called a log write threshold. And basically what happens is if we haven't written log records for a period of time or cer-

tain number of buffers, then basically we force the log records out or write the log records out I should say to the active log. And we have a log write threshold that previously was externalized prior to version 7 with a system ZPARM called write threshold (WRTHRSH). But since version 7 and above releases, we now have a fixed value for this log write threshold of 20 buffers. So if we haven't actually written any log CIs to the active log and after 20 buffers, then we will write these log records out to the active log. Records can also be forced out using an archive log command. Or also, when we have an IFI read request for IFICID 306 from another member in the data sharing group.

When the log writes actually occur, the can be changed writes, in other words we can write multiple log CIs per I/O. And in fact we can write up to 128 4K log CIs per write I/O. Now DB2 sometimes, when you have dual logging, DB2 sometimes will do the write serially. And sometimes they will be overlapped. So in a dual logging environment, basically, if you are writing CI for the first time, then the writes to the primary and secondary copy of the log are fully overlapped. However, with dual logging, if we are re-writing a log CI which is partially filled and written previously, then we write those CIs serially. So what this means basically, if you want to get fully parallel overlap of writes to log copy 1 or log copy 2, you have to have at least 4 log CIs to write. That's because the first log CI is a rewrite of one that was previously there. So that has to be done serially. Then CIs 2 and 3 can be written in parallel. And then we have the fourth CI and then go back to rewrites to copy 1 and copy 2 being serial.

There is a change, enhancement, in version 10 where we will always do fully overlapped writes to log copy 1 and log copy 2 even if we are rewriting a CI that has been previously written. Now let's turn to slide 5.

Slide 5 (04:44)

On slide 5 here we have what is titled Log Write Statistics in the top right hand corner we have an example here of the log activity section of the DB2 Statistics Report. Now we are going to focus here on two important statistics. First of all, the unavailable log output condition. And secondly the paging of a log output buffer. And there are two important rules of thumbs.

But first of all the size of the log output buffer, is controlled by a ZPARM called OUTBUF. And the log output buffer storage is allocated in the master address space. Now the two key rules of thumb are first of all: If the unavailable log output buffer condition is greater than zero, okay, and the actual logging volume is less than max that can be supported, then this is a very bad condition. Because when we have an unavailable log output buffer, all insert, update, and delete activity and other system activities are stopped until the condition is relieved. So if that condition occurs, one needs to look at actually increasing the size of the log output buffer to deal with the increased logging volume.

The second rule of thumb, is related to when the log output buffer starts to page. Page out to auxillary. And if this value is greater than say 5% of the log records created, this is a red zone condition. And if this occurs, then we need to reduce the over commitment of the available real storage. Now one action would be to in fact decrease the size of the log output buffer. But of course that has a risk with it, because it could then trigger the unavailable log output buffer condition. The other approach is to reduce things like buffer pool requirement so that we reduce the overall real storage requirement. Obviously more log output buffer space can

also help in log reads which we will talk about in a minute. If you actually want to calculate the log data volume, then from this report you can take the number of log CIs created per unit of time multiplied by 4 K which is the size of the log CI and then divide by the statistics interval. Basically in today's world with today's devices and channel technology you need to start paying attention when the actual logging volume gets above 10 megabytes per second. Now let's turn to slide 6.

Slide 6 (07:04)

Slide 6 is a pretty old chart now but I use it just for illustration purposes. And it is titled here the maximum reserve rate of active log write. But what it shows here is that technology in the last 5 to10 years both in terms of faster devices, faster technology, faster striping as well, instructions and the speed of it have had a dramatic affect in terms of supporting larger data volumes. So on the Y axis here we have megabytes per second that can be supported. And on the X axis here we have various combinations. So if you look back on the very old ECON channels and the mode E20 of the shark device, we max out at 8.2 megabytes per second. But you can see once you got to the DS8000, for example, without striping, you are able to go to 84 megabytes per second and at the time in which FICON technology, DS8000 two stripes we could go to 116 megabytes per second. So in this particular area of the physical log write, tremendous progress has been made and if you look at it as a staging point today, a point in time statement here. With today's FICON technology and the fastest devices here, even without striping, we can sustain very high logging rates. Now let's turn to slide 7.

Slide 7 (08:24)

Now let's talk about log data set I/O tuning. The first thing to tune for is to avoid I/O interference between active log read/write and the active log read/write process. In other words, you minimize the interference between active log write and basically active log archive process.

Secondly, if the logging rate is near the maximum that you can support, then what are the options in terms of tuning? One is you can move the data set to a faster logging device. You could probably consider the use of DFSMS and striping. And the third thing to do here is to reduce the log data volume. So how could you possible do that? First of all, if you have insert intensive rows, or say insert intensive tables with wide rows that compress well you should consider using DB2 data compression. DB2 data compression won't only reduce the size of the row inserted into the table. It will also reduce the size of the log records created. Another approach that could have been done prior to version 9, or if you are still using the basic row format, would be to optimize the table design to minimize the log record size. But this point is no longer applicable if you are using the reordered row format. And also be aware of the database descriptor update. If you have lots of DDL updates going on. Which affect databases which are large in size and therefore have a large database descriptor with many objects, then what you could do here is basically do things in terms of tuning the DDL itself, by doing multiple DDLs to the same database descriptor in the same unit of work. Or in fact, splitting of a large database into much smaller databases. And therefore each of these databases would have a smaller database description. Now let's turn on to the next slide.

Slide 8 (10:21)

Now on slide 8 here, we are now going to talk about log read activity. Log reads are driven by any of the following types of activity. First of all, the roll back of a unit of work. Secondary, data recovery where we are recovering forward on the RECOVER utility. Or using the new BACKOUT option in version 10. Where we are doing LPL and GRECP recovery with the START DATABASE command. When you get to the log apply phase of online REORG. Normal DB2 crash restart recovery. Also when an application or a monitor is using the IFI log read interface. Examples would be data replication technologies. Where IFCID 129 is used to read a range of log records CIs from the active log. Or using IFCID 306 to support reads from the archive logs, decompression, and so on.

Or using the stand-alone log read utility DSNJSLR as used by the DSN1LOGP service aid. So when the log reads occur, DB2 first of all tries to read to get the log records from the log output buffer and if they are no longer available in the log output buffer then he will go to the active log data sets. And if those log records have finally been archived away to the archive log data set, and no longer available in the active log data sets, then DB2 will read those log records from the archive log data sets. Now let's turn to slide 9.

Slide 9 (11:45)

When it comes to log read performance, there are some statistics there in the DB2 statistics trace. On slide 9 there, in the top right-hand corner, we have the log activity section from the DB2 statistics trace. What you can see here is that it tells you how many times reads were satisfied from the log output buffer, how many times it was done from the active log, and how many times it was done from the archive log. So you can see in this example here, that 21% of the time,

the log reads were retrieved from the log output buffer. And 78% of the time the log records were retrieved from the active log data set. And this is pretty good. Also in this example you see that zero reads were satisfied from the archive logs. Now this is a pretty good situation here, because it is those archive logs that have been migrated away to tape for example, then reading log records from the archive logs can be very expensive in terms of the elapsed time impact.

So some key points here about tuning. One is more output log buffer space may actually help log read performance and reduce the number of reads to the active and archive logs. When it comes to actually reading the active log data set, or the log data records, then reading from the active log is actually the best choice. Because we have prefetched the log CIs, we also get automatic I/O load balancing across copy 1 and copy 2 of the active log pair. VSAM striping can be enable for the active log pair. And it also reduces task switching.

When it comes to archive log reads, then archives on DASD perform better. And one of the main things to be aware of here is the use of tape destroys parallelism. So if you for example have multiple recover jobs, which all touch active log records that have now been written out to archives on tape, then all the recovery jobs will end up serializing. Because tape is fundamentally not serial [parallel]. And that also applies to use of virtual tape devices. A virtual tape device still looks to the operating system as a tape device and therefore there will be no parallel read. So basically also, tape requires serialization in data sharing between concurrent recovers on different members. And I'm restating that you cannot concurrently share tape devices across multiple jobs. Now let's turn to slide 10.

Slide 10 (14:11)

What I'm now going to do in the follow on slides here is talk about some miscellaneous activities.

Slide 11 (14:19)

First of all RID list processing. Now RID list processing was introduced way back in version 2 release 2 of DB2. And it can have a significant effect in terms of reducing I/O resource consumption. Elapsed time impact of doing random unclustered I/O. Now basically the section I'm showing here related to RID list processing, it gives you some negative thresholds like terminated no storage, terminated exceeded RDS limit, exceeded data manager limit. The problem is that when you get RID list failures for any of these reasons, this can cause unnecessary CPU resource consumption and also unnecessary I/O. Because what happens when DB2 has a RID list failure, DB2 reverts back to doing a tablespace scan. And that can be pretty expensive.

So let's look at two of the primary conditions. First of all terminated because you exceeded the DM limit. This is when the number of RID entries is greater than the physical limit which is approximately 26 million RIDs. Now the other case which is called terminated exceeded the RDS limit. This is when the number of RIDs that can fit into the guaranteed number of RID blocks is greater than the maximum limit. Which is basically 25% of the table size. So what are some of the common reasons for this? One is incomplete or inaccurate statistics. In other words, stale statistics or missing statistics. The use of the LIKE operator in SQL statements, and the use of host variables in static SQLs, or character markers in dynamic SQL for range predicates, things like between, greater than, or less than. The recommendation here

is to identify the offending applications and SQL statements using the accounting records or using IFCID 125. And trying to fix up what the shortfall is. Now let's turn to slide 12.

Slide 12 (16:10)

The next topic in this miscellaneous section is phantom or orphaned trace. Now this is a condition that existed prior to version 9. This is the situation where IFI records are created for an online monitor or some sort of application program. But the records are not actually written and consumed. So basically a phantom or orphaned trace record are created by monitor and somehow the monitor has been stopped or gone away but with the corresponding trace is still being active. And this has the same CPU overhead as per a real trace. You can use the DB2 DISPLAY TRACE command to check for orphaned traces. And the good news starting with DB2 version 9 in conversion mode, is that we try to eliminate orphaned trace records. Now let's turn to slide 13.

Slide 13 (16:58)

Now let's talk about a pretty common problem that I have observed. And this is related to the package list or PKLIST search. Again in the top right corner here, I show an extract from a DB2 statistics report related to plan and package processing. And I've highlighted two rows here in this section. First of all package allocation attempt. And then package allocation success. Now obviously the perfect situation here is the number of successes is equal to the number of attempts. But in some customer situations, there can be a big gap between the number of attempts and the number of successes. Now why could this be?

Well here is the problem or the potential cause of it. On the PKLIST for the plan, you can specify multiple collections like COL A dot asterisk, COL B dot asterisk, COL C dot asterisk. And unless you are using things like SET CURRENT PACKAGESET, for example. Or SET CURRENT PACKAGEPATH, then what DB2 will do is search each of these collections serially until it finds a match. So for example, if you have lots of collections on the PKLIST, and they are not ordered based on frequency of access, then it might well be that DB2 has to probe each of these collections using index access before it finds a hit. And this is why there can be a big gap to the number of package attempts, allocation attempts, and the number of successes.

So the first thing to do here is calculate success rate. Now you can do that by dividing through the number of package allocation success by the package allocation attempts multiplied by 100. So basically, if you have a long PK search here, this can drive up CPU consumption, catalog accesses, and elapsed time. And it can also seriously aggravate sometimes DB2 internal latch contention, latch class 32, which is the storage manager latch.

So I have some good recommendations here. First of all, try to reduce the absolute number of collections on the PKLIST. Scrub out all inactive or unused collections. Try and fold in and collapsing of collections. The second recommendation if you have multiple collections still surviving, is to ruthlessly prioritize and reorder the collection sequence on the PKLIST based on frequency of access. And of course the best solution of all, if it is applicable, is to use something like SET CURRENT PACKAGESET or SET CURRENT PACKAGEPATH special register to direct the package search to a specific collection on the PKLIST. Now let’s turn to slide 14.

Slide 14 (19:33)

Now let's talk about disabled SPROCs. What is SPROC all about? SPROC is related to fast column processing. We have SPROCs for SELECTs for example, IPROCs for insert and so on. But SPROCs are generated for static SQL as part of the bind processing. Now the thing with fast common processing and SPROCs, is that basically when you migrate from version 7 to version 8 or 8 to 9 or even to 10, the existing packages and plans that have got SELECT statements and SPROCs, these SPROCs are invalidated. And basically the performance advantage from having prebuilt these SPROCs into the package goes away. And the typical CPU impact here could be from zero up to 10% in a very worse case.

So one of the things to do when you are running smoothly on the new release, is to identify all those plans and packages, particularly packages here, which are using basically SPROCs or have SELECT statements. And basically to rebind those packages under the new release. And by binding those packages, under the new release, the SPROCs will be regenerated. The question is: How do you identify those packages which basically have invalid SPROCs? What you can do here is look in the miscellaneous section. And in the example you can see BYPASS COL value is 1585. So basically when you end up with a non-zero value, this indicates situations where we have SPROCs that are invalidated. Furthermore you can use IFCID 225 [224] to identify plans and packages that need rebinding to re-enable the SPROCs. Now let's turn to slide 15.

Slide 15 (21:20)

The next topic here is incremental bind. Incremental bind is different to automatic bind or normal bind activity. When you have an incremental bind, we go through the bind process but we don't make the result of the bind operation persistent. So that means we repeat the process again and again. So that is what an incremental bind is. And there are activities that can cause incremental binds. As identified by this example from the statistics report. First of all we can have a static plan or package, with validate run, and where there are bind time failures. We can have static SQL with REOPT (VARS). It always occurs for DDF private protocol, on the requester side. It also occurs when you have SQL referencing Declared Global Temporary Table and possibly for some DDL statements. So, incremental bind can be very expensive. And for high volume applications, as I said the results of an incremental bind are not persistent and therefore you are paying the cost again and again as the application executes. So the thing to do here is identify the culprits and get them fixed up. Now let's go to slide 16.

Slide 16 (22:28)

What I would like to do here also now is advertise IFCID 199. IFCID 199 is collected by Statistics trace class 8. One of the Statistics trace classes. Now this is a very simple IFCID. But it does give you some pretty useful information here. So what you can see here for each of the objects, it identifies the bufferpool, the database, the tablespace name and the part, whether it is group bufferpool dependent or not, and it gives you information about whether the I/O average times are synchronous or asynchronous. Tells you what the delays are. Gives you information about the current number of pages in the bufferpool, number of get pages. So basically, we write information out for each object that does more than one I/O per second as an average during the statistics

interval and we actually block these data sets, or information about these data sets, into blocks of 50, into IFCID 199. So this provides very useful tuning information and gives you great insight. For example, in a batch application, you may have an elapsed time problem caused by synchronous I/O. So you can use this IFCID 199 information to identify those objects which have contributing to this synchronous I/O delays.

So that completes this web lecture in the series and I would like to thank you very much for your attention in running through this web lecture.

(23:53)